

UNIT I

Fundamentals of Data Mining:

Data mining refers to extracting or mining knowledge from large amounts of data. The term is actually a misnomer. Thus, data mining should have been more appropriately named as knowledge mining which emphasis on mining from large amounts of data.

It is the computational process of discovering patterns in large datasets involving methods at the intersection of artificial intelligence, machine learning, statistics, and database systems.

The overall goal of the data mining process is to extract information from a data set and transform it into an understandable structure for further use.

The key properties of data mining are

- Automatic discovery of patterns
- Prediction of likely outcomes
- Creation of actionable information
- Focus on large datasets and databases

The Scope of Data Mining

Data mining derives its name from the similarities between searching for valuable business information in a large database — for example, finding linked products in gigabytes of store scanner data — and mining a mountain for a vein of valuable ore. Both processes require either sifting through an immense amount of material, or intelligently probing it to find exactly where the value resides.

Given databases of sufficient size and quality, data mining technology can generate new business opportunities by providing these capabilities:

Automated prediction of trends and behaviors. Data mining automates the process of finding predictive information in large databases. Questions that traditionally required extensive hands-on analysis can now be answered directly from the data — quickly.

A typical example of a predictive problem is targeted marketing. Data mining uses data on past future mailings. Other predictive problems include forecasting bankruptcy and other forms of default, and identifying segments of a population likely to respond similarly to given events.

Automated discovery of previously unknown patterns.

Data mining tools sweep through databases and identify previously hidden patterns in one step. An example of pattern discovery is the analysis of retail sales data to identify seemingly unrelated products that are often purchased together. Other pattern discovery problems include detecting fraudulent credit card transactions and identifying anomalous data that could represent data entry keying errors.

Data Mining Functionalities:

We have observed various types of database and information repositories on which data mining can be performed. Let us now we examine the kinds of data patterns that can be mined. Data mining functionalities are used to specify the kind of patterns to be found in data mining tasks. In general, data mining tasks can be classified into two categories: descriptive and predictive. Descriptive mining tasks characterize the general properties of the data in the database. Predictive mining tasks performing reference on the current data in order to make predictions.

In some cases, users may have no idea regarding what kinds of patterns in their data may be interesting, and hence may like to search for several different kinds of patterns in parallel. Thus it is important to have a data mining system that can mine multiple kinds of patterns to accommodate different user expectations or applications. Furthermore, data mining systems should be able to discover patterns at various granularity (i.e., different levels of abstraction). Data mining systems should also allow users to specify hints to guide or focus the search for interesting patterns. Because some patterns may not hold for all of the data in the database, a measure of certainty or “trustworthiness” is usually associated with each discovered pattern.

Data mining functionalities, and the kinds of patterns they can discover, are described Mining Frequent Patterns, Associations, and Correlations Frequent patterns, as the name suggests, are patterns that occur frequently in data. There are many kinds of frequent patterns, including itemsets, subsequences, and substructures.

A frequent itemset typically refers to a set of items that frequently appear together in a transactional dataset, such as milk and bread. A frequently occurring subsequence, such as the pattern that customers tend to purchase first a PC, followed by a digital camera, and then a memory card, is a (frequent) sequential pattern. A substructure can

refer to different structural forms, such as graphs, trees, or lattices, which may be combined with itemsets or subsequences. If a substructure occurs frequently, it is called a (frequent) structured pattern. Mining frequent patterns leads to the discovery of interesting associations and correlations within data.

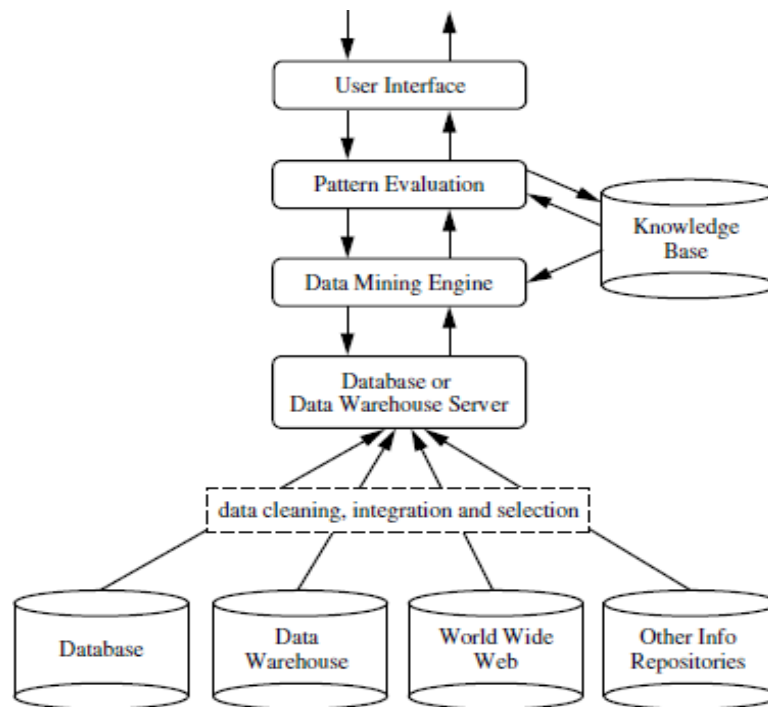
below.

Data mining involves six common classes of tasks:

- **Anomaly detection (Outlier/change/deviation detection)** – The identification of unusual data records, that might be interesting or data errors that require further investigation.
- **Association rule learning (Dependency modelling)** – Searches for relationships between variables. For example a supermarket might gather data on customer purchasing habits. Using association rule learning, the supermarket can determine which products are frequently bought together and use this information for marketing purposes. This is sometimes referred to as market basket analysis.
- **Clustering** – is the task of discovering group and structures in the data that are in some way or another "similar", without using known structures in the data.
- **Classification** – is the task of generalizing known structure to apply to new data. For example, an e-mail program might attempt to classify an e-mail as "legitimate" or as "spam".
- **Regression** – attempts to find a function which models the data with the least error.
- **Summarization** – providing a more compact representation of the dataset, including Visualization and report generation.

Architecture of Data Mining

A typical data mining system may have the following major components.



1. Knowledge Base:

This is the domain knowledge that is used to guide the search or evaluate the interestingness of resulting patterns. Such knowledge can include concept hierarchies, used to organize attributes or attribute values into different levels of abstraction. Knowledge such as user beliefs, which can be used to assess a pattern's interestingness based on its unexpectedness, may also be included. Other examples of domain knowledge are additional interestingness constraints or thresholds, and metadata (e.g., describing data from multiple heterogeneous sources).

2. DataMiningEngine:

This is essential to the data mining system and ideally consists of a set of functional modules for tasks such as characterization, association and correlation analysis, classification,

prediction, cluster analysis, outlier analysis, and evolution analysis.

3. Pattern Evaluation Module:

This component typically employs interestingness measures and interacts with the data mining modules so as to focus the search toward interesting patterns. It may use interestingness thresholds to filter out discovered patterns. Alternatively, the pattern evaluation module may be integrated with the mining module, depending on the implementation of the data mining method used. For efficient data mining, it is highly recommended to push the evaluation of pattern interestingness as deep as possible into the mining process as to confine the search to only the interesting patterns.

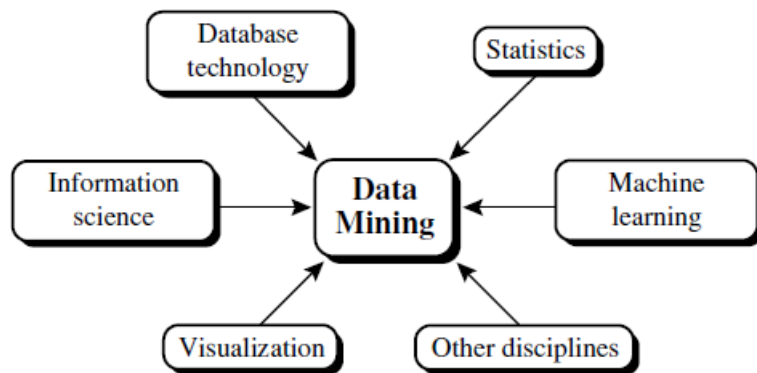
4. User Interface:

This module communicates between users and the data mining system, allowing the user to interact with the system by specifying a data mining query or task, providing information to help focus the search, and performing exploratory data mining based on the intermediate data mining results. In addition, this component allows the user to browse database and data warehouse schemas or data structures, evaluate mined patterns, and visualize the patterns in different forms.

Classification of Data Mining Systems

Data mining is an interdisciplinary field, the confluence of a set of disciplines, including database systems, statistics, machine learning, visualization, and information science. Moreover, depending on the data mining approach used, techniques from other disciplines may be applied, such as neural networks, fuzzy and/or rough set theory, knowledge representation, inductive logic programming, or high-performance computing. Depending on the kinds of data to be mined or on the given data mining application, the data mining system may also integrate techniques from spatial data analysis, information retrieval, pattern recognition, image analysis, signal processing, computer graphics,

Web technology, economics, business, bioinformatics, or psychology. Because of the diversity of disciplines contributing to data mining, data mining research is expected to generate a large variety of data mining systems. Therefore, it is necessary to provide a clear classification of data mining systems, which may help potential users distinguish between such systems and identify those that best match their needs.



Data mining systems can be categorized according to various criteria, as follows:

Classification according to the kinds of databases mined: A data mining system can be classified according to the kinds of databases mined. Database systems can be classified according to different criteria (such as data models, or the types of data or applications involved), each of which may require its own data mining technique. Data mining systems can therefore be classified accordingly.

For instance, if classifying according to data models, we may have a relational, transactional, object-relational, or data warehouse mining system. If classifying according to the special types of data handled, we may have a spatial, time-series, text, stream data, multimedia data mining system, or a World Wide Web mining system.

Classification according to the kinds of knowledge mined: Data mining systems can be categorized according to the kinds of knowledge they mine, that is, based on data mining functionalities, such as characterization, discrimination, association and correlation analysis, classification, prediction, clustering, outlier analysis, and evolution analysis. A comprehensive data mining system usually provides multiple and/or integrated data mining functionalities.

Moreover, data mining systems can be distinguished based on the granularity or levels of abstraction of the knowledge mined, including generalized knowledge (at a high level of abstraction), primitive-level knowledge (at a raw data level), or knowledge at multiple levels (considering several levels of abstraction). An advanced data mining system should facilitate the discovery of knowledge at multiple levels of abstraction.

Data mining systems can also be categorized as those that mine data regularities (commonly occurring patterns) versus those that mine data irregularities (such as exceptions, or outliers). In general, concept description, association and correlation analysis, classification, prediction, and clustering mine data regularities, rejecting outliers as noise. These methods may also help detect outliers.

Classification according to the kinds of techniques utilized: Data mining systems can be categorized according to the underlying data mining techniques employed. These techniques can be described according to the degree of user interaction involved (e.g., autonomous systems, interactive exploratory systems, query-driven systems) or the methods of data analysis employed (e.g., database-oriented or data warehouse-oriented techniques, machine learning, statistics, visualization, pattern recognition, neural networks, and so on). A sophisticated data mining system will often adopt multiple data mining techniques or work out an effective, integrated technique that combines the merits of a few individual approaches.

Classification according to the applications adapted: Data mining systems can also be categorized according to the applications they adapt. For example, data mining systems may be tailored specifically for finance, telecommunications, DNA, stock markets, e-mail, and so on. Different applications often require the integration of application-specific methods. Therefore, a generic, all-purpose data mining system may not fit domain-specific mining tasks.

Data Mining Process:

Data Mining is a process of discovering various models, summaries, and derived values from a given collection of data.

The general experimental procedure adapted to data-mining problems involves the following steps:

1. State the problem and formulate the hypothesis

Most data-based modeling studies are performed in a particular application domain. Hence, domain-specific knowledge and experience are usually necessary in order to come up with a meaningful problem statement. Unfortunately, many application studies tend to focus on the data-mining technique at the expense of a clear problem statement. In this step, a modeler usually specifies a set of variables for the unknown dependency and, if possible, a general form of this dependency as an initial hypothesis. There may be several hypotheses formulated for a single problem at this stage.

The first step requires the combined expertise of an application domain and a data-mining model. In practice, it usually means a close interaction between the data-mining expert and the application expert. In successful data-mining applications, this cooperation does not stop in the initial phase; it continues during the entire data-mining process.

2. Collect the data

This step is concerned with how the data are generated and collected. In general, there are two distinct possibilities. The first is when the data-generation process is under the control of an expert (modeler): this approach is known as a designed experiment.

The second possibility is when the expert cannot influence the data-generation process: this is known as the observational approach. An observational setting, namely, random data generation, is assumed in most data-mining applications.

Typically, the sampling distribution is completely unknown after data are collected, or it is partially and implicitly given in the data-collection procedure. It is very important, however, to understand how data collection affects its theoretical distribution, since such a priori knowledge can be very useful for modeling and, later, for the final interpretation of results. Also, it is important to make sure that the data used for estimating a model and the data used later for testing and applying a model come from the same, unknown, sampling distribution. If this is not the case, the estimated model cannot be successfully used in a final application of the results.

Major Issues In Data Mining:

- **Mining different kinds of knowledge in databases.** - The need of different users is not the same. And Different user may be interested in different kind of knowledge. Therefore it is necessary for data mining to cover broad range of knowledge discovery task.
- **Interactive mining of knowledge at multiple levels of abstraction.** - The data mining process needs to be interactive because it allows users to focus the search for patterns, providing and refining data mining requests based on returned results.
- **Incorporation of background knowledge.** - To guide discovery process and to express the discovered patterns, the background knowledge can be used. Background knowledge may be used to express the discovered patterns not only in concise terms but at multiple level of abstraction.

● **Data mining query languages and ad hoc data mining.** - Data Mining Query language that allows the user to describe ad hoc mining tasks, should be integrated with a data warehouse query language and optimized for efficient and flexible data mining.

● **Presentation and visualization of data mining results.** - Once the patterns are discovered it needs to be expressed in high level languages, visual representations. These representations should be easily understandable by the users.

● **Handling noisy or incomplete data.** - The data cleaning methods are required that can handle the noise, incomplete objects while mining the data regularities. If data cleaning methods are not there then the accuracy of the discovered patterns will be poor.

● **Pattern evaluation.** - It refers to interestingness of the problem. The patterns discovered should be interesting because either they represent common knowledge or lack novelty.

● **Efficiency and scalability of data mining algorithms.** - In order to effectively extract the information from huge amount of data in databases, data mining algorithm must be efficient and scalable.

- **Parallel, distributed, and incremental mining algorithms.** - The factors such as huge size of databases, wide distribution of data, and complexity of data mining methods motivate the development of parallel and distributed data mining algorithms. These algorithms divide the data into partitions which is further processed parallel. Then the results from the partitions are merged. The incremental algorithms, updates the databases without having to mine the data again from the scratch.

Data Integration:

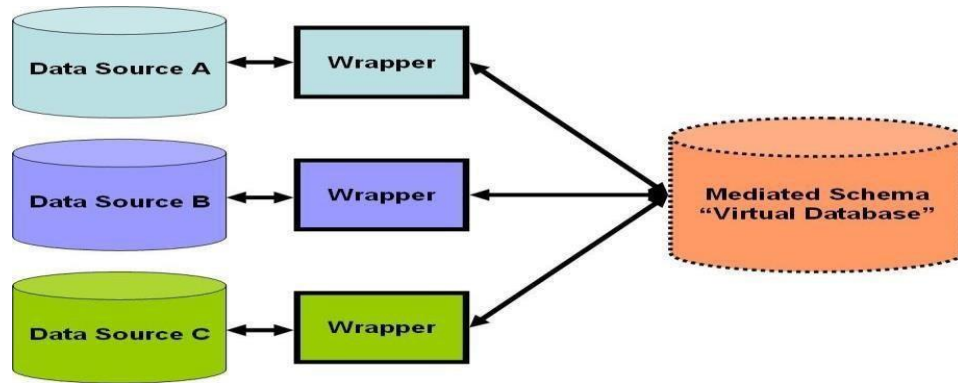
It combines data from multiple sources into a coherent data store, as in data warehousing. These sources may include multiple databases, data cubes, or flat files.

The data integration systems are formally defined as a triple $\langle G, S, M \rangle$

Where G: The global schema

S: Heterogeneous source of schemas

M: Mapping between the queries of source and global schema



Issues in Data Integration:

1. Schema integration and object matching:

How can the data analyst or the computer be sure that customer id in one database and customer number in another reference to the same attribute.

2. Redundancy:

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.

3. Detection and resolution of data value conflicts:

For the same real-world entity, attribute values from different sources may differ.

Data Transformation:

In data transformation, the data are retransformed 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.
- **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:

- **Data cube aggregation**, where aggregation operations are applied to the data in the construction of a data cube.
- **Attribute subset selection**, where irrelevant, weakly relevant, or redundant attributes or dimensions may be detected and removed.
- **Dimensionality reduction**, where encoding mechanisms are used to reduce the dataset size.
- **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.
- **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 Preprocessing:

In the observational setting, data are usually "collected" from the existing databases, data warehouses, and data marts. Data preprocessing usually includes at least two common tasks:

1. Outlier detection (and removal) – Outliers are unusual data values that are not consistent with most observations. Commonly, outliers result from measurement errors, coding and recording errors, and, sometimes, are natural, abnormal values. Such non representative samples can seriously affect the model produced later. There are two strategies for dealing with outliers:

- a. Detect and eventually remove outliers as a part of the preprocessing phase, or
- b. Develop robust modeling methods that are insensitive to outliers.

2. Scaling, encoding, and selecting features –

Data preprocessing includes several steps such as variable scaling and different types of encoding. For example, one feature with the range $[0, 1]$ and the other with the range $[-100, 1000]$ will not have the same weights in the applied technique; they will also influence the final data-mining results differently. Therefore, it is recommended to scale them and bring both features to the same weight for further analysis.

Also, application-specific encoding methods usually achieve dimensionality reduction by providing a smaller number of informative features for subsequent data modeling. These two classes of preprocessing tasks are only illustrative examples of a large spectrum of preprocessing activities in a data-mining process.

Data-preprocessing steps should not be considered completely independent from other data-mining phases. In every iteration of the data-mining process, all activities, together, could define new and improved data sets for subsequent iterations.

Generally, a good preprocessing method provides an optimal representation for a data-mining

technique by incorporating a priori knowledge in the form of application-specific scaling and encoding.

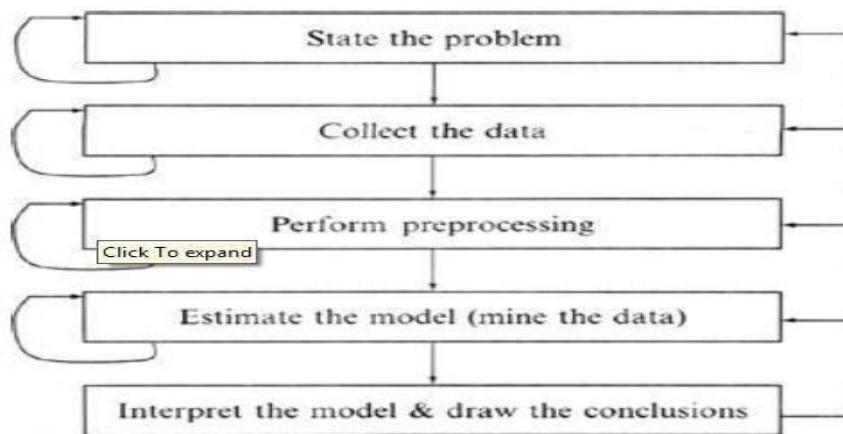
4. Estimate the model

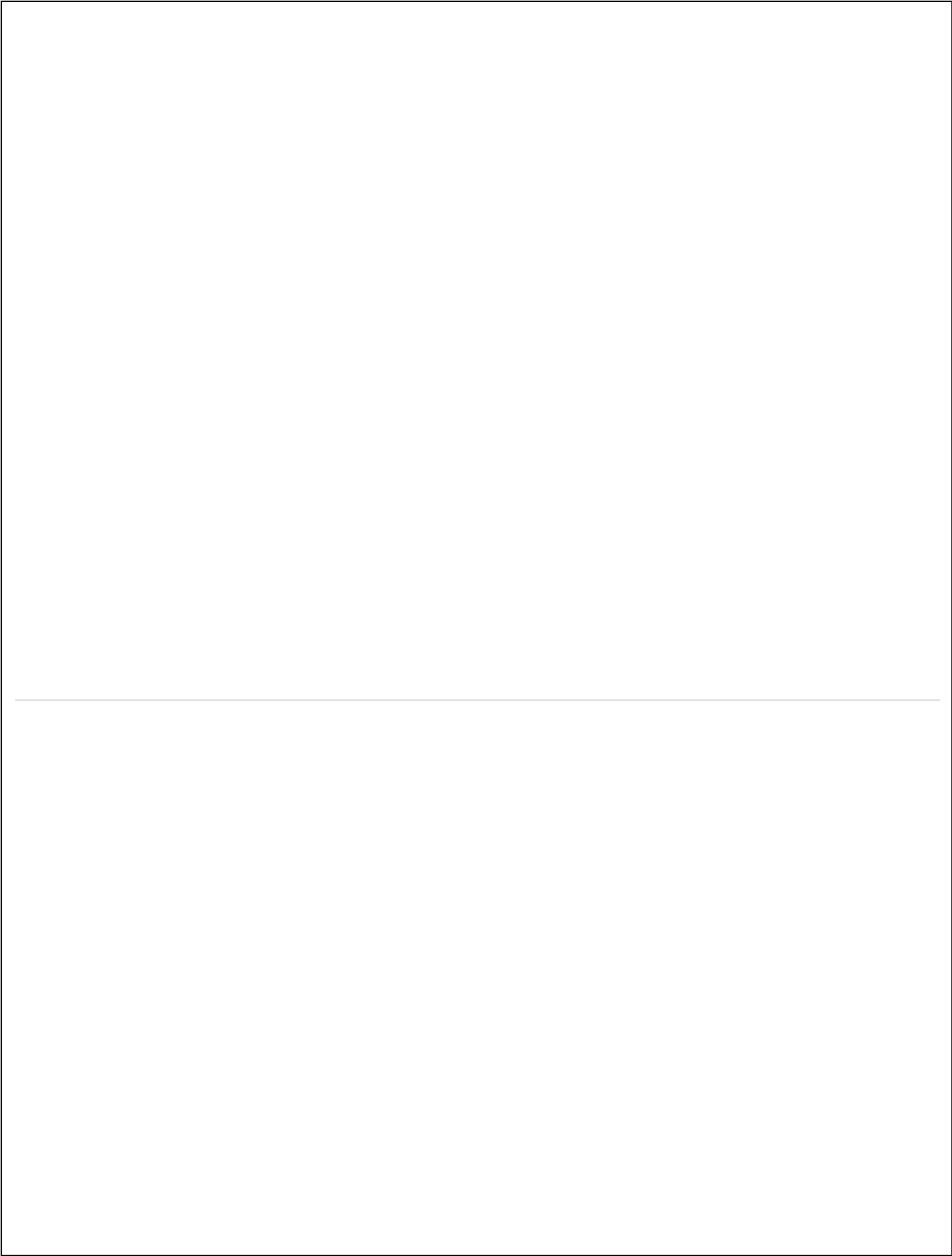
This process is not straightforward; usually, in practice, the implementation is based on several models, and selecting the best one is an additional task. Interpret the model and draw conclusions. In most cases, data-mining models should help in decision making. Hence, such models need to be interpretable in order to be useful because humans are not likely to base their decisions on complex "black-box" models. Note that the goals of accuracy of the model and accuracy of its interpretation are somewhat contradictory.

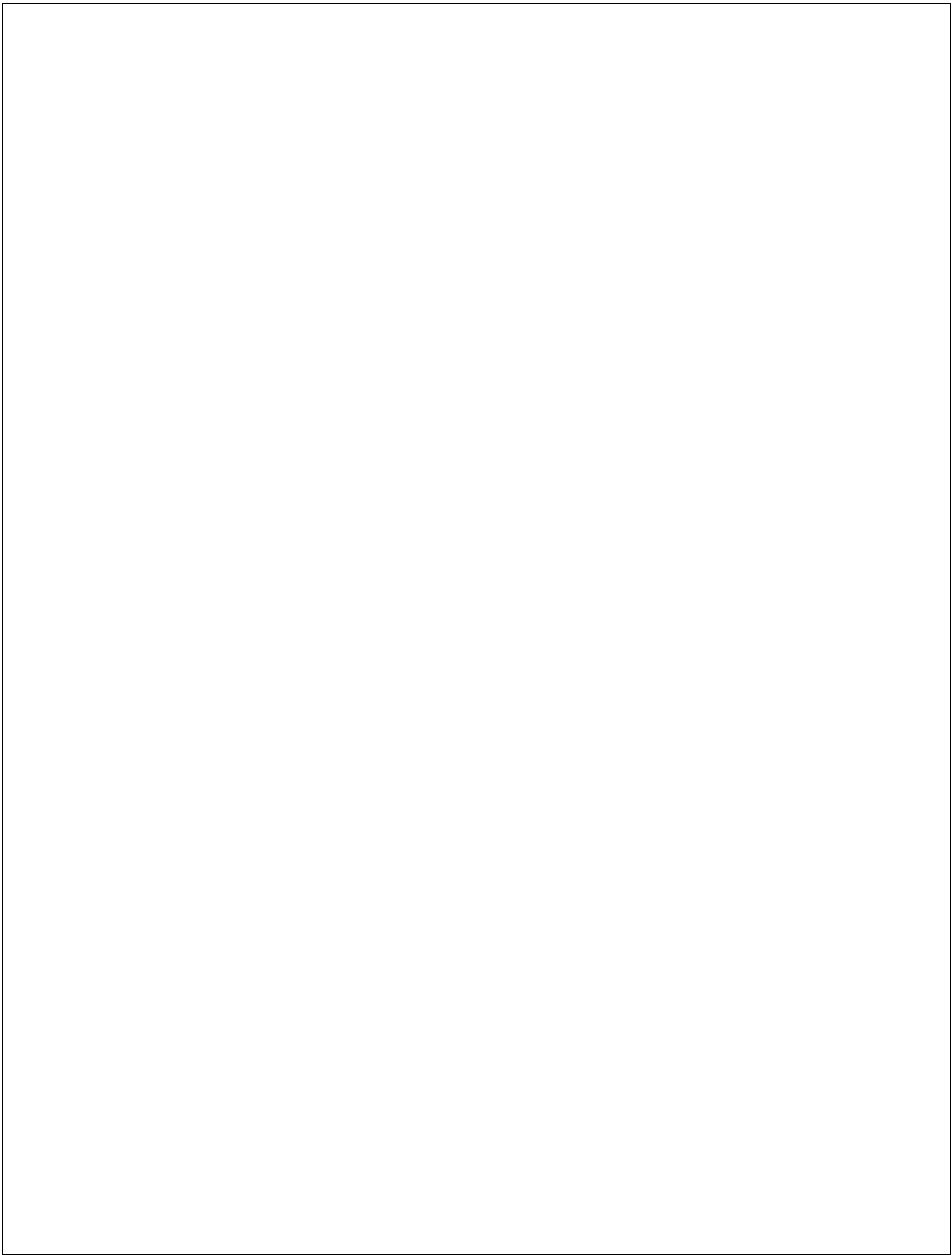
Usually, simple models are more interpretable, but they are also less accurate. Modern data-mining methods are expected to yield highly accurate results using high dimensional models.

The problem of interpreting these models, also very important, is considered a separate task, with specific techniques to validate the results.

A user does not want hundreds of pages of numeric results. He does not understand them; he cannot summarize, interpret, and use them for successful decision making.







Unit-II

Contents

- Statistical Measures
- Measuring Central Tendency
- Measuring Dispersion
- Mining Association Rules

Statistical Measures

Basics of Statistics

Definition: Science of collection, presentation, analysis, and reasonable interpretation of data.

- ▶ Statistics presents a rigorous scientific method for gaining insight into data.
- ▶ For example, suppose we measure the weight of 100 patients in a study. With so many measurements, simply looking at the data fails to provide an informative account.
- ▶ However statistics can give an instant overall picture of data based on graphical presentation or numerical summarization irrespective to the number of data points.
- ▶ Besides data summarization, another important task of statistics is to make inference and predict relations of variables.

What is Data?

Definition: Facts or figures, which are numerical or otherwise, collected with a definite purpose are called *data*.

- ▶ Everyday we come across a lot of information in the form of facts, numerical figures, tables, graphs, etc.
- ▶ These are provided by newspapers, televisions, magazines and other means of communication.
- ▶ These may relate to cricket batting or bowling averages, profits of a company, temperatures of cities, expenditures in various sectors of a five year plan, polling results, and so on.
- ▶ These facts or figures, which are numerical or otherwise, collected with a definite purpose are called *data*.

Gender
(Women,
Men)

Hair color
(Blonde,
Brown)

Ethnicity
(Hispanic,
Asian)

First,
second
and third

Letter
grades: A,
B, C,

Economic
status: low,
medium

NOMINAL DATA

ORDINAL DATA

QUALITATIVE DATA

Types Of Data

QUANTITATIVE DATA

DISCRETE DATA

CONTINUOUS DATA

The
number of
students
in a class

The
number of
workers in
a company

The number
of home runs
in a baseball
game

The
height of
children

The square
footage of a
two-bedroom
house

The speed of
cars

Primary Data Vs Secondary Data

Primary Data

- ▶ Primary data is the data that is collected for the first time through personal experiences or evidence, particularly for research.
- ▶ It is also described as raw data or first-hand information.
- ▶ The mode of assembling the information is costly.
- ▶ The data is mostly collected through observations, physical testing, mailed questionnaires, surveys, personal interviews, telephonic interviews, case studies, and focus groups, etc.

Primary Data Vs Secondary Data

Secondary Data

- ▶ Secondary data is a second-hand data that is already collected and recorded by some researchers for their purpose, and not for the current research problem.
- ▶ It is accessible in the form of data collected from different sources such as government publications, censuses, internal records of the organisation, books, journal articles, websites and reports, etc.

Primary Data Vs Secondary Data

▶ **Secondary Data**

- ▶ This method of gathering data is affordable, readily available, and saves cost and time.
- ▶ However, the one disadvantage is that the information assembled is for some other purpose and may not meet the present research purpose or may not be accurate.

Discrete Vs continuous data

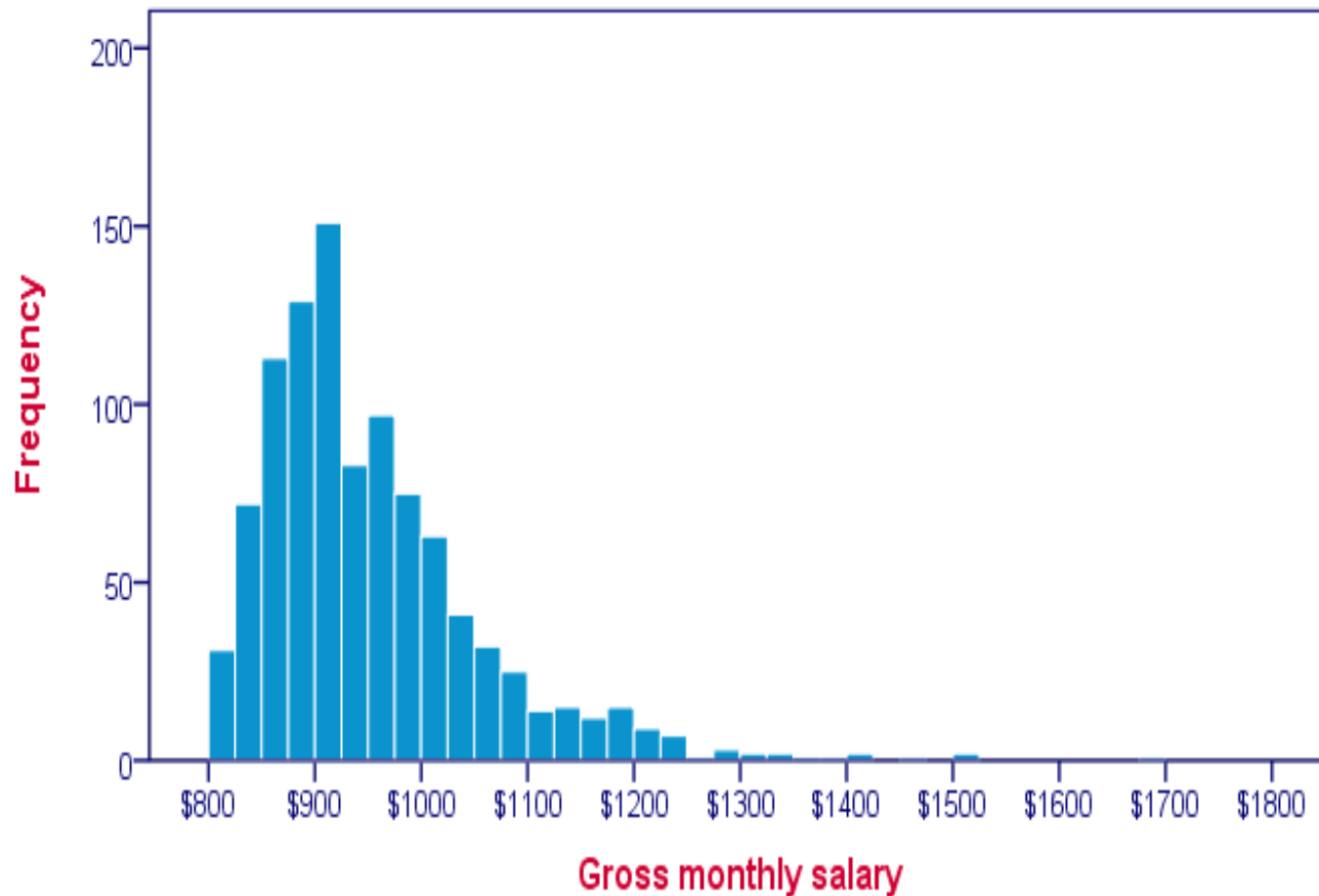
- Discrete data (countable) is information that can only take certain values. These values don't have to be whole numbers but they are fixed values – such as shoe size, number of teeth, number of kids, etc.
- Discrete data includes discrete variables that are finite, numeric, countable, and non-negative integers (5, 10, 15, and so on).
- Continuous data (measurable) is data that can take any value. Height, weight, temperature and length are all examples of continuous data.
- Continuous data changes over time and can have different values at different time intervals

Histogram

- A **histogram** is a graphical display of data using bars of different heights. In a **histogram**, each bar groups numbers into ranges. Taller bars show that more data falls in that range. A **histogram** displays the shape and spread of continuous sample data

Histogram

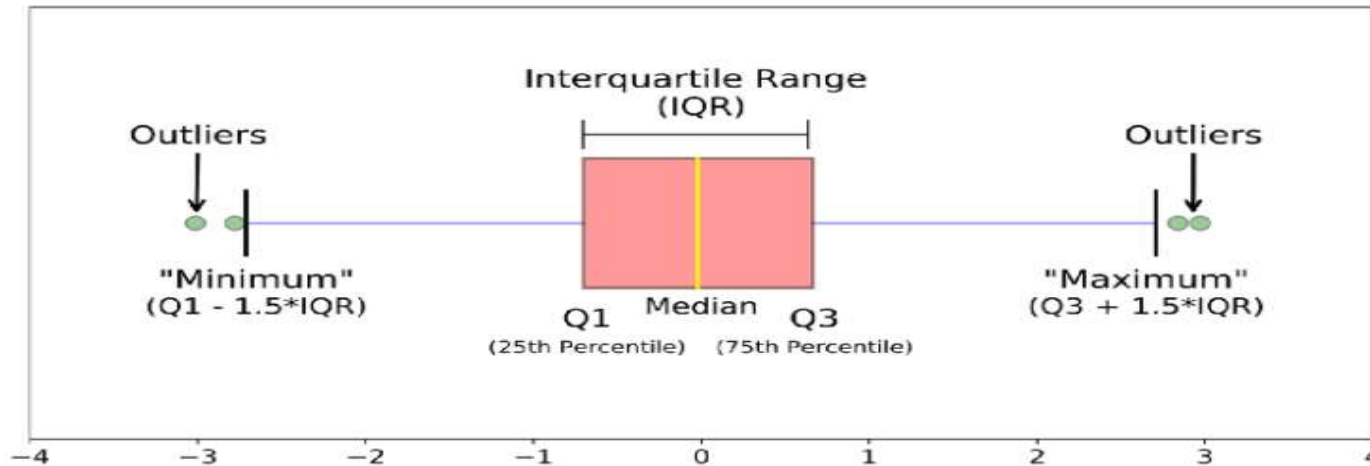
Histogram of Monthly Salary



Box Plotting

- **Box plots** (also called **box-and-whisker plots** or **box-whisker plots**) give a good graphical image of the concentration of the data.
- They also show how far the extreme values are from most of the data.
- A box plot is constructed from five values: the minimum value, the first quartile, the median, the third quartile, and the maximum value.

Box Plotting



The image above is a boxplot. A boxplot is a standardized way of displaying the distribution of data based on a five number summary (“minimum”, first quartile (Q1), median, third quartile (Q3), and “maximum”). It can tell you about your outliers and what their values are. It can also tell you if your data is symmetrical, how tightly your data is grouped, and if and how your data is skewed.

Statistical concepts of classification of Data

- Classification is the process of arranging data into homogeneous (similar) groups according to their common characteristics.
- Raw data cannot be easily understood, and it is not fit for further analysis and interpretation. Arrangement of data helps users in comparison and analysis. It is also important for statistical sampling.

Classification of Data

There are four types of classification. They are:

- ▶ **Geographical classification**

When data are classified on the basis of location or areas, it is called geographical classification

- ▶ **Chronological classification**

Chronological classification means classification on the basis of time, like months, years etc.

- ▶ **Qualitative classification**

In Qualitative classification, data are classified on the basis of some attributes or quality such as gender, colour of hair, literacy and religion. In this type of classification, the attribute under study cannot be measured. It can only be found out whether it is present or absent in the units of study.

- ▶ **Quantitative classification**

Quantitative classification refers to the classification of data according to some characteristics, which can be measured such as height, weight, income, profits etc.

Quantitative classification

- ▶ There are two types of quantitative classification of data: Discrete frequency distribution and Continuous frequency distribution.
- ▶ In this type of classification there are two elements

- ▶ **variable**

Variable refers to the characteristic that varies in magnitude or quantity. E.g. weight of the students. A variable may be discrete or continuous.

- ▶ **Frequency**

Frequency refers to the number of times each variable gets repeated. For example there are 50 students having weight of 60 kgs. Here 50 students is the frequency.

Frequency distribution

- Frequency distribution refers to data classified on the basis of some variable that can be measured such as prices, weight, height, wages etc.

Weights (in kg)	Number of students
31 - 35	9
36 - 40	5
41 - 45	14
46 - 50	3
51 - 55	1
56 - 60	2
61 - 65	2
66 - 70	1
71 - 75	1
Total	38

Frequency distribution

The following technical terms are important when a continuous frequency distribution is formed

Class limits: Class limits are the lowest and highest values that can be included in a class. For example take the class 51-55. The lowest value of the class is 51 and the highest value is 55. In this class there can be no value lesser than 51 or more than 55. 51 is the lower class limit and 55 is the upper class limit.

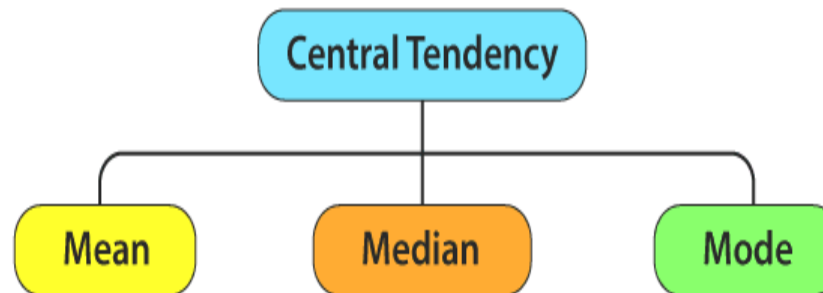
Class interval: The difference between the upper and lower limit of a class is known as class interval of that class.

Class frequency: The number of observations corresponding to a particular class is known as the frequency of that class

Weights (in kg)	Number of students
31 - 35	9
36 - 40	5
41 - 45	14
46 - 50	3
51 - 55	1
56 - 60	2
61 - 65	2
66 - 70	1
71 - 75	1
Total	38

Measures of Centre Tendency

- In statistics, the **central tendency** is the descriptive summary of a data set.
- Through the single value from the dataset, it reflects the centre of the data distribution.
- Moreover, it does not provide information regarding individual data from the dataset, which can be obtained from the dataset.



dataset.
dataset
asures

Mean

- The mean represents the average value of the dataset.
- It can be calculated as the sum of all the values in the dataset divided by the number of values. In general, it is considered as the **arithmetic mean**.
- Some other measures of mean used to find the central tendency are as follows:
 - **Geometric Mean** (nth root of the product of n numbers)
 - **Harmonic Mean** (the reciprocal of the average of the reciprocals)
 - **Weighted Mean** (where some values contribute more than others)
- It is observed that if all the values in the dataset are the same, then all geometric, arithmetic and harmonic mean values are the same. If there is variability in the data, then the mean value differs.

Arithmetic Mean

Arithmetic mean represents a number that is obtained by dividing the sum of the elements of a set by the number of values in the set. So you can use the layman term Average. If any data set consisting of the values $b_1, b_2, b_3, \dots, b_n$ then the arithmetic mean B is defined as:

$$B = (\text{Sum of all observations}) / (\text{Total number of observation})$$

Innings	1	2	3	4	5	6	7	8	9	10
Runs	50	59	90	8	106	117	59	91	7	74

The arithmetic mean of Virat Kohli's batting scores also called his Batting Average is;

$$\text{Sum of runs scored} / \text{Number of innings} = 661/10$$

The arithmetic mean of his scores in the last 10 innings is 66.1.

Harmonic Mean

A Harmonic Progression is a sequence if the reciprocals of its terms are in Arithmetic Progression, and harmonic mean (or shortly written as HM) can be calculated by dividing the number of terms by reciprocals of its terms.

For example, Terms t_1, t_2, t_3 is HP if and only if

$\frac{1}{t_1}, \frac{1}{t_2}, \frac{1}{t_3}, \dots$ is an AP.

Then, Harmonic Mean = $\frac{3}{\frac{1}{t_1} + \frac{1}{t_2} + \frac{1}{t_3}}$

In particular cases, especially those involving rates and ratios, the harmonic mean gives the most correct value of the mean. For example, if a vehicle travels a specified distance at speed x (eg 60 km / h) and then travels again at the speed y (e.g. 40 km / h), the average speed value is the harmonic mean x, y (ie, 48 km / h).

Geometric Mean

- The **Geometric Mean (GM)** is the average value or mean which signifies the central tendency of the set of numbers by finding the product of their values.
- Basically, we multiply the numbers altogether and take out the nth root of the multiplied numbers, where n is the total number of values.
- For example: for a given set of two numbers such as 3 and 1, the geometric mean is equal to $\sqrt[2]{3 \times 1} = \sqrt{3} \approx 1.732$.

Consider, if $x_1, x_2, \dots, x_n$ are the observation, then the G.M is defined as:

$$G.M = \sqrt[n]{x_1 \times x_2 \times \dots \times x_n}$$

or

$$G.M = (x_1 \times x_2 \times \dots \times x_n)^{\frac{1}{n}}$$

Use of Geometric Mean

- ▶ For example, suppose you have an investment which earns 10% the first year, 50% the second year, and 30% the third year. What is its average rate of return?
- ▶ It is not the arithmetic mean, because what these numbers mean is that on the first year your investment was multiplied (not added to) by 1.10, on the second year it was multiplied by 1.60, and the third year it was multiplied by 1.20. The relevant quantity is the geometric mean of these three numbers.
- ▶ The question about finding the average rate of return can be rephrased as: "by what constant factor would your investment need to be multiplied by each year in order to achieve the same effect as multiplying by 1.10 one year, 1.60 the next, and 1.20 the third?"
- ▶ If you calculate this geometric mean
- ▶ You get approximately 1.283, so the average rate of return is about 28% (not 30% which is what the arithmetic mean of 10%, 50%, and 20% would give you).

$$(1.10 \times 1.60 \times 1.20)^{1/3}$$

Median

- ▶ Median is the middle value of the dataset in which the dataset is arranged in the ascending order or in descending order.
- ▶ When the dataset contains an even number of values, then the median value of the dataset can be found by taking the mean of the middle two values.
- ▶ If you have skewed distribution, the best measure of finding the central tendency is the median.
- ▶ The median is less sensitive to outliers (extreme scores) than the mean and thus a better measure than the mean for highly skewed distributions, e.g. family income. For example mean of 20, 30, 40, and 990 is $(20+30+40+990)/4 = 270$. The median of these four observations is $(30+40)/2 = 35$. Here 3 observations out of 4 lie between 20-40. So, the mean 270 really fails to give a realistic picture of the major part of the data. It is influenced by extreme value 990.

Median odd
23
21
18
16
15
13
12
10
9
7
6
5
2

Median even
40
38
35
33
32
30
29
27
26
24
23
22
19
17

28

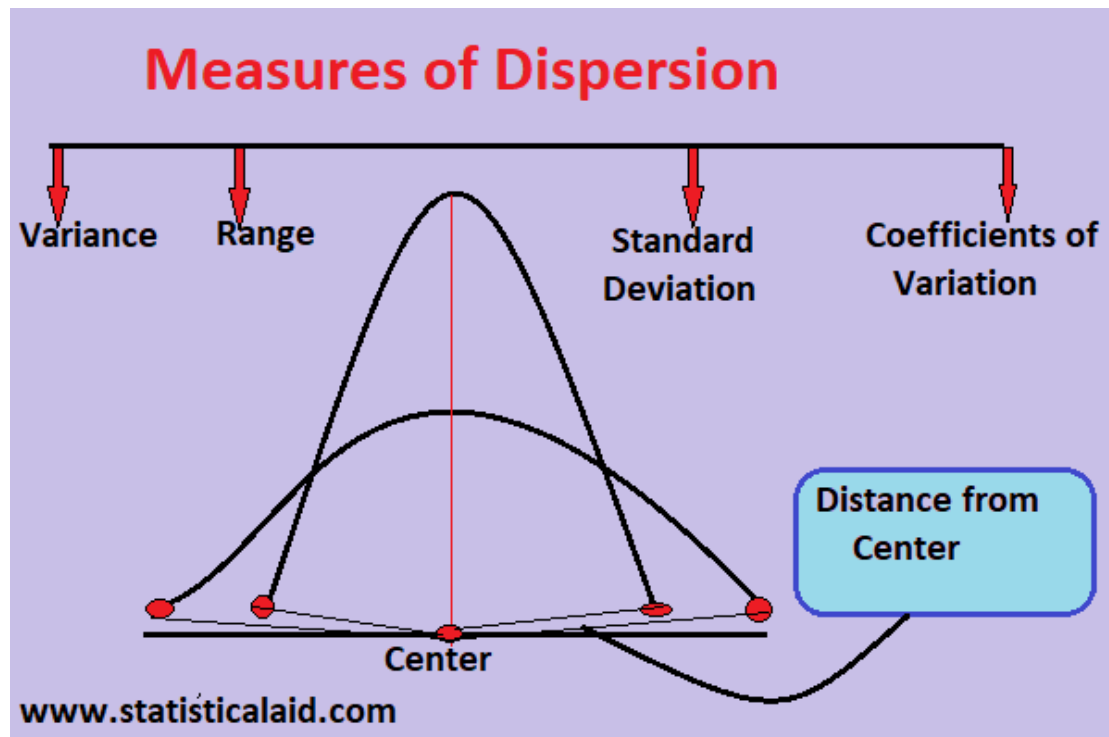
Mode

- The mode represents the frequently occurring value in the dataset.
- Sometimes the dataset may contain multiple modes and in some cases, it does not contain any mode at all.
- If you have categorical data, the mode is the best choice to find the central tendency.

Mode
5
5
5
4
4
3
2
2
1

Measures of Dispersion

Dispersion is the state of getting dispersed or spread. Statistical dispersion means the extent to which a numerical data is likely to vary about an average value. In other words, dispersion helps to understand the distribution of the data.



Objectives of computing dispersion

Comparative study

- ▶ Measures of dispersion give a single value indicating the degree of consistency or uniformity of distribution. This single value helps us in making comparisons of various distributions.

Reliability of an average

- ▶ A small value of dispersion means low variation between observations and average. It means that the average is a good representative of observation and very reliable. A higher value of dispersion means greater deviation among the observations.

Control the variability

- ▶ Different measures of dispersion provide us data of variability from different angles, and this knowledge can prove helpful in controlling the variation.

Basis for further statistical analysis

- ▶ Measures of dispersion provide the basis for further statistical analysis like computing correlation, regression, test of hypothesis, sampling etc.

Definition

- *Measures of dispersion* are descriptive statistics that describe how similar a set of scores are to each other
 - The more similar the scores are to each other, the lower the measure of dispersion will be
 - The less similar the scores are to each other, the higher the measure of dispersion will be
 - In general, the more spread out a distribution is, the larger the measure of dispersion will be

Types of Measures of Dispersion

There are two main types of dispersion methods in statistics which are:

- ▶ Absolute Measure of Dispersion
- ▶ Relative Measure of Dispersion

Absolute Measure of Dispersion

An absolute measure of dispersion contains the same unit as the original data set. Absolute dispersion method expresses the variations in terms of the average of deviations of observations like standard or means deviations. It includes range, standard deviation, quartile deviation, etc. The types of absolute measures of dispersion are:

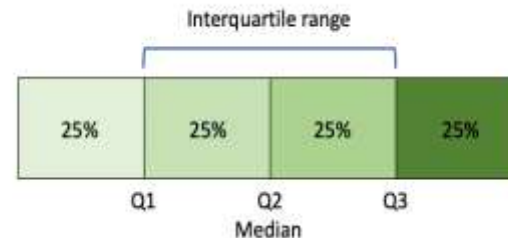
- ▶ **Range:** It is simply the difference between the maximum value and the minimum value given in a data set. Example: 1, 3, 5, 6, 7 $\Rightarrow$ Range = $7 - 1 = 6$
- ▶ **Variance:** Deduct the mean from each data in the set then squaring each of them and adding each square and finally dividing them by the total no of values in the data set is the variance. Variance $(\sigma^2) = \sum(X - \mu)^2 / N$
- ▶ **Standard Deviation:** The square root of the variance is known as the standard deviation i.e. S.D. = $\sqrt{\sigma}$.
- ▶ **Quartiles and Quartile Deviation:** The quartiles are values that divide a list of numbers into quarters. The quartile deviation is half of the distance between the third and the first quartile.
- ▶ **Mean and Mean Deviation:** The average of numbers is known as the mean and the arithmetic mean of the absolute deviations of the observations from a measure of central tendency is known as the mean deviation (also called mean absolute deviation).

Range

- ▶ It is the simplest method of measurement of dispersion.
- ▶ It is defined as the difference between the largest and the smallest item in a given distribution.
- ▶ **Range = Largest item (L) – Smallest item (S)**

Interquartile Range

- ▶ It is defined as the difference between the Upper Quartile and Lower Quartile of a given distribution.
- ▶ **Interquartile Range = Upper Quartile (Q_3) – Lower Quartile (Q_1)**



The Range

- The *range* is defined as the difference between the largest score in the set of data and the smallest score in the set of data, $X_L - X_S$
- What is the range of the following data:
4 8 1 6 6 2 9 3 6 9
- The largest score (X_L) is 9; the smallest score (X_S) is 1; the range is $X_L - X_S = 9 - 1 = 8$

When To Use the Range

- The range is used when
 - you have ordinal data or
 - you are presenting your results to people with little or no knowledge of statistics
- The range is rarely used in scientific work as it is fairly insensitive
 - It depends on only two scores in the set of data, X_L and X_S
 - Two very different sets of data can have the same range:
1 1 1 1 9 vs 1 3 5 7 9

The Semi-Interquartile Range

- The *semi-interquartile range* (or *SIR*) is defined as the difference of the first and third quartiles divided by two
 - The first quartile is the 25th percentile
 - The third quartile is the 75th percentile
- $SIR = (Q_3 - Q_1) / 2$

SIR Example

- What is the SIR for the data to the right?
- 25 % of the scores are below 5
 - 5 is the first quartile
- 25 % of the scores are above 25
 - 25 is the third quartile
- $SIR = (Q_3 - Q_1) / 2 = (25 - 5) / 2 = 10$

2
4
6
8
10
12
14
20
30
60

← 5 = 25th %tile

← 25 = 75th %tile

When To Use the SIR

- The SIR is often used with skewed data as it is insensitive to the extreme scores

Variance

- ▶ Variance is a measure of how data points differ from the mean.
- ▶ A variance is a measure of how far a set of data (numbers) are spread out from their mean (average) value.
- ▶ The more the value of variance, the data is more scattered from its mean and if the value of variance is low or minimum, then it is less scattered from mean. Therefore, it is called a measure of spread of data from mean.
- ▶ the formula for variance is
$$\text{Var}(X) = E[(X - \mu)^2]$$
- ▶ the variance is the square of standard deviation, i.e.,
Variance = (Standard deviation)² = σ^2

Variance

Example: Find the variance of the numbers 3, 8, 6, 10, 12, 9, 11, 10, 12, 7.

Given,

3, 8, 6, 10, 12, 9, 11, 10, 12, 7

Step 1: Compute the mean of the 10 values given.

$$\begin{aligned}\text{Mean } (\mu) &= (3+8+6+10+12+9+11+10+12+7) / 10 \\ &= 88 / 10 = 8.8\end{aligned}$$

Variance

Value X	$X - \mu$	$(X - \mu)^2$
3	-5.8	33.64
8	-0.8	0.64
6	-2.8	7.84
10	1.2	1.44
12	3.2	10.24
9	0.2	0.04
11	2.2	4.84
10	1.2	1.44
12	3.2	10.24
7	-1.8	3.24
Total	0	73.6

$$\begin{aligned}\sigma^2 &= \frac{\sum (X - \mu)^2}{N} \\ &= 73.6 / 10 \\ &= 7.36\end{aligned}$$

Coefficient of variance

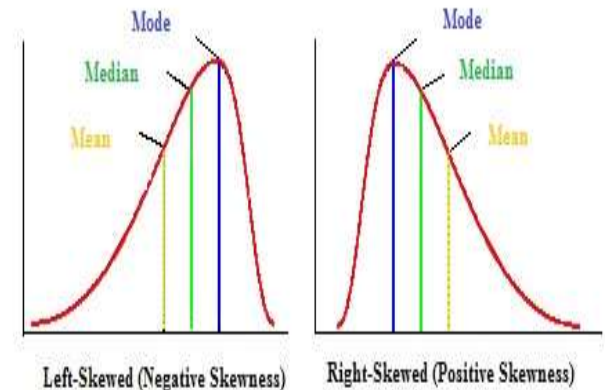
- The coefficient of variance (CV) is a relative measure of variability that indicates the size of a standard deviation in relation to its mean.
- It is a standardized, unitless measure that allows you to compare variability between disparate groups and characteristics.
- It is calculated as $CV = \frac{\text{Standard deviation}}{\text{Mean}}$
- The coefficient of variance facilitates meaningful comparisons in scenarios where absolute measures cannot.

Quartile Deviation

- The Quartile Deviation (QD) is the product of half of the difference between the upper and lower quartiles.
- Mathematically we can define as: Quartile Deviation = $(Q3 - Q1) / 2$
- Quartile Deviation defines the absolute measure of dispersion. Whereas the relative measure corresponding to QD, is known as the coefficient of QD, which is obtained by applying the certain set of the formula: Coefficient of Quartile Deviation = $(Q3 - Q1) / (Q3 + Q1)$
- A Coefficient of QD is used to study & compare the degree of variation in different situations.

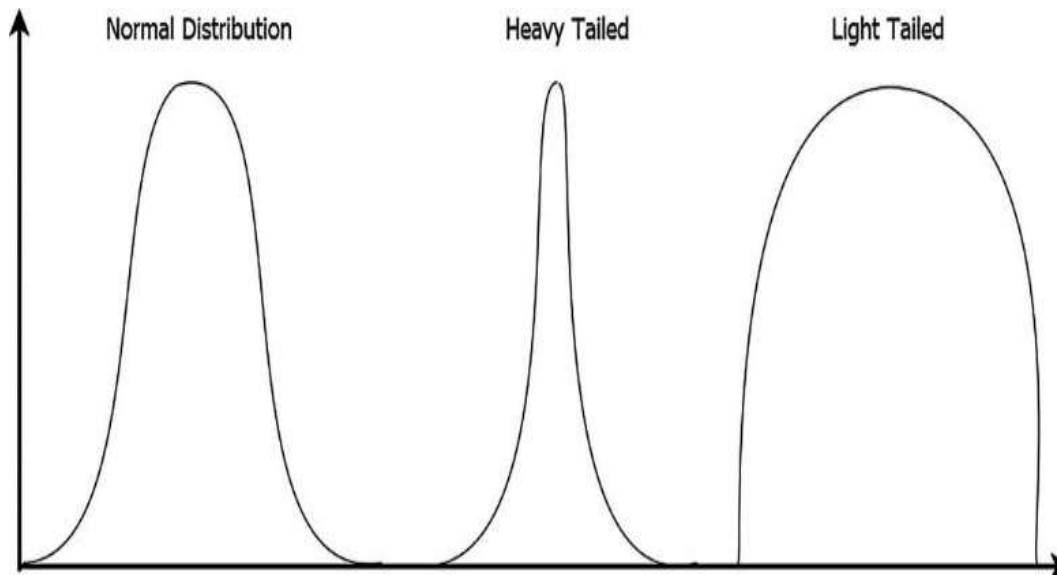
Skewness

- Skewness is a measure of the degree of asymmetry of a distribution.
- If the left tail (tail at small end of the distribution) is more pronounced than the right tail (tail at the large end of the distribution), the function is said to have negative skewness.
- If the reverse is true, it has positive skewness. If the two are equal, it has zero skewness.

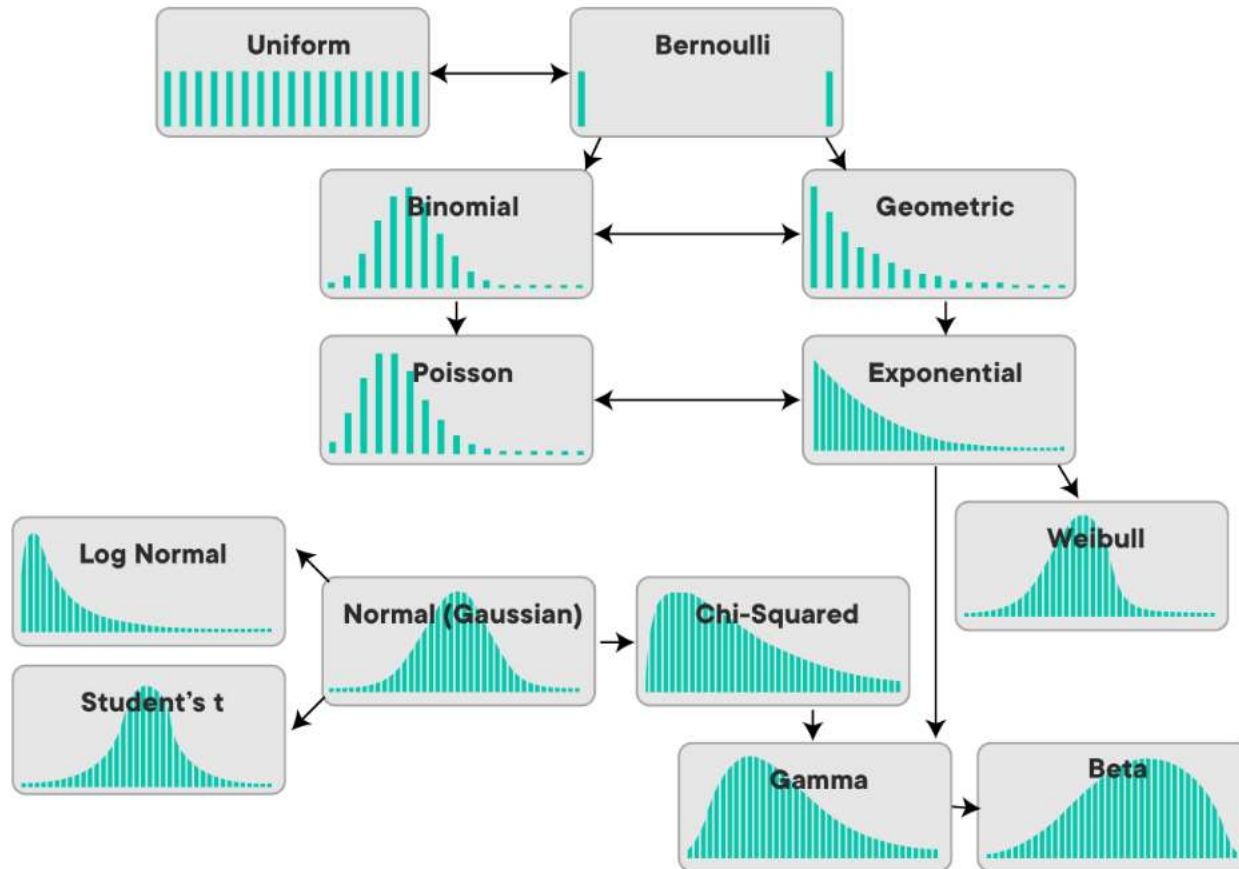


Kurtosis

- Kurtosis is a measure of whether the data are heavy-tailed or light-tailed relative to a normal distribution.
- That is, data sets with high kurtosis tend to have heavy tails, or outliers. Data sets with low kurtosis tend to have light tails, or lack of outliers.
- Significant skewness and kurtosis clearly indicate that data are not normal.



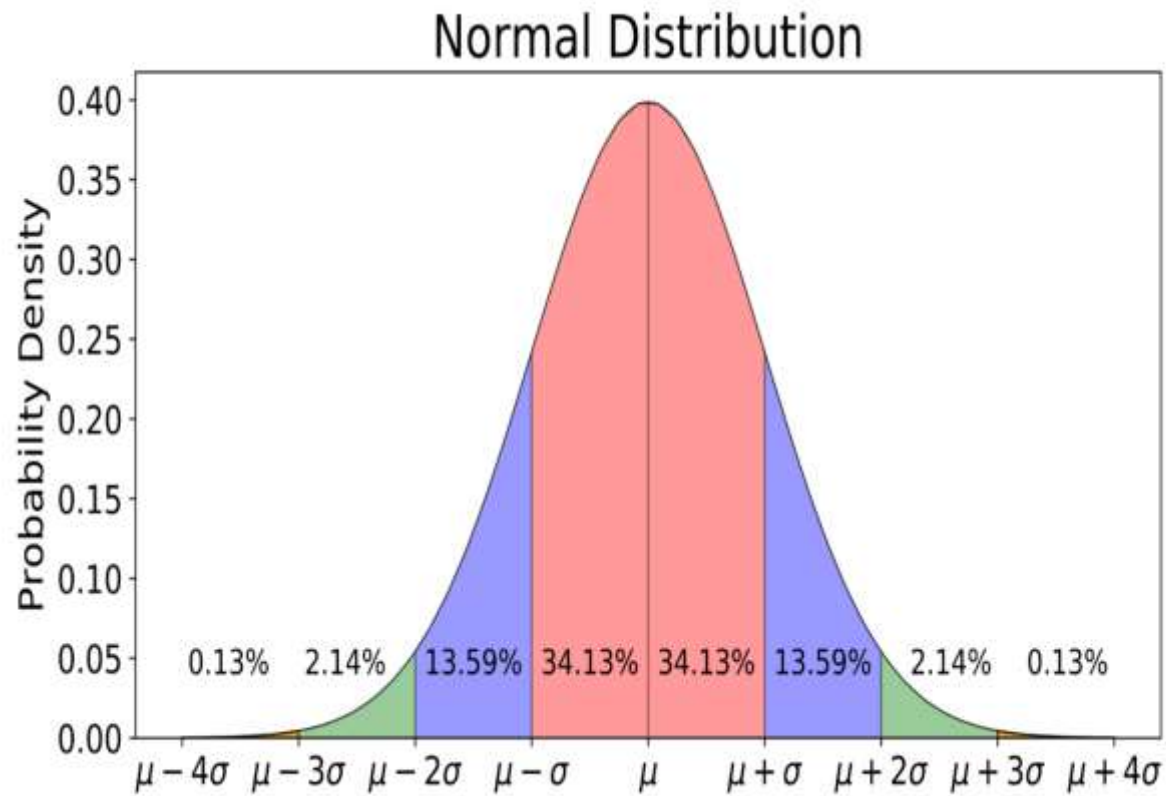
Types of Distributions



Normal Distribution

- In probability theory and statistics, the Normal Distribution, also called the Gaussian Distribution, is the most significant continuous probability distribution.
- A large number of random variables are either nearly or exactly represented by the normal distribution, in every physical science and economics.
- In a normal distribution, the mean, mean and mode are equal.(i.e., $\text{Mean} = \text{Median} = \text{Mode}$). The normally distributed curve should be symmetric at the centre.

Normal Distribution



Association Rule Mining

- Given a set of transactions, find rules that will predict the occurrence of an item based on the occurrences of other items in the transaction

Market-Basket transactions

<i>TID</i>	<i>Items</i>
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Example of Association Rules

$\{\text{Diaper}\} \rightarrow \{\text{Beer}\},$
 $\{\text{Milk, Bread}\} \rightarrow \{\text{Eggs, Coke}\},$
 $\{\text{Beer, Bread}\} \rightarrow \{\text{Milk}\},$

Implication means co-occurrence, not causality!

Definition: Frequent Itemset

- **Itemset**
 - A collection of one or more items
 - Example: {Milk, Bread, Diaper}
 - k-itemset
 - An itemset that contains k items
- **Support count (σ)**
 - Frequency of occurrence of an itemset
 - E.g. $\sigma(\{\text{Milk, Bread, Diaper}\}) = 2$
- **Support**
 - Fraction of transactions that contain an itemset
 - E.g. $s(\{\text{Milk, Bread, Diaper}\}) = 2/5$
- **Frequent Itemset**
 - An itemset whose support is greater than or equal to a *minsup* threshold

<i>TID</i>	<i>Items</i>
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Support and Confidence

- **Support count**: The support count of an itemset X , denoted by $X.count$, in a data set T is the number of transactions in T that contain X . Assume T has n transactions.

- Then,
$$support = \frac{(X \cup Y).count}{n}$$

$$confidence = \frac{(X \cup Y).count}{X.count}$$

Goal and key features

- **Goal:** Find all rules that satisfy the user-specified *minimum support* (minsup) and *minimum confidence* (minconf).
- **Key Features**
 - **Completeness:** find all rules.
 - **No target item(s)** on the right-hand-side
 - Mining with data on **hard disk** (not in memory)

Definition: Association Rule

- Association Rule
 - An implication expression of the form $X \rightarrow Y$, where X and Y are itemsets
 - Example:
 $\{\text{Milk, Diaper}\} \rightarrow \{\text{Beer}\}$

<i>TID</i>	<i>Items</i>
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

- Rule Evaluation Metrics
 - Support (s)
 - ◆ Fraction of transactions that contain both X and Y
 - Confidence (c)
 - ◆ Measures how often items in Y appear in transactions that contain X

Example:

$\{\text{Milk, Diaper}\} \Rightarrow \text{Beer}$

$$s = \frac{\sigma(\text{Milk, Diaper, Beer})}{|T|} = \frac{2}{5} = 0.4$$

$$c = \frac{\sigma(\text{Milk, Diaper, Beer})}{\sigma(\text{Milk, Diaper})} = \frac{2}{3} = 0.67$$

Association Rule Mining Task

- Given a set of transactions T , the goal of association rule mining is to find all rules having
 - support $\geq \textit{minsup}$ threshold
 - confidence $\geq \textit{minconf}$ threshold
- Brute-force approach:
 - List all possible association rules
 - Compute the support and confidence for each rule
 - Prune rules that fail the *minsup* and *minconf* thresholds

⇒ **Computationally prohibitive!**

Mining Association Rules

Example of Rules:

<i>TID</i>	<i>Items</i>
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

$\{\text{Milk, Diaper}\} \rightarrow \{\text{Beer}\}$ ($s=0.4, c=0.67$)
 $\{\text{Milk, Beer}\} \rightarrow \{\text{Diaper}\}$ ($s=0.4, c=1.0$)
 $\{\text{Diaper, Beer}\} \rightarrow \{\text{Milk}\}$ ($s=0.4, c=0.67$)
 $\{\text{Beer}\} \rightarrow \{\text{Milk, Diaper}\}$ ($s=0.4, c=0.67$)
 $\{\text{Diaper}\} \rightarrow \{\text{Milk, Beer}\}$ ($s=0.4, c=0.5$)
 $\{\text{Milk}\} \rightarrow \{\text{Diaper, Beer}\}$ ($s=0.4, c=0.5$)

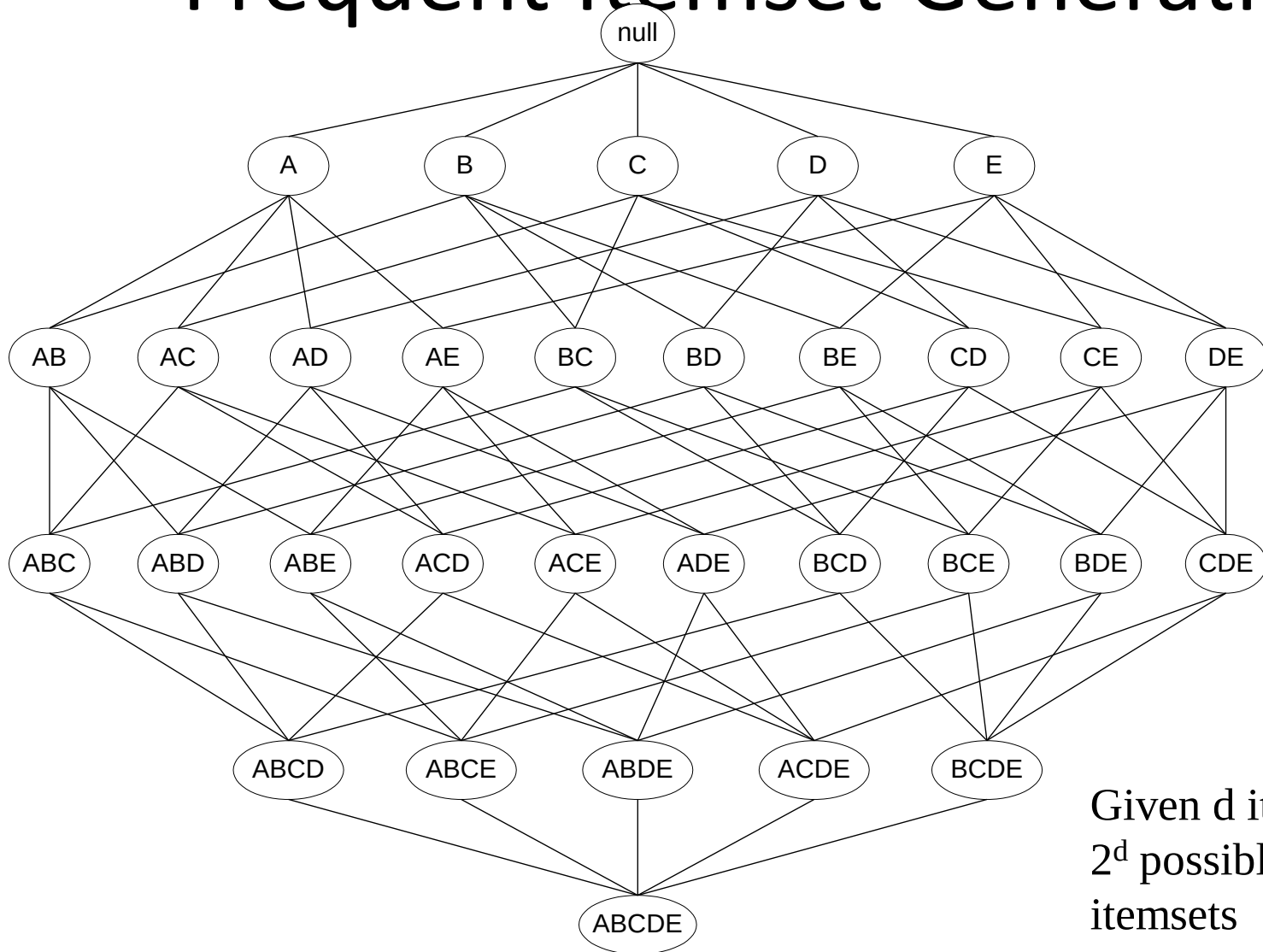
Observations:

- All the above rules are binary partitions of the same itemset:
 $\{\text{Milk, Diaper, Beer}\}$
- Rules originating from the same itemset have identical support but can have different confidence
- Thus, we may decouple the support and confidence requirements

Mining Association Rules

- Two-step approach:
 1. Frequent Itemset Generation
 - Generate all itemsets whose support $\geq$ minsup
 2. Rule Generation
 - Generate high confidence rules from each frequent itemset, where each rule is a binary partitioning of a frequent itemset
- Frequent itemset generation is still computationally expensive

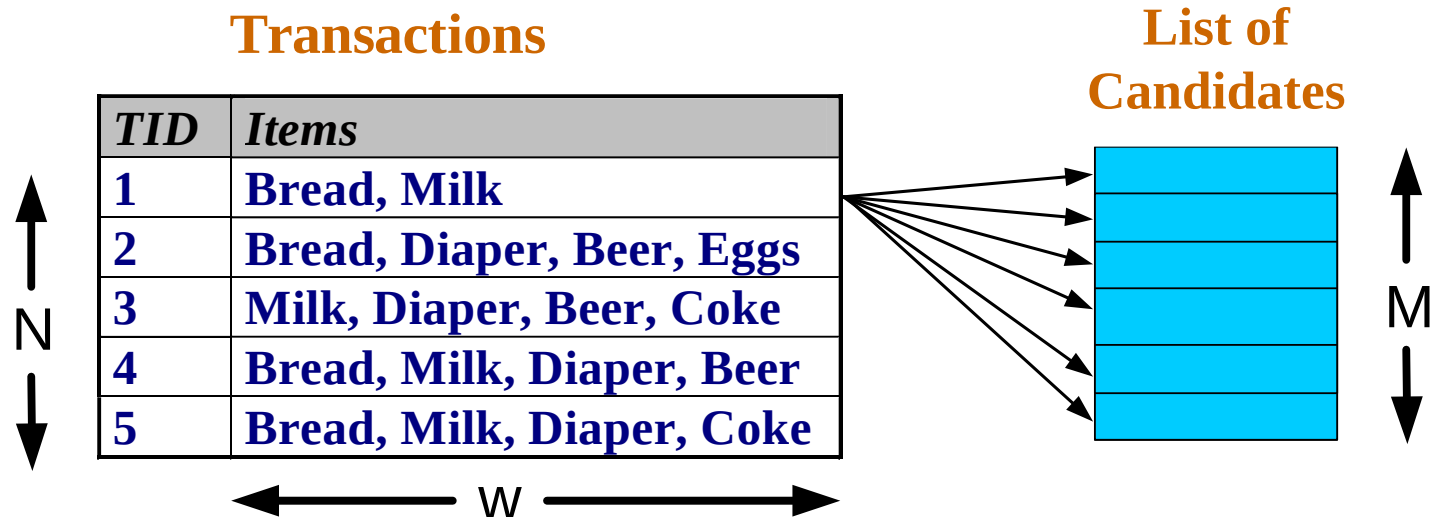
Frequent Itemset Generation



Given d items, there are 2^d possible candidate itemsets

Frequent Itemset Generation

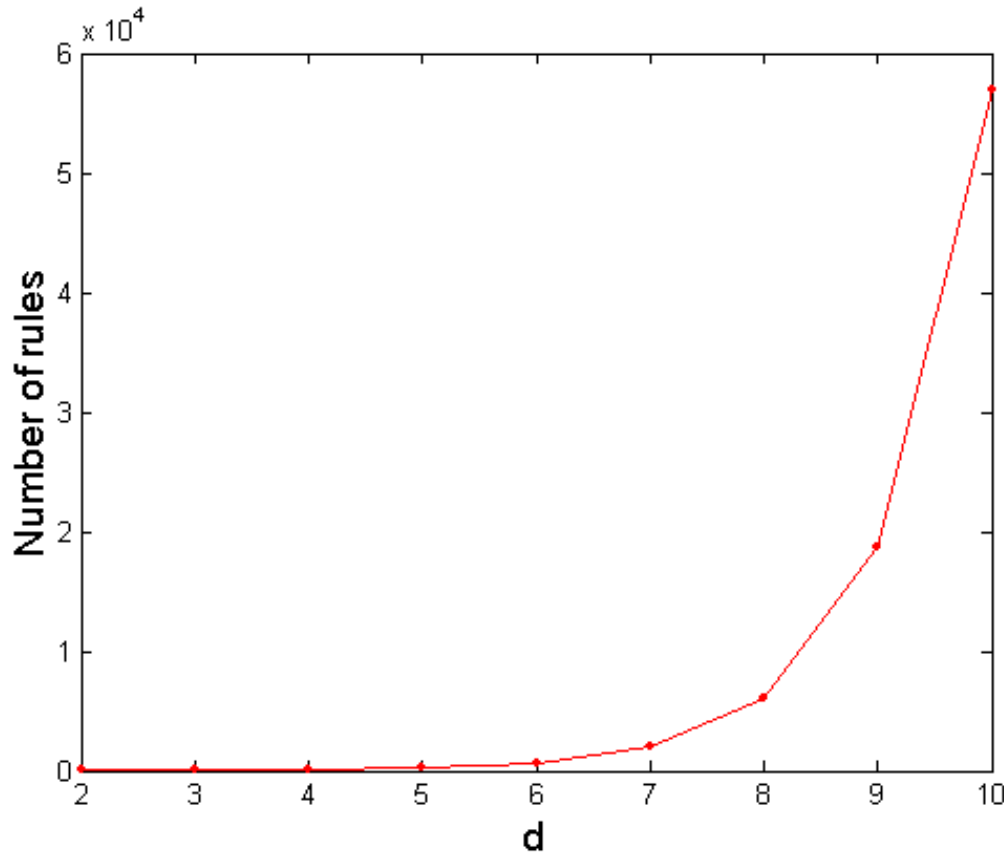
- Brute-force approach:
 - Each itemset in the lattice is a **candidate** frequent itemset
 - Count the support of each candidate by scanning the database



- Match each transaction against every candidate
- Complexity $\sim O(NMw) \Rightarrow$ **Expensive since $M = 2^d$!!!**

Computational Complexity

- Given d unique items:
 - Total number of itemsets = 2^d
 - Total number of possible association rules:



$$R = \sum_{k=1}^{d-1} \left[\binom{d}{k} \times \sum_{j=1}^{d-k} \binom{d-k}{j} \right]$$
$$= 3^d - 2^{d+1} + 1$$

If $d=6$, $R = 602$ rules

Frequent Itemset Generation Strategies

- Reduce the **number of candidates** (M)
 - Complete search: $M=2^d$
 - Use pruning techniques to reduce M
- Reduce the **number of transactions** (N)
 - Reduce size of N as the size of itemset increases
 - Used by DHP and vertical-based mining algorithms
- Reduce the **number of comparisons** (NM)
 - Use efficient data structures to store the candidates or transactions
 - No need to match every candidate against every transaction

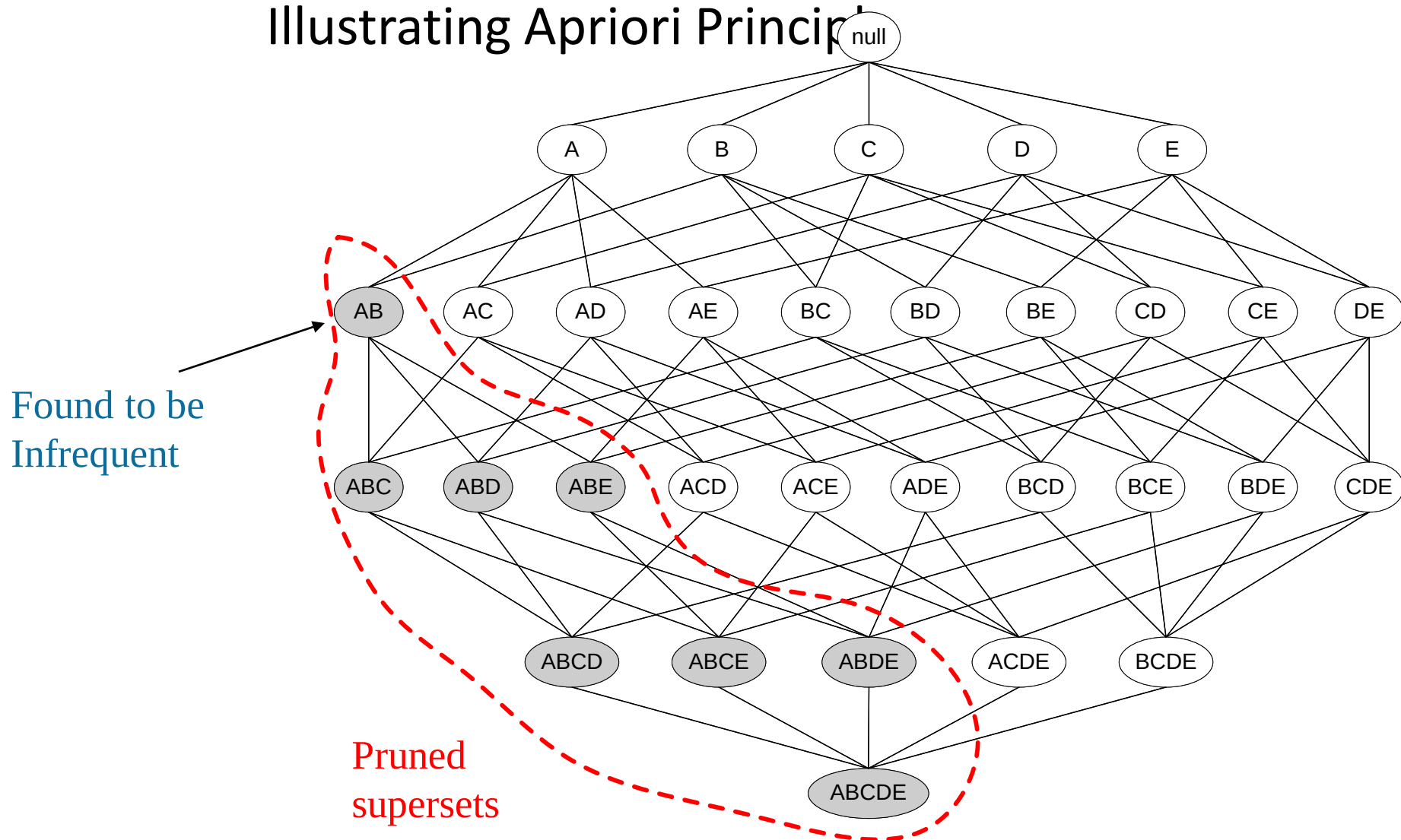
Reducing Number of Candidates

- **Apriori principle:**
 - If an itemset is frequent, then all of its subsets must also be frequent
- Apriori principle holds due to the following property of the support measure:

$$\forall X, Y : (X \subseteq Y) \Rightarrow s(X) \geq s(Y)$$

- Support of an itemset never exceeds the support of its subsets
- This is known as the **anti-monotone** property of support

Illustrating Apriori Principle



Illustrating Apriori Principle

Item	Count
Bread	4
Coke	2
Milk	4
Beer	3
Diaper	4
Eggs	1

Items (1-itemsets)



Itemset	Count
{Bread,Milk}	3
{Bread,Beer}	2
{Bread,Diaper}	3
{Milk,Beer}	2
{Milk,Diaper}	3
{Beer,Diaper}	3

Pairs (2-itemsets)

(No need to generate candidates involving Coke or Eggs)

Minimum Support = 3

If every subset is considered,
 ${}^6C_1 + {}^6C_2 + {}^6C_3 = 41$
 With support-based pruning,
 $6 + 6 + 1 = 13$



Triplets (3-itemsets)

Itemset	Count
{Bread,Milk,Diaper}	3



Apriori Algorithm

- Method:
 - Let $k=1$
 - Generate frequent itemsets of length 1
 - Repeat until no new frequent itemsets are identified
 - Generate length $(k+1)$ candidate itemsets from length k frequent itemsets
 - Prune candidate itemsets containing subsets of length k that are infrequent
 - Count the support of each candidate by scanning the DB
 - Eliminate candidates that are infrequent, leaving only those that are frequent

Candidate-gen function

Function candidate-gen(F_{k-1})

$C_k \leftarrow \emptyset;$

forall $f_1, f_2 \in F_{k-1}$

 with $f_1 = \{i_1, \dots, i_{k-2}, i_{k-1}\}$

 and $f_2 = \{i_1, \dots, i_{k-2}, i'_{k-1}\}$

 and $i_{k-1} < i'_{k-1}$ **do**

$c \leftarrow \{i_1, \dots, i_{k-1}, i'_{k-1}\};$ // join f_1 and f_2

$C_k \leftarrow C_k \cup \{c\};$

for each $(k-1)$ -subset s of c **do**

if ($s \notin F_{k-1}$) **then**

 delete c from $C_k;$ // prune

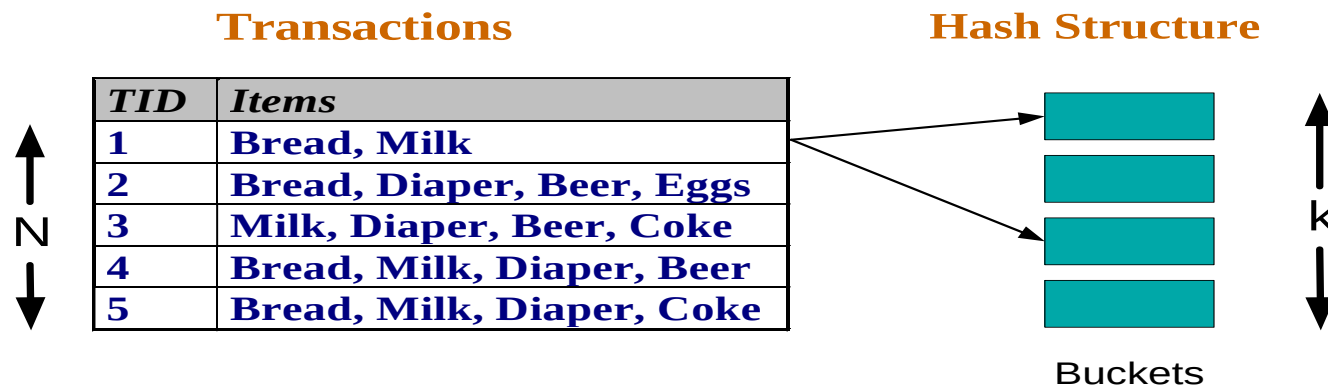
end

end

return $C_k;$

Reducing Number of Comparisons

- Candidate counting:
 - Scan the database of transactions to determine the support of each candidate itemset
 - To reduce the number of comparisons, store the candidates in a hash structure
 - Instead of matching each transaction against every candidate, match it against candidates contained in the hashed buckets



Improving Apriori's Efficiency

- **Hash-based itemset counting:** A k -itemset whose corresponding hashing bucket count is below the threshold cannot be frequent
- **Transaction reduction:** A transaction that does not contain any frequent k -itemset is useless in subsequent scans
- **Partitioning:** Any itemset that is potentially frequent in DB must be frequent in at least one of the partitions of DB
- **Sampling:** mining on a subset of given data, need a lower support threshold + a method to determine the completeness
- **Dynamic itemset counting:** add new candidate itemsets immediately (unlike Apriori) when all of their subsets are estimated to be frequent

Is Apriori Fast Enough? — Performance Bottlenecks

- The core of the Apriori algorithm:
 - Use frequent $(k - 1)$ -itemsets to generate candidate frequent k -itemsets
 - Use database scan and pattern matching to collect counts for the candidate itemsets
- The bottleneck of *Apriori*: candidate generation
 - Huge candidate sets:
 - 10^4 frequent 1-itemset will generate 10^7 candidate 2-itemsets
 - To discover a frequent pattern of size 100, e.g., $\{a_1, a_2, \dots, a_{100}\}$, one needs to generate $2^{100} \approx 10^{30}$ candidates.
 - Multiple scans of database:
 - Needs $(n + 1)$ scans, n is the length of the longest pattern

Construct FP-tree from a Transaction DB

<i>TID</i>	<i>Items bought</i>	<i>(ordered) frequent items</i>
100	{ <i>f, a, c, d, g, i, m, p</i> }	{ <i>f, c, a, m, p</i> }
200	{ <i>a, b, c, f, l, m, o</i> }	{ <i>f, c, a, b, m</i> }
300	{ <i>b, f, h, j, o</i> }	{ <i>f, b</i> }
400	{ <i>b, c, k, s, p</i> }	{ <i>c, b, p</i> }
500	{ <i>a, f, c, e, l, p, m, n</i> }	{ <i>f, c, a, m, p</i> }

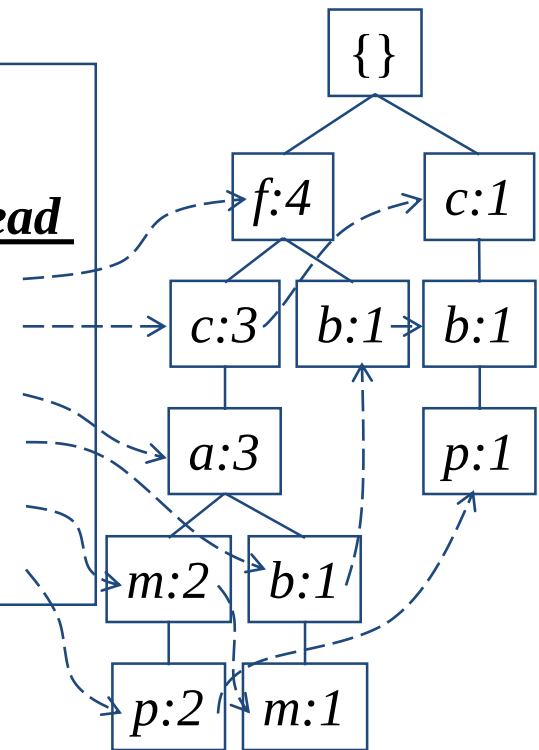
min_support = 0.5

Steps:

1. Scan DB once, find frequent 1-itemset (single item pattern)
2. Order frequent items in frequency descending order
3. Scan DB again, construct FP-tree

Header Table

<i>Item</i>	<i>frequency</i>	<i>head</i>
<i>f</i>	4	
<i>c</i>	4	
<i>a</i>	3	
<i>b</i>	3	
<i>m</i>	3	
<i>p</i>	3	



Benefits of the FP-tree Structure

- Completeness:
 - never breaks a long pattern of any transaction
 - preserves complete information for frequent pattern mining
- Compactness
 - reduce irrelevant information—infrequent items are gone
 - frequency descending ordering: more frequent items are more likely to be shared
 - never be larger than the original database (if not count node-links and counts)

Mining Frequent Patterns Using FP-tree

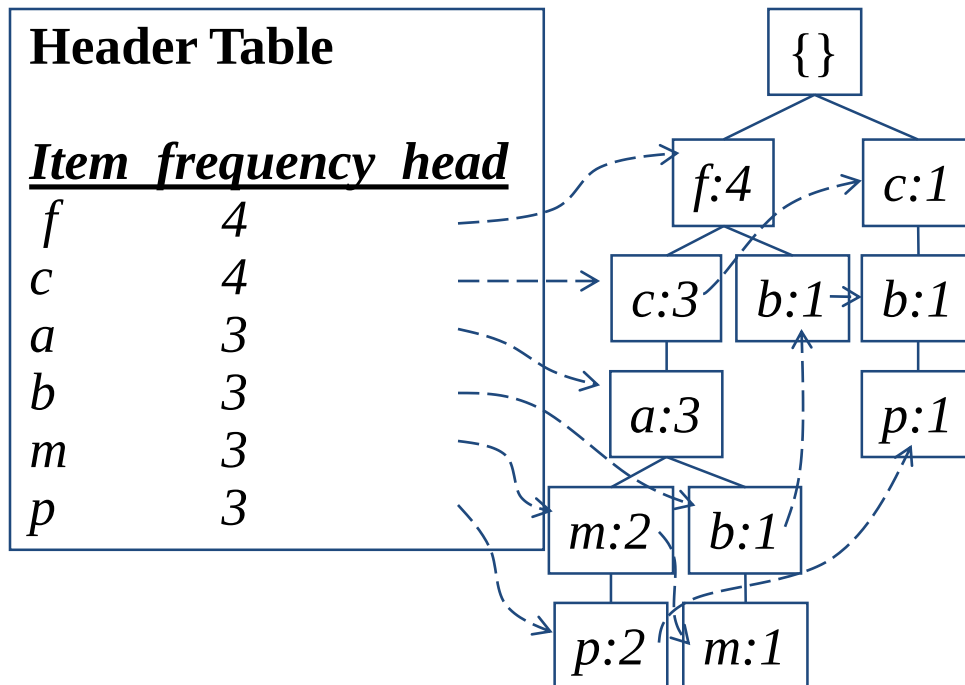
- General idea (divide-and-conquer)
 - Recursively grow frequent pattern path using the FP-tree
- Method
 - For each item, construct its **conditional pattern-base**, and then its **conditional FP-tree**
 - Repeat the process on each newly created conditional FP-tree
 - Until the resulting FP-tree is **empty**, or it contains **only one path** (single path will generate all the combinations of its sub-paths, each of which is a frequent pattern)

Major Steps to Mine FP-tree

- 1) Construct conditional pattern base for each node in the FP-tree
- 2) Construct conditional FP-tree from each conditional pattern-base
- 3) Recursively mine conditional FP-trees and grow frequent patterns obtained so far
 - If the conditional FP-tree contains a single path, simply enumerate all the patterns

Step 1: From FP-tree to Conditional Pattern Base

- Starting at the frequent header table in the FP-tree
- Traverse the FP-tree by following the link of each frequent item
- Accumulate all of transformed prefix paths of that item to form a conditional pattern base



Conditional pattern bases

item cond. pattern base

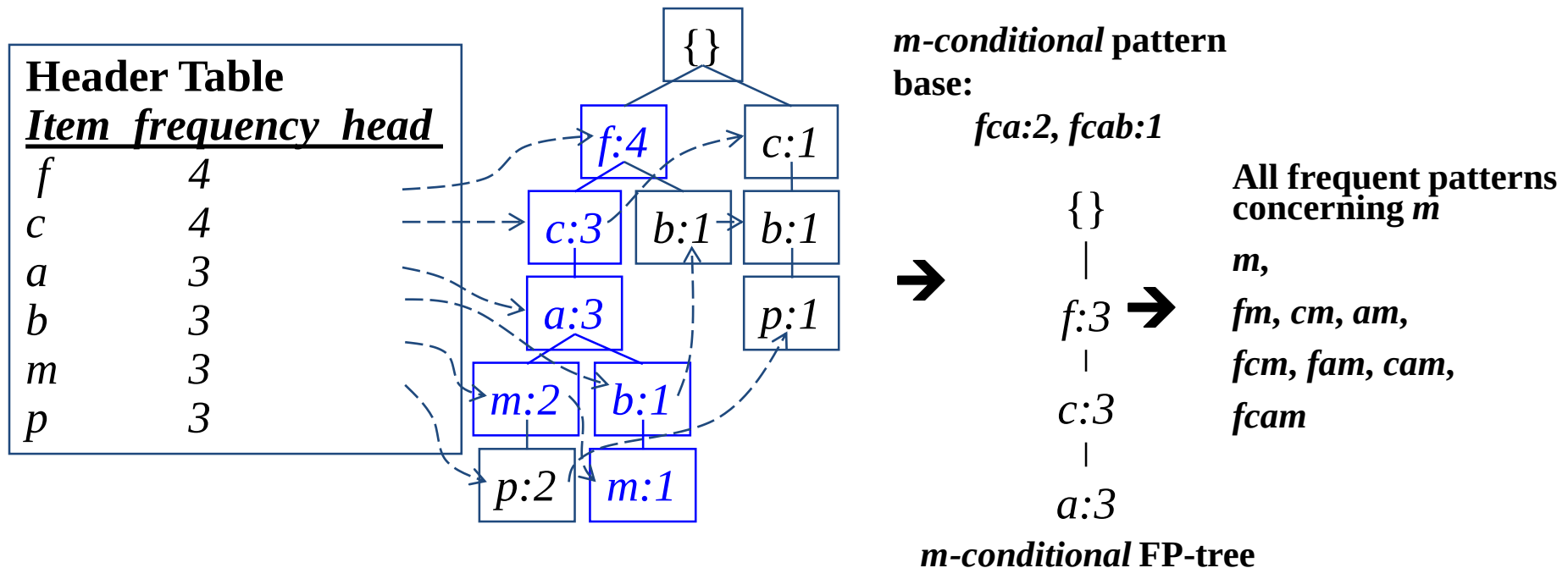
c *f:3*
a *fc:3*
b *fca:1, f:1, c:1*
m *fca:2, fcab:1*
p *fcam:2, cb:1*

Properties of FP-tree for Conditional Pattern Base Construction

- Node-link property
 - For any frequent item a_i , all the possible frequent patterns that contain a_i can be obtained by following a_i 's node-links, starting from a_i 's head in the FP-tree header
- Prefix path property
 - To calculate the frequent patterns for a node a_i in a path P , only the prefix sub-path of a_i in P need to be accumulated, and its frequency count should carry the same count as node a_i .

Step 2: Construct Conditional FP-tree

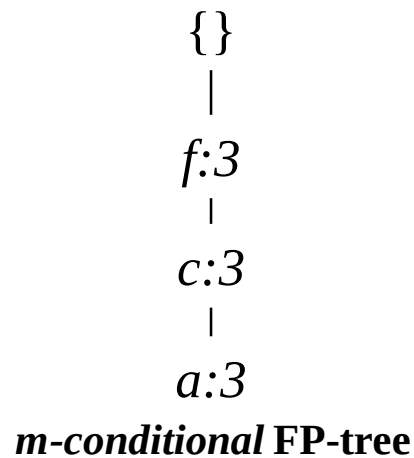
- For each pattern-base
 - Accumulate the count for each item in the base
 - Construct the FP-tree for the frequent items of the pattern base



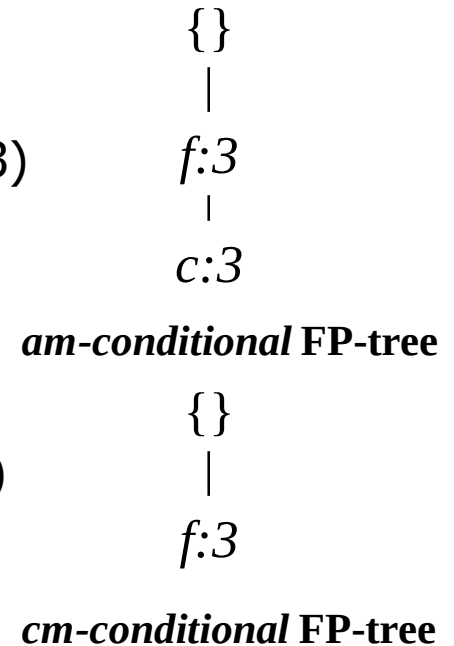
Mining Frequent Patterns by Creating Conditional Pattern-Bases

Item	Conditional pattern-base	Conditional FP-tree
p	$\{(fca:m:2), (cb:1)\}$	$\{(c:3)\} p$
m	$\{(fca:2), (fcab:1)\}$	$\{(f:3, c:3, a:3)\} m$
b	$\{(fca:1), (f:1), (c:1)\}$	Empty
a	$\{(fc:3)\}$	$\{(f:3, c:3)\} a$
c	$\{(f:3)\}$	$\{(f:3)\} c$
f	Empty	Empty

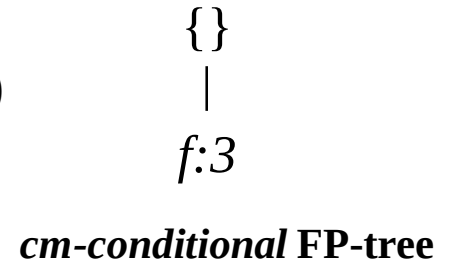
Step 3: Recursively mine the conditional FP-tree



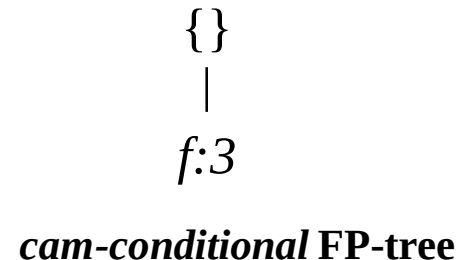
Cond. pattern base of "am": (f:3)



Cond. pattern base of "cm": (f:3)

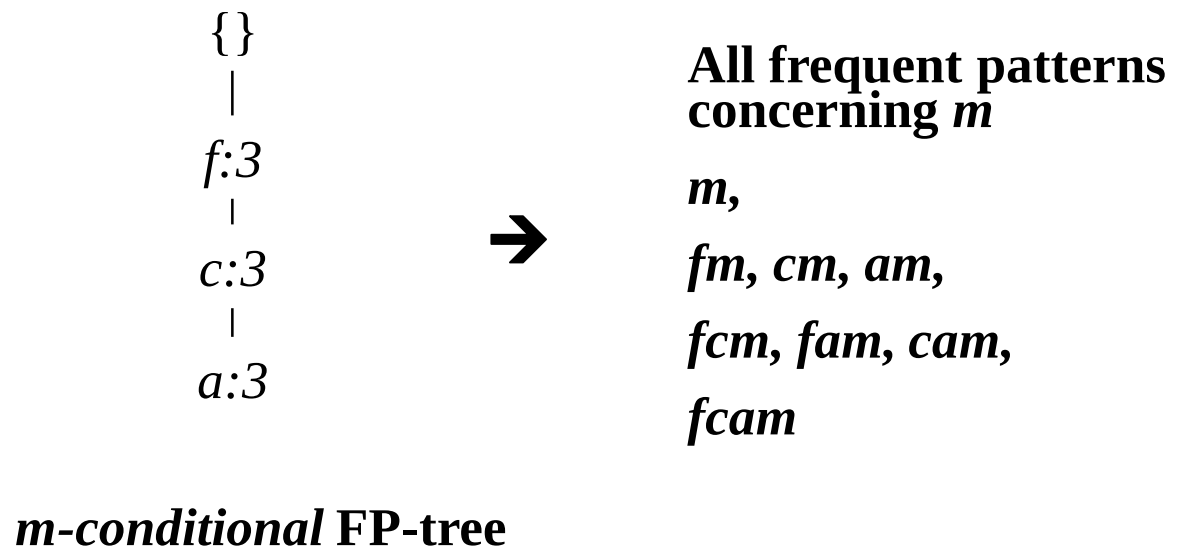


Cond. pattern base of "cam": (f:3)



Single FP-tree Path Generation

- Suppose an FP-tree T has a single path P
- The complete set of frequent pattern of T can be generated by enumeration of all the combinations of the sub-paths of P



Principles of Frequent Pattern Growth

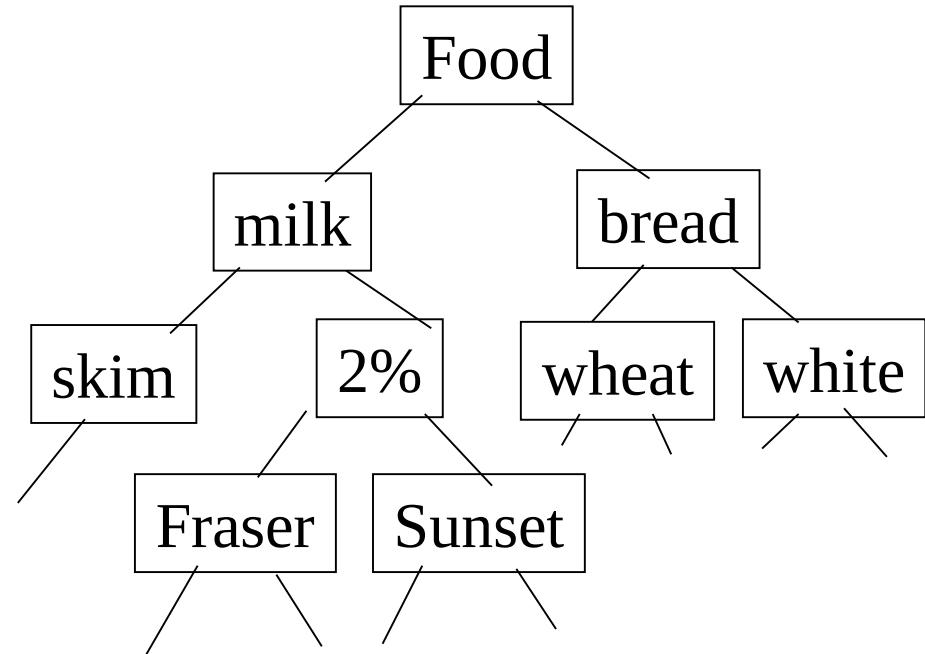
- Pattern growth property
 - Let α be a frequent itemset in DB, B be α 's conditional pattern base, and β be an itemset in B. Then $\alpha \cup \beta$ is a frequent itemset in DB iff β is frequent in B.
- “*abcdef*” is a frequent pattern, if and only if
 - “*abcde*” is a frequent pattern, and
 - “*f*” is frequent in the set of transactions containing “*abcde*”

Why Is Frequent Pattern Growth Fast?

- Our performance study shows
 - FP-growth is an order of magnitude faster than Apriori, and is also faster than tree-projection
- Reasoning
 - No candidate generation, no candidate test
 - Use compact data structure
 - Eliminate repeated database scan
 - Basic operation is counting and FP-tree building

Multiple-Level Association Rules

- Items often form hierarchies.
- Items at the lower level are expected to have lower support.
- Rules regarding itemsets at appropriate levels could be quite useful.
- Transaction database can be encoded based on dimensions and levels
- We can explore shared multi-level mining



TID	Items
T1	{111, 121, 211, 221}
T2	{111, 211, 222, 323}
T3	{112, 122, 221, 411}
T4	{111, 121}
T5	{111, 122, 211, 221, 413}

Mining Multi-Level Associations

- A top_down, progressive deepening approach:
 - First find high-level strong rules:
milk $\rightarrow$ bread [20%, 60%].
 - Then find their lower-level “weaker” rules:
2% milk $\rightarrow$ wheat bread [6%, 50%].
- Variations at mining multiple-level association rules.
 - Level-crossed association rules:
2% milk $\rightarrow$ *Wonder* wheat bread
 - Association rules with multiple, alternative hierarchies:
2% milk $\rightarrow$ *Wonder* bread

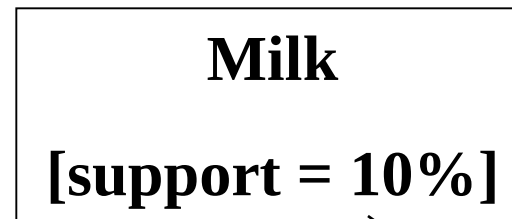
Multi-level Association: Uniform Support vs. Reduced Support

- **Uniform Support:** the same minimum support for all levels
 - + One minimum support threshold. No need to examine itemsets containing any item whose ancestors do not have minimum support.
 - — Lower level items do not occur as frequently. If support threshold
 - too high $\Rightarrow$ miss low level associations
 - too low $\Rightarrow$ generate too many high level associations
- **Reduced Support:** reduced minimum support at lower levels
 - There are 4 search strategies:
 - Level-by-level independent
 - Level-cross filtering by k-itemset
 - Level-cross filtering by single item
 - Controlled level-cross filtering by single item (level passage threshold)

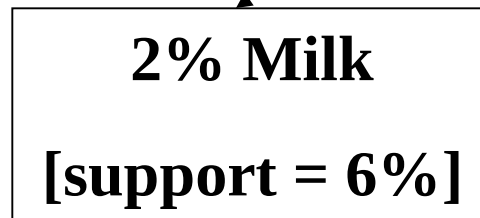
Uniform Support

Multi-level mining with uniform support

Level 1
min_sup = 5%



Level 2
min_sup = 5%



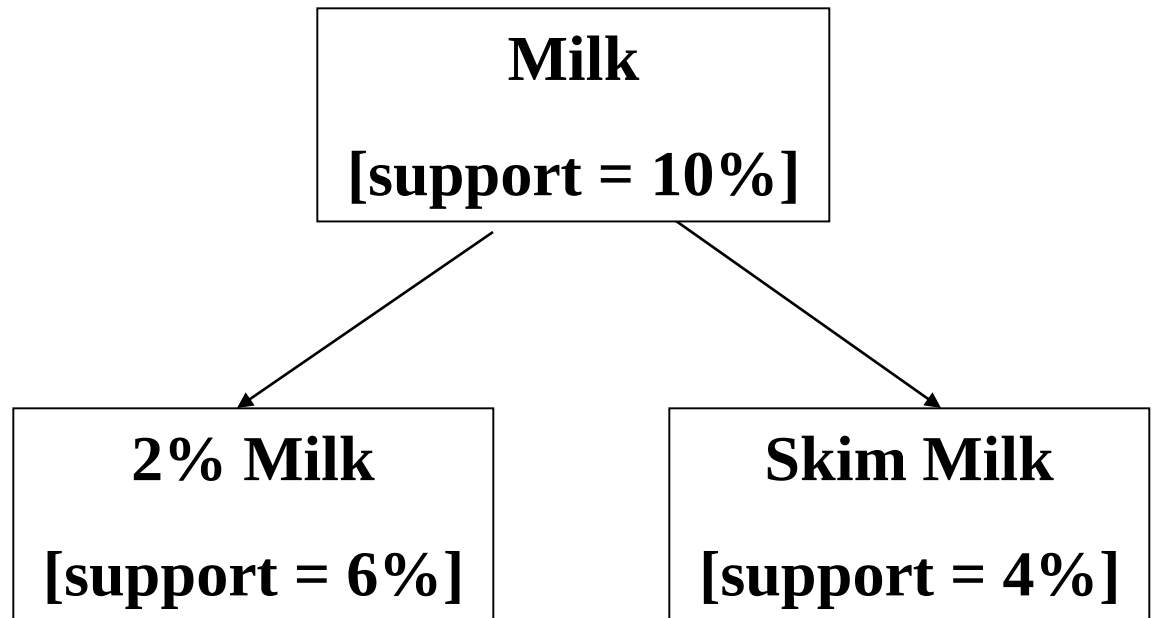
[Back](#)

Reduced Support

Multi-level mining with reduced support

Level 1
min_sup = 5%

Level 2
min_sup = 3%



[Back](#)

Multi-level Association: Redundancy Filtering

- Some rules may be **redundant** due to “ancestor” relationships between items.
- Example
 - **milk $\Rightarrow$ wheat bread** [support = 8%, confidence = 70%]
 - **2% milk $\Rightarrow$ wheat bread** [support = 2%, confidence = 72%]
- We say the first rule is an ancestor of the second rule.
- A rule is redundant if its support is close to the “expected” value, based on the rule’s ancestor.

Multi-Level Mining: Progressive Deepening

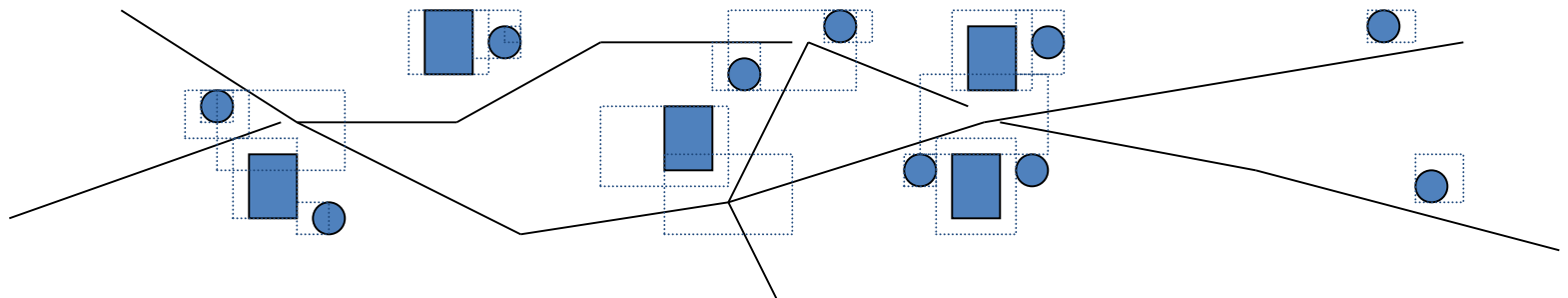
- A top-down, progressive deepening approach:
 - First mine high-level frequent items:
milk (15%), bread (10%)
 - Then mine their lower-level “weaker” frequent itemsets:
2% milk (5%), wheat bread (4%)
- Different `min_support` threshold across multi-levels lead to different algorithms:
 - If adopting the same *min_support* across multi-levels then toss t if any of t 's ancestors is infrequent.
 - If adopting reduced *min_support* at lower levels then examine only those descendents whose ancestor's support is frequent/non-negligible.

Progressive Refinement of Data Mining Quality

- Why progressive refinement?
 - Mining operator can be expensive or cheap, fine or rough
 - Trade speed with quality: step-by-step refinement.
- Superset coverage property:
 - Preserve all the positive answers—allow a positive false test but not a false negative test.
- Two- or multi-step mining:
 - First apply rough/cheap operator (superset coverage)
 - Then apply expensive algorithm on a substantially reduced candidate set (Koperski & Han, [SSD'95](#)).

Progressive Refinement Mining of Spatial Association Rules

- Hierarchy of spatial relationship:
 - “g_close_to”: near_by, touch, intersect, contain, etc.
 - First search for rough relationship and then refine it.
- Two-step mining of spatial association:
 - Step 1: rough spatial computation (as a filter)
 - Using MBR or R-tree for rough estimation.
 - Step2: Detailed spatial algorithm (as refinement)
 - Apply only to those objects which have passed the rough spatial association test (no less than *min_support*)



Multi-Dimensional Association: Concepts

- Single-dimensional rules:
 $\text{buys}(X, \text{"milk"}) \Rightarrow \text{buys}(X, \text{"bread"})$
- Multi-dimensional rules: ○ 2 dimensions or predicates
 - Inter-dimension association rules (*no repeated predicates*)
 $\text{age}(X, \text{"19-25"}) \wedge \text{occupation}(X, \text{"student"}) \Rightarrow \text{buys}(X, \text{"coke"})$
 - hybrid-dimension association rules (*repeated predicates*)
 $\text{age}(X, \text{"19-25"}) \wedge \text{buys}(X, \text{"popcorn"}) \Rightarrow \text{buys}(X, \text{"coke"})$
- Categorical Attributes
 - finite number of possible values, no ordering among values
- Quantitative Attributes
 - numeric, implicit ordering among values

Techniques for Mining MD Associations

- Search for frequent k -predicate set:
 - Example: {age, occupation, buys} is a 3-predicate set.
 - Techniques can be categorized by how age is treated:
- 1. Using static discretization of quantitative attributes
 - Quantitative attributes are statically discretized by using predefined concept hierarchies.
- 2. Quantitative association rules
 - Quantitative attributes are dynamically discretized into “bins” based on the distribution of the data.
- 3. Distance-based association rules
 - This is a dynamic discretization process that considers the distance between data points.

Constraint-Based Mining

- Interactive, exploratory mining giga-bytes of data?
 - Could it be real? — Making good use of constraints!
- What kinds of constraints can be used in mining?
 - **Knowledge type constraint**: classification, association, etc.
 - **Data constraint**: SQL-like queries
 - Find product pairs sold together in [Vancouver](#) in [Dec.'98](#).
 - **Dimension/level constraints**:
 - in relevance to [region](#), [price](#), [brand](#), [customer category](#).
 - **Rule constraints**
 - On the form of the rules to be mined (e.g., # of predicates, etc)
 - small sales (price < \$10) triggers big sales (sum > \$200).
 - **Interestingness constraints**:
 - Thresholds on measures of interestingness
 - strong rules (min_support $\geq$ 3%, min_confidence $\geq$ 60%).

Rule Constraints in Association Mining

- Two kind of rule constraints:
 - Rule form constraints: meta-rule guided mining.
 - $P(x, y) \wedge Q(x, w) \rightarrow \text{takes}(x, \text{"database systems"})$.
 - Rule (content) constraint: constraint-based query optimization (Ng, et al., SIGMOD'98).
 - $\text{sum}(\text{LHS}) < 100 \wedge \text{min}(\text{LHS}) > 20 \wedge \text{count}(\text{LHS}) > 3 \wedge \text{sum}(\text{RHS}) > 1000$
- 1-variable vs. 2-variable constraints (Lakshmanan, et al. SIGMOD'99):
 - 1-var: A constraint confining only one side (L/R) of the rule, e.g., as shown above.
 - 2-var: A constraint confining both sides (L and R).
 - $\text{sum}(\text{LHS}) < \text{min}(\text{RHS}) \wedge \text{max}(\text{RHS}) < 5 * \text{sum}(\text{LHS})$

Interestingness Measures

- Objective measures

Two popular measurements:

★ *support*; and

🕒 *confidence*

- Subjective measures (Silberschatz & Tuzhilin, KDD95)

A rule (pattern) is interesting if

★ it is *unexpected* (surprising to the user); and/or

🕒 *actionable* (the user can do something with it)

- From association to correlation and causal structure analysis.

- Association does not necessarily imply correlation or causal relationships

Criticism to Support and Confidence

- Example 1: (Aggarwal & Yu, PODS98)
 - Among 5000 students
 - 3000 play basketball
 - 3750 eat cereal
 - 2000 both play basket ball and eat cereal
 - *play basketball* $\Rightarrow$ *eat cereal* [40%, 66.7%] is misleading because the overall percentage of students eating cereal is 75% which is higher than 66.7%.
 - *play basketball* $\Rightarrow$ *not eat cereal* [20%, 33.3%] is far more accurate, although with lower support and confidence

	basketball	not basketball	sum(row)
cereal	2000	1750	3750
not cereal	1000	250	1250
sum(col.)	3000	2000	5000

Criticism to Support and Confidence

- Example 2:

- X and Y: positively correlated,
- X and Z, negatively related
- support and confidence of $X \Rightarrow Z$ dominates

X	1	1	1	1	0	0	0	0
Y	1	1	0	0	0	0	0	0
Z	0	1	1	1	1	1	1	1

- We need a measure of dependent or correlated events

$$corr_{A,B} = \frac{P(A \cup B)}{P(A)P(B)}$$

Rule	Support	Confidence
$X \Rightarrow Y$	25%	50%
$X \Rightarrow Z$	37.50%	75%

- ...= $P(B|A)/P(B)$ is also called the **lift** of rule $A \Rightarrow B$

Other Interestingness Measures: Interest

- Interest (correlation, lift) $\frac{P(A \wedge B)}{P(A)P(B)}$
 - taking both $P(A)$ and $P(B)$ in consideration
 - $P(A \wedge B) = P(B) * P(A)$, if A and B are independent events
 - A and B negatively correlated, if the value is less than 1;
otherwise A and B positively correlated

X	1	1	1	1	0	0	0	0
Y	1	1	0	0	0	0	0	0
Z	0	1	1	1	1	1	1	1

Itemset	Support	Interest
X,Y	25%	2
X,Z	37.50%	0.9
Y,Z	12.50%	0.57

Summary

- Association rule mining
 - probably the most significant contribution from the database community in KDD
 - large number of papers
- Many interesting issues have been explored
- An interesting research direction
 - Association analysis in other types of data: spatial data, multimedia data, time series data, etc.

References

1. Kamber and Han, “Data Mining Concepts and Techniques”, Hartcourt India P. Ltd.
2. <https://www.geeksforgeeks.org/>
3. <https://www.knowledgehut.com/>

UNIT III

Classification: Basic Concepts

Classification is a form of data analysis that extracts models describing important data classes. Such models, called classifiers, predict categorical (discrete, unordered) class labels. For example, we can build a classification model to categorize bank loan applications as either safe or risky. Such analysis can help provide us with a better understanding of the data at large.

What Is Classification?

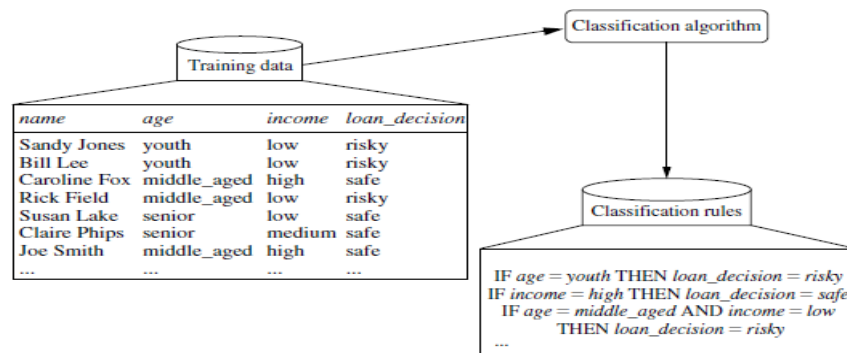
A bank loans officer needs analysis of her data to learn which loan applicants are “safe” and which are “risky” for the bank. A marketing manager at AllElectronics needs data analysis to help guess whether a customer with a given profile will buy a new computer. A medical researcher wants to analyze breast cancer data 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 class (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.

Suppose that the marketing manager wants to predict how much a given customer will spend during a sale at AllElectronics. This data analysis task is an example of numeric prediction, where the model constructed predicts a continuous-valued function, or ordered value, as opposed to a class label. This model is a predictor. Regression analysis is a statistical methodology that is most often used for numeric prediction;

General Approach to Classification

“How does classification work?” Data classification is a two-step process, consisting of a learning step (where a classification model is constructed) and a classification step (where the model is used to predict class labels for given data).

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, X , is represented by an n -dimensional attribute vector, $X = (x_1, x_2, \dots, x_n)$, depicting n measurements made on the tuple from n database attributes, respectively, $A_1, A_2, \dots, A_n$. Each tuple, 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 (or nominal) in that each value serves as a category or class. The individual tuples making up the training set are referred to as training tuples.



(a)

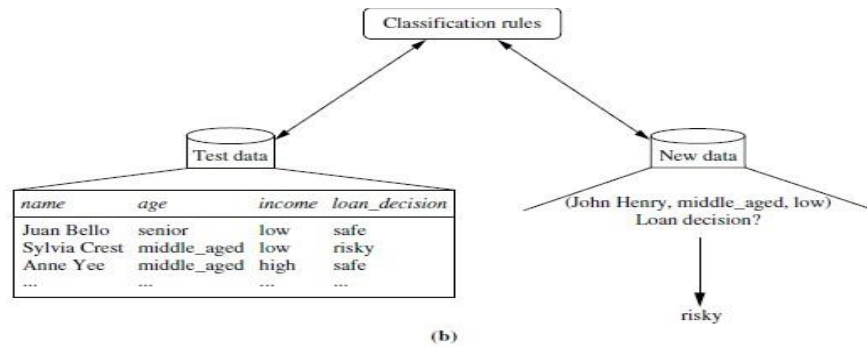


Figure 8.1 The data classification process: (a) *Learning*: Training data are analyzed by a classification algorithm. Here, the class label attribute is *loan_decision*, and the learned model or classifier is represented in the form of classification rules. (b) *Classification*: Test data are used to estimate the accuracy of the classification rules. If the accuracy is considered acceptable, the rules can be applied to the classification of new data tuples.

Because the class label of each training tuple is provided, this step is also known as supervised learning (i.e., the learning of the classifier is “supervised” in that it is told to which class each training tuple belongs). It contrasts with unsupervised learning (or clustering), in which the class label of each training tuple is not known, and the number or set of classes to be learned may not be known in advance. For example, if we did not have the loan decision data available for the training set, we could use clustering to try to determine “groups of like tuples,” which may correspond to risk groups within the loan application data.

“What about classification accuracy?” In the second step (Figure 8.1b), the model is used for classification. First, the predictive accuracy of the classifier is estimated. If we were to use the training set to measure the classifier’s accuracy, this estimate would likely be optimistic, because the classifier tends to overfit the data (i.e., during learning it may incorporate some particular anomalies of the training data that are not present in the general data set overall). Therefore, a test set is used, made up of test tuples and their associated class labels. They are independent of the training tuples, meaning that they were not used to construct the classifier. The accuracy of a classifier on a given test set is the percentage of test set tuples that are correctly classified by the classifier.

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.

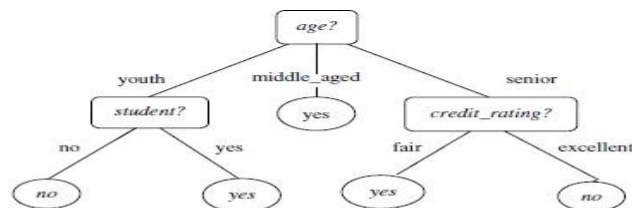


Figure 8.2 A decision tree for the concept *buys_computer*, indicating whether an *AllElectronics* customer is likely to purchase a computer. Each internal (nonleaf) node represents a test on an attribute. Each leaf node represents a class (either *buys_computer* = *yes* or *buys_computer* = *no*).

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

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

During the late 1970s and early 1980s, J. Ross Quinlan, a researcher in machine learning, developed a decision tree algorithm known as ID3 (Iterative Dichotomiser). This work expanded on earlier work on concept learning systems, described by E. B. Hunt, J. Marin, and P. T. Stone. Quinlan later presented C4.5 (a successor of ID3), which became a benchmark to which newer supervised learning algorithms are often compared. In 1984, a group of statisticians (L. Breiman, J. Friedman, R. Olshen, and C. Stone) published the book Classification and Regression Trees (CART), which described the generation of binary decision trees. ID3 and CART were invented independently of one another at around the same time, yet follow a similar approach for learning decision trees from training tuples.

ID3, C4.5, and CART adopt a greedy (i.e., nonbacktracking) approach in which decision trees are constructed in a top-down recursive divide-and-conquer manner. Most algorithms for decision tree induction also follow a top-down approach, which starts with a training set of tuples and their associated class labels. A basic decision tree algorithm is summarized as follows

Strategy:

- The tree starts as a single node, N , representing the training tuples in D (step 1).
- If the tuples in D are all of the same class, then node N becomes a leaf and is labeled with that class (steps 2 and 3). Note that steps 4 and 5 are terminating conditions. Otherwise, the algorithm calls Attribute selection method to determine the splitting criterion. The splitting criterion tells us which attribute to test at node N by determining the “best” way to separate or partition the tuples in D into individual classes (step 6) - a split-point or a splitting subset
A partition is pure if all the tuples in it belong to the same class.
- The node N is labeled with the splitting criterion, which serves as a test at the node (step 7). Let A be the splitting attribute. A has v distinct values, $\{a_1, a_2, \dots, a_v\}$, based on the training data.
 1. A is discrete-valued:
 2. A is continuous-valued:
 3. A is discrete-valued and a binary tree must be produced (as dictated by the attributeselection measure or algorithm being used):
 4. The algorithm uses the same process recursively to form a decision tree for the tuples at each resulting partition, D_j , of D (step 14).
 5. The recursive partitioning stops only when any one of the following terminating conditions is true:
 1. All the tuples in partition D (represented at node N) belong to the same class (steps 2 and 3).
- 2. There are no remaining attributes on which the tuples may be further partitioned (step 4). In this case, majority voting is employed (step 5). This involves converting node N into a leaf and labeling it

with the most common class in D . Alternatively, the class distribution of the node tuples may bestored

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 ;
- (15) **endfor**
- (16) return N ;

Figure 8.3 Basic algorithm for inducing a decision tree from training tuples.

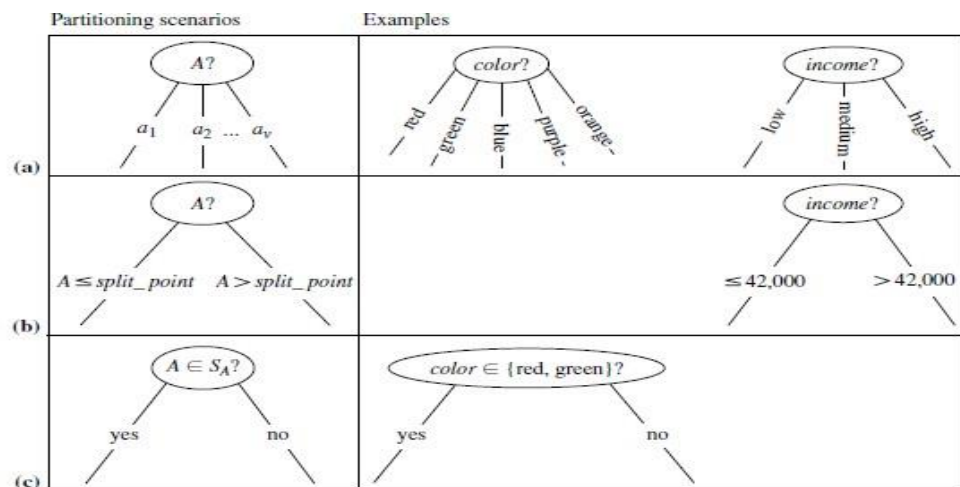


Figure 8.4 This figure shows three possibilities for partitioning tuples based on the splitting criterion, each with examples. Let A be the splitting attribute. (a) If A is discrete-valued, then one branch is grown for each known value of A . (b) If A is continuous-valued, then two branches are grown, 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 .

3. There are no tuples for a given branch, that is, a partition D_j is empty (step 12). In this case, a leaf is created with the majority class in D (step 13).

- The resulting decision tree is returned (step 15).

The computational complexity of the algorithm given training set D is $O(n \times |D| \times \log(|D|))$, where n is the number of attributes describing the tuples in D and $|D|$ is the number of training tuples in D . **Attribute Selection Measures**

An attribute selection measure is a heuristic for selecting the splitting criterion that “best” separates a given data partition, D , of class-labeled training tuples into individual classes. If we were to split D into smaller partitions according to the outcomes of the splitting criterion, ideally each partition would be pure (i.e., all the tuples that fall into a given partition would belong to the same class).

The attribute selection measure provides a ranking for each attribute describing the given training tuples. The attribute having the best score for the measure⁴ is chosen as the splitting attribute for the given tuples. If the splitting attribute is continuous-valued or if we are restricted to binary trees, then, respectively, either a split point or a splitting subset must also be determined as part of the splitting criterion. The three popular attribute selection measures—information gain, gain ratio, and Gini index.

Information Gain

ID3 uses information gain as its attribute selection measure. This measure is based on pioneering work by Claude Shannon on information theory, which studied the value or “information content” of messages.

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

$$Info(D) = - \sum_{i=1}^m p_i \log_2(p_i), \quad (8.1)$$

where p_i is the nonzero probability that an arbitrary tuple in D belongs to class C_i and is estimated by $|C_{i,D}|/|D|$.

A log function to the base 2 is used, because the information is encoded in bits. $Info(D)$ is just the average amount of information needed to identify the class label of a tuple in D . $Info(D)$ is also known as the entropy of D .

Now, suppose we were to partition the tuples in D on some attribute A having v distinct values, $\{a_1, a_2, \dots, a_v\}$, as observed from the training data. If A is discrete-valued, these values correspond directly to the v outcomes of a test on A . Attribute A can be used to split D into v partitions or subsets, $\{D_1, D_2, \dots, D_v\}$,

where D_j contains those tuples in D that have outcome a_j of A .

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

$$Info_A(D) = \sum_{j=1}^v \frac{|D_j|}{|D|} \times Info(D_j). \quad (8.2)$$

The term $\frac{|D_j|}{|D|}$ acts as the weight of the j th partition. $Info_A(D)$ is the expected information required to classify a tuple from D based on the partitioning by A . The smaller the expected information (still) required, the greater the purity of the partitions.

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

$$Gain(A) = Info(D) - Info_A(D). \quad (8.3)$$

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

Example 8.1 Induction of a decision tree using information gain. Table 8.1 presents a training set, D , of class-labeled tuples randomly selected from the *AlIElectronics* customer database. (The data are adapted from Quinlan [Qui86]. 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*); therefore, there are two distinct classes (i.e., $m = 2$). Let class C_1 correspond to *yes* and class C_2 correspond to *no*.

There are nine tuples of class *yes* and five tuples of class *no*. A (root) node N is created for the tuples in D . To find the splitting criterion for these tuples, we must compute the information gain of each attribute. We first use Eq. (8.1) to compute the expected information needed to classify a tuple in D :

Table 8.1 Class-Labeled Training Tuples from the *AlIElectronics* Customer Database

<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

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

Next, we need to compute the expected information requirement for each attribute. Let's start with the attribute *age*. We need to look at the distribution of *yes* and *no* tuples for each category of *age*. For the *age* category "youth," there are two *yes* tuples and three *no* tuples. For the category "middle_aged," there are four *yes* tuples and zero *no* tuples. For the category "senior," there are three *yes* tuples and two *no* tuples. Using Eq. (8.2), the expected information needed to classify a tuple in D if the tuples are partitioned according to *age* is

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

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

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

Similarly, we can compute $\text{Gain.income}/D$ 0.029 bits, $\text{Gain.student}/D$ 0.151 bits, and $\text{Gain.credit rating}/D$ 0.048 bits. Because age has the highest information gain among the attributes, it is selected as the splitting attribute. Node N is labeled with age, and branches are grown for each of the attribute's values. The tuples are then partitioned accordingly, as shown in Figure 8.5. Notice that the tuples falling into the partition for age D 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 "yes." The final decision tree returned by the algorithm was shown earlier in Figure 8.2.

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

We first sort the values of A in increasing order. Typically, the midpoint between each pair of adjacent values is considered as a possible split-point. Therefore, given v values of A, then v - 1 possible splits are evaluated. For example, the midpoint between the values a_i and a_{i+1} of A is

$$\frac{a_i + a_{i+1}}{2}. \quad (8.4)$$

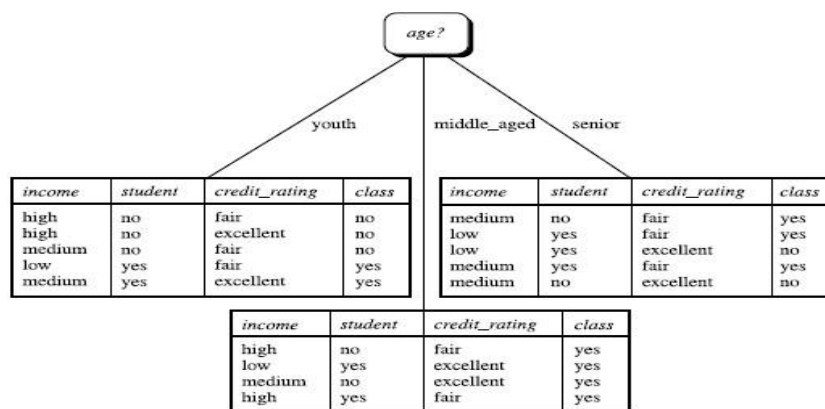


Figure 8.5 The attribute *age* has the highest information gain and therefore becomes the splitting attribute at the root node of the decision tree. Branches are grown for each outcome of *age*. The tuples are shown partitioned accordingly.

The point with the minimum expected information requirement for A is selected as the split point for A. D1 is the set of tuples in D satisfying $A \leq \text{split point}$, and D2 is the set of tuples in D satisfying $A > \text{split point}$.

Gain Ratio

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

$$\text{SplitInfo}_A(D) = - \sum_{j=1}^v \frac{|D_j|}{|D|} \times \log_2 \left(\frac{|D_j|}{|D|} \right). \quad (8.5)$$

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 . The gain ratio is defined as

$$\text{GainRatio}(A) = \frac{\text{Gain}(A)}{\text{SplitInfo}_A(D)}. \quad (8.6)$$

The attribute with the maximum gain ratio is selected as the splitting attribute. Note, however, that as the split information approaches 0, the ratio becomes unstable.

Example 8.2 Computation of gain ratio for the attribute income. A test on income splits the data of Table 8.1 into three partitions, namely low, medium, and high, containing four, six, and four tuples, respectively. To compute the gain ratio of income, we first use Eq. (8.5) to obtain

$$\begin{aligned} \text{SplitInfo}_{\text{income}}(D) &= -\frac{4}{14} \times \log_2 \left(\frac{4}{14} \right) - \frac{6}{14} \times \log_2 \left(\frac{6}{14} \right) - \frac{4}{14} \times \log_2 \left(\frac{4}{14} \right) \\ &= 1.557. \end{aligned}$$

From Example 8.1, we have $\text{Gain}(\text{income}) = 0.029$. Therefore, $\text{GainRatio}(\text{income}) = 0.029/1.557 = 0.019$.

Gini Index

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

$$\text{Gini}(D) = 1 - \sum_{i=1}^m p_i^2, \quad (8.7)$$

where p_i is the probability that a tuple in D belongs to class C_i and is estimated by $|C_i \cap D|/|D|$. The sum is computed over m classes.

The Gini index considers a binary split for each attribute. Let's first consider the case where A is a discrete-valued attribute having v distinct values, $\{a_1, a_2, \dots, a_v\}$, occurring in D . To determine the best binary split on A , we examine all 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 of the form " $A \in SA$?" Given a tuple, this test is satisfied if the value of A for the tuple is among the values listed in SA . If A has v possible values, then there are 2^v possible subsets.

When considering a binary split, we compute a weighted sum of the impurity of each resulting partition. For example, if a binary split on A partitions D into D_1 and D_2 , the Gini index of D given that partitioning is

$$\text{Gini}_A(D) = \frac{|D_1|}{|D|} \text{Gini}(D_1) + \frac{|D_2|}{|D|} \text{Gini}(D_2). \quad (8.8)$$

For each attribute, each of the possible binary splits is considered. For a discrete-valued attribute, the subset that gives the minimum Gini index for that attribute is selected as its splitting subset.

For continuous-valued attributes, each possible split-point must be considered. The strategy is similar to that described earlier for information gain, where the midpoint between each pair of (sorted) adjacent values is taken as a possible split-point.

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

$$\Delta Gini(A) = Gini(D) - Gini_A(D). \quad (8.9)$$

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

Example 8.3 Induction of a decision tree using the Gini index. Let D be the training data shown earlier in Table 8.1, where there are nine tuples belonging to the class buys computer = yes and the remaining five tuples belong to the class buys computer = no. A (root) node N is created for the tuples in D . We first use Eq. (8.7) for the Gini index to compute the impurity of D :

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

To find the splitting criterion for the tuples in D , we need to compute the Gini index for each attribute. Let's start with the attribute *income* and consider each of the possible splitting subsets. Consider the subset {low, medium}. This would result in 10 tuples in partition D_1 satisfying the condition "*income* $\in$ {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

$$\begin{aligned} Gini_{income \in \{low, medium\}}(D) &= \frac{10}{14} Gini(D_1) + \frac{4}{14} Gini(D_2) \\ &= \frac{10}{14} \left(1 - \left(\frac{7}{10}\right)^2 - \left(\frac{3}{10}\right)^2\right) + \frac{4}{14} \left(1 - \left(\frac{2}{4}\right)^2 - \left(\frac{2}{4}\right)^2\right) \\ &= 0.443 \\ &= Gini_{income \in \{high\}}(D). \end{aligned}$$

Similarly, the Gini index values for splits on the remaining subsets are 0.458 (for the subsets {low, high} and {medium}) and 0.450 (for the subsets {medium, high} and {low}). Therefore, the best binary split for attribute *income* is on {low, medium} (or {high}) because it minimizes the Gini index. Evaluating *age*, 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 *age* and splitting subset {youth, senior} therefore give the minimum Gini index overall, with a reduction in impurity of $0.459 - 0.357 = 0.102$. The binary split "*age* $\in$ {youth, senior}?" results in the maximum reduction in impurity of the tuples in D and is returned as the splitting criterion. Node N is labeled with the criterion, two branches are grown from it, and the tuples are partitioned accordingly. ■

Other Attribute Selection Measures

Many other attribute selection measures have been proposed. CHAID, a decision tree algorithm that is popular in marketing, uses an attribute selection measure that is based on the statistical test for independence. Other measures include C-SEP (which performs better than information gain and the Gini index in certain cases) and G-statistic (an information theoretic measure that is a close approximation to χ^2 distribution).

Attribute selection measures based on the **Minimum Description Length (MDL)** principle have the least bias toward multivalued attributes. MDL-based measures use encoding techniques to define the "best" decision tree as the one that requires the fewest number of bits to both (1) encode the tree and (2) encode the exceptions to the tree (i.e., cases that are not correctly classified by the tree).

Other attribute selection measures consider **multivariate splits** (i.e., where the partitioning of tuples is based on a *combination* of attributes, rather than on a single attribute).

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.

“How does tree pruning work?” 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).

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.

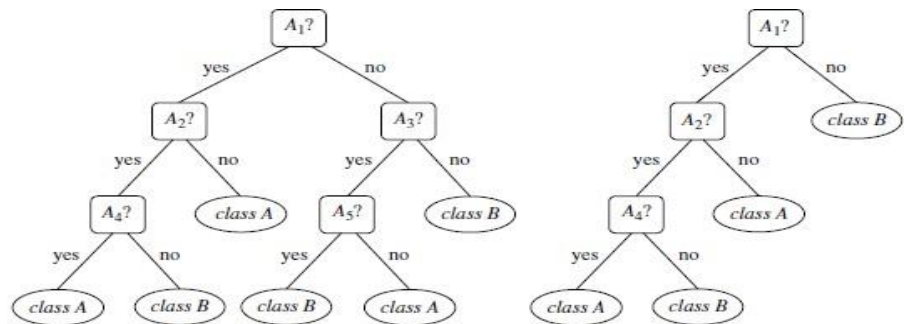
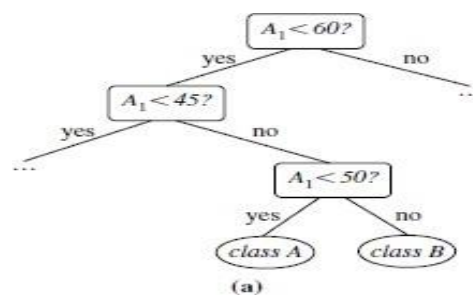


Figure 8.6 An unpruned decision tree and a pruned version of it.

Alternatively, prepruning and postpruning may be interleaved for a combined approach. Postpruning requires more computation than prepruning, yet generally leads to a more reliable tree. No single pruning method has been found to be superior over all others.

Decision trees can suffer from *repetition* and *replication* (Figure 8.7), making them overwhelming to interpret.

Repetition occurs when an attribute is repeatedly tested along a given branch of the tree. In **replication**, duplicate subtrees exist within the tree.



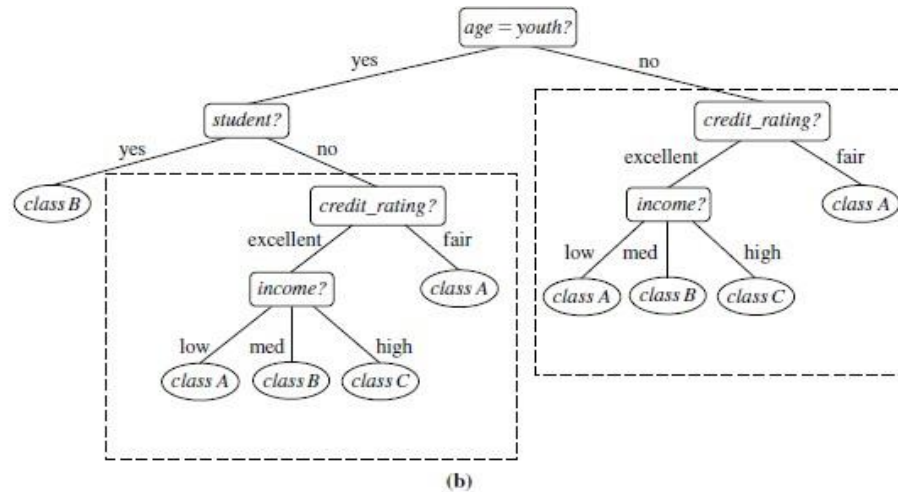


Figure 8.7 An example of: (a) subtree repetition, where an attribute is repeatedly tested along a given branch of the tree (e.g., *age*) and (b) subtree replication, where duplicate subtrees exist within a tree (e.g., the subtree headed by the node “*credit_rating?*”).

Scalability and Decision Tree Induction

“What if D , the disk-resident training set of class-labeled tuples, does not fit in memory? In other words, how scalable is decision tree induction?” The efficiency of existing decision tree algorithms, such as ID3, C4.5, and CART, has been well established for relatively small data sets. Efficiency becomes an issue of concern when these algorithms are applied to the mining of very large real-world databases.

RainForest, for example, adapts to the amount of main memory available and applies to any decision tree induction algorithm. The method maintains an **AVC-set** (where “AVC” stands for “Attribute- Value, Classlabel”) for each attribute, at each tree node, describing the training tuples at the node. The AVC-set of an attribute A at node N gives the class label counts for each value of A for the tuples at N . The set of all AVC-sets at a node N is the **AVC-group** of N . The size of an AVC-set for attribute A at node N depends only on the number of distinct values of A and the number of classes in the set of tuples at N . Typically, this size should fit in memory, even for real-world data. RainForest also has techniques, however, for handling the case where the AVC-group does not fit in memory.

BOAT (Bootstrapped Optimistic Algorithm for Tree construction) is a decision tree algorithm that takes a completely different approach to scalability—it is not based on the use of any special data structures. Instead, it uses a statistical technique known as “bootstrapping” to create several smaller samples (or subsets) of the given training data, each of which fits in memory. Each subset is used to construct a tree, resulting in several trees. The trees are examined and used to construct a new tree, T_0 , that turns out to be “very close” to the tree that would have been generated if all the original training data had fit in memory.

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

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

student	buys_computer	
	yes	no
yes	6	1
no	3	4

credit_rating	buys_computer	
	yes	no
fair	6	2
excellent	3	3

Figure 8.8 The use of data structures to hold aggregate information regarding the training data (e.g., these AVC-sets describing Table 8.1's data) are one approach to improving the scalability of decision tree induction.

BOAT can use any attribute selection measure that selects binary splits and that is based on the notion of purity of partitions such as the Gini index. BOAT uses a lower bound on the attribute selection measure to detect if this “very good” tree, T_0 , is different from the “real” tree, T , that would have been generated using all of the data. It refines T_0 to arrive at T . BOAT usually requires only two scans of D . This is quite an improvement, even in comparison to traditional decision tree algorithms.

Visual Mining for Decision Tree Induction

“Are there any interactive approaches to decision tree induction that allow us to visualize the data and the tree as it is being constructed? Can we use any knowledge of our data to help in building the tree?” In this section, you will learn about an approach to decision tree induction that supports these options. **Perception-based classification (PBC)** is an interactive approach based on multidimensional visualization techniques and allows the user to incorporate background knowledge about the data when building a decision tree. By visually interacting with the data, the user is also likely to develop a deeper understanding of the data. The resulting trees tend to be smaller than those built using traditional decision tree induction methods and so are easier to interpret, while achieving about the same accuracy. The trees constructed with PBC were compared with trees generated by the CART, C4.5, and SPRINT algorithms from various data sets.

Bayes Classification Methods

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

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

Bayes' Theorem

Bayes' theorem is named after Thomas Bayes, a nonconformist English clergyman who did early work in probability and decision theory during the 18th century. Let $\mathbf{X}$ be a data tuple. In Bayesian terms, $\mathbf{X}$ is considered “evidence.” As usual, it is described by measurements made on a set of n attributes. Let H be some hypothesis such as that the data tuple $\mathbf{X}$ belongs to a specified class C . For classification problems, we want to determine $P(H|\mathbf{X})$, the probability that the hypothesis H holds given the “evidence” or observed data tuple $\mathbf{X}$. In other words, we are looking for the probability that tuple $\mathbf{X}$ belongs to class C , given that we know the attribute description of $\mathbf{X}$.

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

In contrast, $P(H)$ is the **prior probability**, or a *priori probability*, of H . For our example, this is the probability that any given customer will buy a computer, regardless of age, income, or any other information, for that matter.

Similarly, $P(X|H)$ is the posterior probability of X conditioned on H . That is, it is the probability that a customer, X , is 35 years old and earns \$40,000, given that we know the customer will buy a computer.

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

$$P(H|X) = \frac{P(X|H)P(H)}{P(X)}. \quad (8.10)$$

Naïve Bayesian Classification

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

1. Let D be a training set of tuples and their associated class labels. As usual, each tuple is represented by an n -dimensional attribute vector, $X = (x_1, x_2, \dots, x_n)$, depicting n measurements made on the tuple from n attributes, respectively, $A_1, A_2, \dots, A_n$.

2. Suppose that there are m classes, $C_1, C_2, \dots, C_m$. Given a tuple, X , the classifier will predict that X

belongs to the class having the highest posterior probability, conditioned on X . That is, the naïve Bayesian classifier predicts that tuple X belongs to the class C_i if and only if

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

Thus, we maximize $P(C_i|X)$. The class C_i for which $P(C_i|X)$ is maximized is called the *maximum posteriori hypothesis*. By Bayes' theorem (Eq. 8.10),

$$P(C_i|X) = \frac{P(X|C_i)P(C_i)}{P(X)}. \quad (8.11)$$

3. As $P(X)$ is constant for all classes, only $P(X|C_i)P(C_i)$ needs to be maximized. If the class prior probabilities are not known, then it is commonly assumed that the classes are equally likely, that is, $P(C_1) = P(C_2) = \dots = P(C_m)$, and we would therefore maximize $P(X|C_i)$.

Otherwise, we maximize $P(X|C_i)P(C_i)$. Note that the class prior probabilities may be estimated by $P(C_i) = |C_i, D| / |D|$, where

$|C_i, D|$ is the number of training tuples of class C_i in D .

4. Given data sets with many attributes, it would be extremely computationally expensive to compute $P(X|C_i)$. To reduce computation in evaluating $P(X|C_i)$, the naïve assumption of **class-conditional independence** is made. This presumes that the attributes' values are conditionally independent of one another, given the class label of the tuple (i.e., that there are no dependence relationships among the attributes). Thus,

$$P(X|C_i) = \prod_{k=1}^n P(x_k|C_i) \quad (8.12)$$

$$= P(x_1|C_i) \times P(x_2|C_i) \times \dots \times P(x_n|C_i).$$

We can easily estimate the probabilities $P(x_1|C_i), P(x_2|C_i), \dots, P(x_n|C_i)$ from the training tuples. Recall that here x_k refers to the value of attribute A_k for tuple X . For each attribute, we look at whether the attribute is categorical or continuous-valued.

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

(a) If A_k is categorical, then $P(x_k|C_i)$ is the number of tuples of class C_i in D having the value x_k for

A_k , divided by $|C_i|$, the number of tuples of class C_i in D .

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

$$g(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \quad (8.13)$$

so that

$$P(x_k|C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i}). \quad (8.14)$$

For example, let $X = (35, \$40,000)$, where A_1 and A_2 are the attributes *age* and *income*, respectively. Let the class label attribute be *buys_computer*. The associated class label for X is *yes* (i.e., *buys_computer* = *yes*). Let's suppose that *age* has not been discretized and therefore exists as a continuous-valued attribute. Suppose that from the training set, we find that customers in D who buy a computer are 38 ± 12 years of age. In other words, for attribute *age* and this class, we have $\mu = 38$ years and $\sigma = 12$. We can plug these quantities, along with $x_1 = 35$ for our tuple X , into Eq. (8.13) to estimate $P(\text{age} = 35 | \text{buys_computer} = \text{yes})$.

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

$$P(X|C_i)P(C_i) > P(X|C_j)P(C_j) \quad \text{for } 1 \leq j \leq m, j \neq i. \quad (8.15)$$

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

“How effective are Bayesian classifiers?”

In theory, Bayesian classifiers have the minimum error rate in comparison to all other classifiers. However, in practice this is not always the case, owing to inaccuracies in the assumptions made for its use, such as class-conditional independence, and the lack of available probability data.

Example 8.4 Predicting a class label using naïve Bayesian classification. We wish to predict the class label of a tuple using naïve Bayesian classification, given the same training data as in Example 8.3 for decision tree induction. The training data were shown earlier in Table 8.1. The data tuples are

described by the attributes *age*, *income*, *student*, and *credit rating*. The class label attribute, *buys computer*, has two distinct values (namely, {yes, no}). Let C_1 correspond to the class *buys_computer* = yes and C_2 correspond to *buys_computer* = no. The tuple we wish to classify is

$X = (\text{age} = \text{youth}, \text{income} = \text{medium}, \text{student} = \text{yes}, \text{credit_rating} = \text{fair})$

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

$$P(\text{buys_computer} = \text{yes}) = 9/14 = 0.643$$

$$P(\text{buys_computer} = \text{no}) = 5/14 = 0.357$$

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

$$P(\text{age} = \text{youth} | \text{buys_computer} = \text{yes}) = 2/9 = 0.222$$

$$P(\text{age} = \text{youth} | \text{buys_computer} = \text{no}) = 3/5 = 0.600$$

$$P(\text{income} = \text{medium} | \text{buys_computer} = \text{yes}) = 4/9 = 0.444$$

$$P(\text{income} = \text{medium} | \text{buys_computer} = \text{no}) = 2/5 = 0.400$$

$$P(\text{student} = \text{yes} | \text{buys_computer} = \text{yes}) = 6/9 = 0.667$$

$$P(\text{student} = \text{yes} | \text{buys_computer} = \text{no}) = 1/5 = 0.200$$

$$P(\text{credit_rating} = \text{fair} | \text{buys_computer} = \text{yes}) = 6/9 = 0.667$$

$$P(\text{credit_rating} = \text{fair} | \text{buys_computer} = \text{no}) = 2/5 = 0.400$$

Using these probabilities, we obtain

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

Similarly,

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

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

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

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

Therefore, the naïve Bayesian classifier predicts *buys_computer* = yes for tuple X . ■

what happens if we should end up with a probability value of zero for some $P(x_k|C_i)$?

There is a simple trick to avoid this problem. We can assume that our training database, D , is so large that adding one to each count that we need would only make a negligible difference in the estimated probability value, yet would conveniently avoid the case of probability values of zero. This technique for probability estimation is known as the **Laplacian correction** or **Laplace estimator**.

Example 8.5 Using the Laplacian correction to avoid computing probability values of zero. Suppose that for the class *buys_computer* = yes in some training database, D , containing 1000 tuples, we have 0 tuples with *income* = low, 990 tuples with *income* = medium, and 10 tuples with *income* = high. The probabilities of these events, without the Laplacian correction, are 0, 0.990 (from 990/1000), and 0.010 (from 10/1000), respectively. Using the Laplacian correction for the three quantities, we pretend that we have 1 more tuple for each income-value pair. In this way, we instead obtain the following probabilities (rounded up to three decimal places):

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

respectively. The “corrected” probability estimates are close to their “uncorrected” counterparts, yet the zero probability value is avoided. ■

Rule-Based Classification

Using IF-THEN Rules for Classification

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

IF *condition* THEN *conclusion*.

An example is rule *R1*,

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

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

R1: (*age* = *youth*) $\wedge$ (*student* = *yes*) $\Rightarrow$ (*buys_computer* = *yes*).

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

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

That is, a rule’s coverage is the percentage of tuples that are covered by the rule.

$$\text{coverage}(R) = \frac{n_{\text{covers}}}{|D|} \quad (8.16)$$

$$\text{accuracy}(R) = \frac{n_{\text{correct}}}{n_{\text{covers}}}. \quad (8.17)$$

Example 8.6 Rule accuracy and coverage. Let’s go back to our data in Table 8.1. These are classlabeled

tuples from the *AllElectronics* customer database. Our task is to predict whether a customer will buy a computer. Consider rule *R1*, which covers 2 of the 14 tuples. It can correctly classify both tuples. Therefore, $\text{coverage}(R1) = 2/14 = 14.28\%$ and $\text{accuracy}(R1) = 2/2 = 100\%$.

If a rule is satisfied by *X*, the rule is said to be **triggered**. For example, suppose we have

X = (*age* = *youth*, *income* = *medium*, *student* = *yes*, *credit_rating* = *fair*).

We would like to classify *X* according to *buys computer*. *X* satisfies *R1*, which triggers the rule.

If $R1$ is the only rule satisfied, then the rule **fires** by returning the class prediction for X . Note that triggering does not always mean firing because there may be more than one rule that is satisfied! If more than one rule is triggered, we have a potential problem. What if they each specify a different class? Or what if no rule is satisfied by X ?

We tackle the first question. If more than one rule is triggered, we need a **conflict resolution strategy** to figure out which rule gets to fire and assign its class prediction to X . There are many possible strategies. We look at two, namely *size ordering* and *rule ordering*.

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

The **rule ordering** scheme prioritizes the rules beforehand. The ordering may be *class-based* or *rule-based*. With **class-based ordering**, the classes are sorted in order of decreasing “importance” such as by decreasing *order of prevalence*. That is, all the rules for the most prevalent (or most frequent) class come first, the rules for the next prevalent class come next, and so on.

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

Note that in the first strategy, overall the rules are *unordered*. They can be applied in any order when classifying a tuple. That is, a disjunction (logical OR) is implied between each of the rules. Each rule represents a standalone nugget or piece of knowledge. This is in contrast to the rule ordering (decision list) scheme for which rules must be applied in the prescribed order so as to avoid conflicts. Each rule in a decision list implies the negation of the rules that come before it in the list. Hence, rules in a decision list are more difficult to interpret. Now that we have seen how we can handle conflicts, let's go back to the scenario where there is no rule satisfied by X . How, then, can we determine the class label of X ? In this case, a fallback or **default rule** can be set up to specify a default class, based on a training set. This may be the class in majority or the majority class of the tuples that were not covered by any rule. The default rule is evaluated at the end, if and only if no other rule covers X .

Rule Extraction from a Decision Tree

It is easy to understand how decision trees work and they are known for their accuracy. Decision trees can become large and difficult to interpret. In this subsection, we look at how to build a rule-based classifier by extracting IF-THEN rules from a decision tree. In comparison with a decision tree, the IF-THEN rules may be easier for humans to understand, particularly if the decision tree is very large.

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

Example 8.7 Extracting classification rules from a decision tree. The decision tree of Figure 8.2 can be converted to classification IF-THEN rules by tracing the path from the root node to each leaf node in the tree. The rules extracted from Figure 8.2 are as follows:

R1: IF <i>age</i> = <i>youth</i>	AND <i>student</i> = <i>no</i>	THEN <i>buys_computer</i> = <i>no</i>
R2: IF <i>age</i> = <i>youth</i>	AND <i>student</i> = <i>yes</i>	THEN <i>buys_computer</i> = <i>yes</i>
R3: IF <i>age</i> = <i>middle_aged</i>		THEN <i>buys_computer</i> = <i>yes</i>
R4: IF <i>age</i> = <i>senior</i>	AND <i>credit_rating</i> = <i>excellent</i>	THEN <i>buys_computer</i> = <i>yes</i>
R5: IF <i>age</i> = <i>senior</i>	AND <i>credit_rating</i> = <i>fair</i>	THEN <i>buys_computer</i> = <i>no</i>

A disjunction (logical OR) is implied between each of the extracted rules. Because the rules are extracted directly from the tree, they are **mutually exclusive** and **exhaustive**. *Mutually exclusive* means that we cannot have rule conflicts here because no two rules will be triggered for the same tuple. (We have one rule per leaf, and any tuple can map to only one leaf.) *Exhaustive* means there is one rule for each possible attribute–value combination, so that this set of rules does not require a default rule. Therefore, the order of the rules does not matter—they are *unordered*.

“How can we prune the rule set?” For a given rule antecedent, any condition that does not improve the estimated accuracy of the rule can be pruned (i.e., removed), thereby generalizing the rule. C4.5 extracts rules from an unpruned tree, and then prunes the rules using a pessimistic approach similar to its tree pruning method. The training tuples and their associated class labels are used to estimate rule accuracy. However, because this would result in an optimistic estimate, alternatively, the estimate is adjusted to compensate for the bias, resulting in a pessimistic estimate.

Other problems arise during rule pruning, however, as the rules *will no longer be* mutually exclusive and exhaustive. For conflict resolution, C4.5 adopts a **class-based ordering scheme**. It groups together all rules for a single class, and then determines a ranking of these class rule sets. Within a rule set, the rules are not ordered.

Rule Induction Using a Sequential Covering Algorithm

IF-THEN rules can be extracted directly from the training data (i.e., without having to generate a decision tree first) using a **sequential covering algorithm**. The name comes from the notion that the rules are learned *sequentially* (one at a time), where each rule for a given class will ideally *cover* many of the class’s tuples (and hopefully none of the tuples of other classes).

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

Input:

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

Output: A set of IF-THEN rules.

Method:

```

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

Figure 8.10 Basic sequential covering algorithm.

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

“How are rules learned?” Typically, rules are grown in a *general-to-specific* manner (Figure 8.11). We can think of this as a beam search, where we start off with an empty rule and then gradually keep appending attribute tests to it. We append by adding the attribute test as a logical conjunct to the existing condition of the rule antecedent. Suppose our training set, D , consists of loan application data. Attributes regarding each applicant include their age, income, education level, residence, credit rating, and the term of the loan. The classifying attribute is *loan decision*, which indicates whether a loan is accepted (considered safe) or rejected (considered risky). To learn a rule for the class “accept,” we start off with the most general rule possible, that is, the condition of the rule antecedent is empty. The rule is

IF THEN *loan_decision* = *accept*.

We then consider each possible attribute test that may be added to the rule. These can be derived from the parameter *Att_vals*, which contains a list of attributes with their associated values. For example, for an attribute–value pair (*att*, *val*), we can consider attribute tests such as *att* = *val*, *att* ≤ *val*, *att* > *val*, and so on. Typically, the training data will contain many attributes, each of which may have several possible values. Finding an optimal rule set becomes computationally explosive. Instead, *Learn One Rule* adopts a greedy depth-first strategy. Each time it is faced with adding a new attribute test (conjunct) to the current rule, it picks the one that most improves the rule quality, based on the training samples.

so that the current rule becomes

IF *income* = *high* THEN *loan_decision* = *accept*.

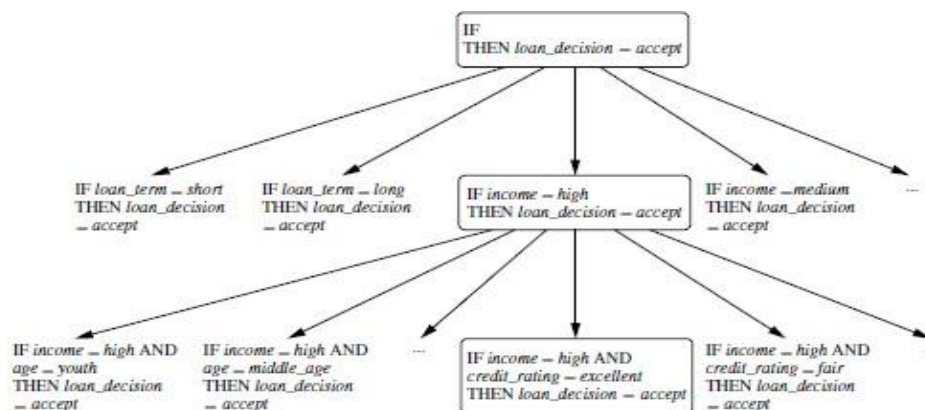


Figure 8.11 A general-to-specific search through rule space.

Each time we add an attribute test to a rule, the resulting rule should cover relatively more of the “accept” tuples. During the next iteration, we again consider the possible attribute tests and end up selecting *credit_rating = excellent*. Our current rule grows to become

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

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

Rule Quality Measures

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

Example 8.8 Choosing between two rules based on accuracy. Consider the two rules as illustrated in Figure 8.12. Both are for the class *loan decision D accept*. We use “a” to represent the tuples of class “accept” and “r” for the tuples of class “reject.” Rule *R1* correctly classifies 38 of the 40 tuples it covers. Rule *R2* covers only two tuples, which it correctly classifies. Their respective accuracies are 95% and 100%. Thus, *R2* has greater accuracy than *R1*, but it is not the better rule because of its small coverage.

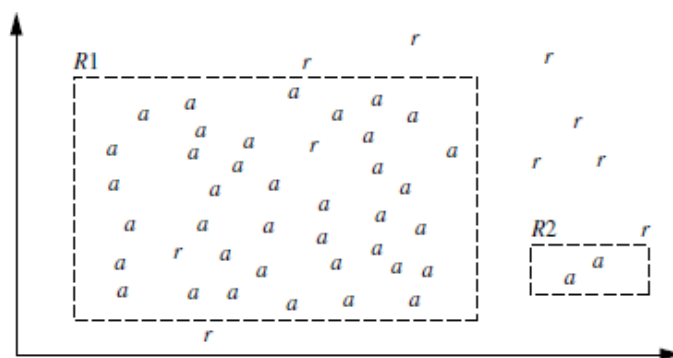


Figure 8.12 Rules for the class *loan_decision = accept*, showing *accept* (a) and *reject* (r) tuples.

Our current rule is *R*: IF *condition* THEN *class = c*. We want to see if logically ANDing a given attribute test to *condition* would result in a better rule. We call the new condition, *condition0*, where *R0*: IF *condition0* THEN *class = c* is our potential new rule. In other words, we want to see if *R0* is any better than *R*.

Another measure is based on information gain and was proposed in **FOIL** (First Order Inductive Learner), a sequential covering algorithm that learns first-order logic rules. Learning first-order rules is more complex because such rules contain variables, whereas the rules we are concerned with in this section are propositional (i.e., variablefree).

FOIL assesses the information gained by extending *condition0* as

$$FOIL_Gain = pos' \times \left(\log_2 \frac{pos'}{pos' + neg'} - \log_2 \frac{pos}{pos + neg} \right). \quad (8.18)$$

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

Rule Pruning

Learn One Rule does not employ a test set when evaluating rules. Assessments of rule quality as described previously are made with tuples from the original training data. These assessments are optimistic because the rules will likely overfit the data. That is, the rules may perform well on the training data, but less well on subsequent data. To compensate for this, we can prune the rules. A rule is pruned by removing a conjunct (attribute test). We choose to prune a rule, R , if the pruned version of R has greater quality, as assessed on an independent set of tuples.

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

$$FOIL_Prune(R) = \frac{pos - neg}{pos + neg}, \quad (8.20)$$

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

Model Evaluation and Selection

But what is accuracy? How can we estimate it? Are some measures of a classifier's accuracy more appropriate than others? How can we obtain a *reliable* accuracy estimate? Answers to these questions can be found here. What if we have more than one classifier and want to choose the "best" one? This is referred to as **model selection** (i.e., choosing one classifier over another).

Metrics for Evaluating Classifier Performance

Here, we discuss measures for assessing how good or how "accurate" your classifier is at predicting the class label of tuples. They include accuracy (also known as recognition rate), sensitivity (or recall), specificity, precision,

$F1$, and $F\beta$. Note that although accuracy is a specific measure, the word "accuracy" is also used as a general term to refer to a classifier's predictive abilities.

Using training data to derive a classifier and then estimate the accuracy of the resulting learned model can result in misleading overoptimistic estimates due to overspecialization of the learning algorithm to the data. Recall that we can talk in terms of **positive tuples** (tuples of the main class of interest) and **negative tuples** (all other tuples). Given two classes, for example, the positive tuples may be *buys computer = yes* while the negative tuples are *buys computer = no*. Suppose we use our classifier on a test set of labeled tuples. P is the number of positive tuples and N is the number of negative tuples. For each tuple, we compare the classifier's class label prediction with the tuple's known class label.

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

Figure 8.13 Evaluation measures. Note that some measures are known by more than one name. TP , TN , FP , P , N refer to the number of true positive, true negative, false positive, positive, and negative samples, respectively (see text).

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

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

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

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

These terms are summarized in the **confusion matrix** of Figure 8.14.

The confusion matrix is a useful tool for analyzing how well your classifier can recognize tuples of different classes. TP and TN tell us when the classifier is getting things right, while FP and FN tell us when the classifier is getting things wrong (i.e., mislabeling). Given m classes (where $m \geq 2$), a **confusion matrix** is a table of at least size m by m . An entry, $CM_{i,j}$ in the first m rows and m columns indicates the number of tuples of class i that were labeled by the classifier as class j . For a classifier to have good accuracy, ideally most of the tuples would be represented along the diagonal of the confusion matrix, from entry $CM_{1,1}$ to entry $CM_{m,m}$, with the rest of the entries being zero or close to zero. That is, ideally, FP and FN are around zero.

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

$$\text{accuracy} = \frac{TP + TN}{P + N}. \quad (8.21)$$

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

Figure 8.14 Confusion matrix, shown with totals for positive and negative tuples.

Classes	<i>buys_computer</i> = yes	<i>buys_computer</i> = no	Total	Recognition (%)
<i>buys_computer</i> = yes	6954	46	7000	99.34
<i>buys_computer</i> = no	412	2588	3000	86.27
Total	7366	2634	10,000	95.42

Figure 8.15 Confusion matrix for the classes *buys_computer* = yes and *buys_computer* = no, where an entry in row *i* and column *j* shows the number of tuples of class *i* that were labeled by the classifier as class *j*. Ideally, the nondiagonal entries should be zero or close to zero.

In the pattern recognition literature, this is also referred to as the overall **recognition rate** of the classifier, that is, it reflects how well the classifier recognizes tuples of the various classes. An example of a confusion matrix for the two classes *buys computer* = yes (positive) and *buys computer* = no (negative) is given in Figure 8.15.

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

$$\text{error rate} = \frac{FP + FN}{P + N}. \quad (8.22)$$

If we were to use the training set (instead of a test set) to estimate the error rate of a model, this quantity is known as the **resubstitution error**. This error estimate is optimistic of the true error rate (and similarly, the corresponding accuracy estimate is optimistic) because the model is not tested on any samples that it has not already seen.

We now consider the **class imbalance problem**, where the main class of interest is rare. That is, the data set distribution reflects a significant majority of the negative class and a minority positive class. For example, in **fraud detection applications**, the class of interest (or positive class) is “*fraud*,” which occurs much less frequently than the negative “*nonfraudulent*” class. In medical data, there may be a rare class, such as “*cancer*.” Suppose that you have trained a classifier to classify medical data tuples, where the class label attribute is “*cancer*” and the possible class values are “yes” and “no.” An accuracy rate of, say, 97% may make the classifier seem quite accurate, but what if only, say, 3% of the training tuples are actually cancer? Clearly, an accuracy rate of 97% may not be acceptable—the classifier could be correctly labeling only the noncancer tuples, for instance, and misclassifying all the cancer tuples. Instead, we need other measures, which access how well the

classifier can recognize the positive tuples (*cancer* = *yes*) and how well it can recognize the negative tuples (*cancer* = *no*).

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

$$\text{sensitivity} = \frac{TP}{P} \quad (8.23)$$

$$\text{specificity} = \frac{TN}{N}. \quad (8.24)$$

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

$$\text{accuracy} = \text{sensitivity} \frac{P}{(P+N)} + \text{specificity} \frac{N}{(P+N)}. \quad (8.25)$$

Example 8.9 Sensitivity and specificity. Figure 8.16 shows a confusion matrix for medical data where the class values are *yes* and *no* for a class label attribute, *cancer*. The sensitivity of the classifier is 90/300 = 30.00%. The specificity is 9560/9700 = 98.56%. The classifier's overall accuracy is 9650/10,000 = 96.50%. Thus, we note that although the classifier has a high accuracy, its ability to correctly label the positive (rare) class is poor given its low sensitivity. It has high specificity, meaning that it can accurately recognize negative tuples.

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

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

The *precision* and *recall* measures are also widely used in classification. **Precision** can be thought of as a measure of *exactness* (i.e., what percentage of tuples labeled as positive are actually such), whereas **recall** is a measure of *completeness* (what percentage of positive tuples are labeled as such). If recall seems familiar, that's because it is the same as sensitivity (or the *true positive rate*). These measures can be computed as

$$\text{precision} = \frac{TP}{TP + FP} \quad (8.26)$$

$$\text{recall} = \frac{TP}{TP + FN} = \frac{TP}{P}. \quad (8.27)$$

Example 8.10 Precision and recall. The precision of the classifier in Figure 8.16 for the *yes* class is 90/230 = 39.13%. The recall is 90/300 = 30.00%.

A perfect precision score of 1.0 for a class C means that every tuple that the classifier labeled as belonging to class C does indeed belong to class C . However, it does not tell us anything about the number of class C tuples that the classifier mislabeled. A perfect recall score of 1.0 for C means that every item from class C was labeled as such, but it does not tell us how many other tuples were incorrectly labeled as belonging to class C . There tends to be an inverse relationship between precision and recall, where it is possible to increase one at the cost of reducing the other. For example, our medical classifier may achieve high precision by labeling all cancer tuples that present a certain way as *cancer*, but may have low recall if it mislabels many other instances of *cancer* tuples. Precision and recall scores are typically used together, where precision values are compared for a fixed value of recall, or vice versa. For example, we may compare precision values at a recall value of, say, 0.75.

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

$$F = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \quad (8.28)$$

$$F_\beta = \frac{(1 + \beta^2) \times \text{precision} \times \text{recall}}{\beta^2 \times \text{precision} + \text{recall}}, \quad (8.29)$$

where β is a non-negative real number. The F measure is the *harmonic mean* of precision and recall (the proof of which is left as an exercise). It gives equal weight to precision and recall. The measure is a F_β weighted measure of precision and recall. It assigns β times as much weight to recall as to precision. Commonly used F_β measures are $F2$ (which weights recall twice as much as precision) and $F0.5$ (which weights precision twice as much as recall).

“Are there other cases where accuracy may not be appropriate?” In classification problems, it is commonly assumed that all tuples are uniquely classifiable, that is, that each training tuple can belong to only one class. Yet, owing to the wide diversity of data in large databases, it is not always reasonable to assume that all tuples are uniquely classifiable. Rather, it is more probable to assume that each tuple may belong to more than one class. How then can the accuracy of classifiers on large databases be measured? The accuracy measure is not appropriate, because it does not take into account the possibility of tuples belonging to more than one class.

Speed: This refers to the computational costs involved in generating and using the given classifier. **Robustness:** This is the ability of the classifier to make correct predictions given noisy data or data with missing values. Robustness is typically assessed with a series of synthetic data sets representing increasing degrees of noise and missing values.

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

Interpretability: This refers to the level of understanding and insight that is provided by the classifier or predictor. Interpretability is subjective and therefore more difficult to assess.

In summary, we have presented several evaluation measures. The accuracy measure works best when the data classes are fairly evenly distributed. Other measures, such as sensitivity (or recall), specificity, precision, F , and F_{β} , are better suited to the class imbalance problem, where the main class of interest is rare.

Holdout Method and Random Subsampling

The **holdout** method is what we have alluded to so far in our discussions about accuracy. In this method, the given data are randomly partitioned into two independent sets, a *training set* and a *test set*. Typically, two-thirds of the data are allocated to the training set, and the remaining one-third is allocated to the test set. The training set is used to derive the model. The model's accuracy is then estimated with the test set.

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

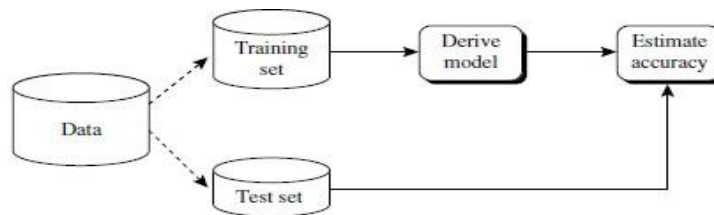


Figure 8.17 Estimating accuracy with the holdout method.

Cross-Validation

In **k -fold cross-validation**, the initial data are randomly partitioned into k mutually exclusive subsets or “folds,” $D_1, D_2, \dots, D_k$, each of approximately equal size. Training and testing is performed k times. In iteration i , partition D_i is reserved as the test set, and the remaining partitions are collectively used to train the model. That is, in the first iteration, subsets $D_2, \dots, D_k$ collectively serve as the training set to obtain a first model, which is tested on D_1 ; the second iteration is trained on subsets $D_1, D_3, \dots, D_k$ and tested on D_2 ; and so on. Unlike the holdout and random subsampling methods, here each sample is used the same number of times for training and once for testing. For classification, the accuracy estimate is the overall number of correct classifications from the k iterations, divided by the total number of tuples in the initial data.

Leave-one-out is a special case of k -fold cross-validation where k is set to the number of initial tuples. That is, only one sample is “left out” at a time for the test set. In **stratified cross-validation**, the folds are stratified so that the class distribution of the tuples in each fold is approximately the same as that in the initial data.

In general, stratified 10-fold cross-validation is recommended for estimating accuracy (even if computation power allows using more folds) due to its relatively low bias and variance.

Bootstrap

the **bootstrap method** samples the given training tuples uniformly *with replacement*. That is, each time a tuple is selected, it is equally likely to be selected again and re-added to the training set. For instance, imagine a machine that randomly selects tuples for our training set. In *sampling with replacement*, the machine is allowed to select the same tuple more than once.

There are several bootstrap methods. A commonly used one is the **.632 bootstrap**, which works as follows. Suppose we are given a data set of d tuples. The data set is sampled d times, with replacement, resulting in a *bootstrap sample* or training set of d samples. It is very likely that some of the original data tuples will occur more than once in this sample. The data tuples that did not make it into the training set end up forming the test set. Suppose we were to try this out several times. As it turns out, on average, 63.2% of the original data tuples will end up in the bootstrap sample, and the remaining 36.8% will form the test set (hence, the name, .632 bootstrap).

“Where does the figure, 63.2%, come from?” Each tuple has a probability of $1/d$ of being selected, so the probability of not being chosen is $(1 - 1/d)$. We have to select d times, so the probability that a tuple will not be chosen during this whole time is $(1 - 1/d)^d$. If d is large, the probability approaches $e^{-1} = 0.368$. Thus, 36.8% of tuples will not be selected for training and thereby end up in the test set, and the remaining 63.2% will form the training set.

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

$$Acc(M) = \frac{1}{k} \sum_{i=1}^k (0.632 \times Acc(M_i)_{test_set} + 0.368 \times Acc(M_i)_{train_set}), \quad (8.30)$$

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

Model Selection Using Statistical Tests of Significance

Suppose that we have generated two classification models, $M1$ and $M2$, from our data. We have performed 10-fold cross-validation to obtain a mean error rate for each. How can we determine which model is best? It may seem intuitive to select the model with the lowest error rate; however, the mean error rates are just *estimates* of error on the true population of future data cases. There can be considerable variance between error rates within any given 10-fold cross-validation experiment. Although the mean error rates obtained for $M1$ and $M2$ may appear different, that difference may not be statistically significant. What if any difference between the two may just be attributed to chance?

What do we need to perform the statistical test? Suppose that for each model, we did 10-fold cross-validation, say, 10 times, each time using a different 10-fold data partitioning. Each partitioning is independently drawn. We can average the 10 error rates obtained each for $M1$ and $M2$, respectively,

to obtain the mean error rate for each model. For a given model, the individual error rates calculated in the cross-validations may be considered as different, independent samples from a probability distribution. In general, they follow a *t-distribution with $k - 1$ degrees of freedom* where, here, $k = 10$. (This distribution looks very similar to a normal, or Gaussian, distribution even though the functions defining the two are quite different. Both are unimodal, symmetric, and bellshaped.) This allows us to do hypothesis testing where the significance test used is the ***t*-test**, or **Student's *t*-test**. Our hypothesis is that the two models are the same, or in other words, that the difference in mean error rate between the two is zero. If we can reject this hypothesis (referred to as the *null hypothesis*), then we can conclude that the difference between the two models is statistically significant, in which case we can select the model with the lower error rate.

In data mining practice, we may often employ a single test set, that is, the same test set can be used for both M_1 and M_2 . In such cases, we do a **pairwise comparison** of the two models *for each* 10-fold cross-validation round. That is, for the i th round of 10-fold cross-validation, the same cross-validation partitioning is used to obtain an error rate for M_1 and for M_2 .

To determine whether M_1 and M_2 are significantly different, we compute t and select a **significance level**, *sig*. In practice, a significance level of 5% or 1% is typically used. We then consult a table for the *t-distribution*

However, because the *t*-distribution is symmetric, typically only the upper percentage points of the distribution are shown. Therefore, we look up the table value for $z = \text{sig}/2$, which in this case is 0.025, where z is also referred to as a **confidence limit**. If $t > z$ or $t < -z$, then our value of t lies in the rejection region, within the distribution's tails. This means that we can reject the null hypothesis that the means of M_1 and M_2 are the same and conclude that there is a statistically significant difference between the two models. Otherwise, if we cannot reject the null hypothesis, we conclude that any difference between M_1 and M_2 can be attributed to chance.

If two test sets are available instead of a single test set, then a nonpaired version of the *t*-test is used, where the variance between the means of the two models is estimated as

$$\text{var}(M_1 - M_2) = \sqrt{\frac{\text{var}(M_1)}{k_1} + \frac{\text{var}(M_2)}{k_2}}, \quad (8.33)$$

and k_1 and k_2 are the number of cross-validation samples (in our case, 10-fold crossvalidation rounds) used for M_1 and M_2 , respectively. This is also known as the **two sample *t*-test**.

Comparing Classifiers Based on Cost–Benefit and ROC Curves

The true positives, true negatives, false positives, and false negatives are also useful in assessing the **costs and benefits** (or risks and gains) associated with a classification model. The cost associated with a false negative (such as incorrectly predicting that a cancerous patient is not cancerous) is far greater than those of a false positive (incorrectly yet conservatively labeling a noncancerous patient as cancerous). In such cases, we can outweigh one type of error over another by assigning a different cost to each. These costs may consider the danger to the patient, financial costs of resulting therapies, and other hospital costs. Similarly, the benefits associated with a true positive decision may be different than those of a true negative. Up to now, to compute classifier accuracy, we

have assumed equal costs and essentially divided the sum of true positives and true negatives by the total number of test tuples.

Receiver operating characteristic curves are a useful visual tool for comparing two classification models. ROC curves come from signal detection theory that was developed during World War II for the analysis of radar images. An ROC curve for a given model shows the trade-off between the *true positive rate* (*TPR*) and the *false positive rate* (*FPR*).¹⁰ Given a test set and a model, *TPR* is the proportion of positive (or “yes”) tuples that are correctly labeled by the model; *FPR* is the proportion of negative (or “no”) tuples that are mislabeled as positive. Given that *TP*, *FP*, *P*, and *N* are the number of true positive, false positive, positive, and negative tuples, respectively, from Section 8.5.1 we know that $TPR = TP/P$, which is sensitivity. Furthermore, $FPR = FP/N$, which is $1 - \text{specificity}$.

For a two-class problem, an ROC curve allows us to visualize the trade-off between the rate at which the model can accurately recognize positive cases versus the rate at which it mistakenly identifies negative cases as positive for different portions of the test set. Any increase in *TPR* occurs at the cost of an increase in *FPR*. The area under the ROC curve is a measure of the accuracy of the model.

To plot an ROC curve for a given classification model, *M*, the model must be able to return a probability of the predicted class for each test tuple. With this information, we rank and sort the tuples so that the tuple that is most likely to belong to the positive or “yes” class appears at the top of the list, and the tuple that is least likely to belong to the positive class lands at the bottom of the list.

Example 8.11 Plotting an ROC curve. Figure 8.18 shows the probability value (column 3) returned by a probabilistic classifier for each of the 10 tuples in a test set, sorted by decreasing probability order. Column 1 is merely a tuple identification number, which aids in our explanation. Column 2 is the actual class label of the tuple. There are five positive tuples and five negative tuples, thus $P = 5$ and $N = 5$. As we examine the known class label of each tuple, we can determine the values of the remaining columns, *TP*, *FP*, *TN*, *FN*, *TPR*, and *FPR*. We start with tuple 1, which has the highest probability score, and take that score as our threshold, that is, $t = 0.9$. Thus, the classifier considers tuple 1 to be positive, and all the other tuples are considered negative. Since the actual class label of tuple 1 is positive, we have a true positive, hence $TP = 1$ and $FP = 0$. Among the

Tuple #	Class	Prob.	TP	FP	TN	FN	TPR	FPR
1	P	0.90	1	0	5	4	0.2	0
2	P	0.80	2	0	5	3	0.4	0
3	N	0.70	2	1	4	3	0.4	0.2
4	P	0.60	3	1	4	2	0.6	0.2
5	P	0.55	4	1	4	1	0.8	0.2
6	N	0.54	4	2	3	1	0.8	0.4
7	N	0.53	4	3	2	1	0.8	0.6
8	N	0.51	4	4	1	1	0.8	0.8
9	P	0.50	5	4	0	1	1.0	0.8
10	N	0.40	5	5	0	0	1.0	1.0

Figure 8.18 Tuples sorted by decreasing score, where the score is the value returned by a probabilistic classifier.

remaining nine tuples, which are all classified as negative, five actually are negative (thus, $TN = 5$). The remaining four are all actually positive, thus, $FN = 4$. We can therefore compute $TPR = TP/P = 1/5 = 0.2$, while $FPR = 0$. Thus, we have the point (0.2,0) for the ROC curve.

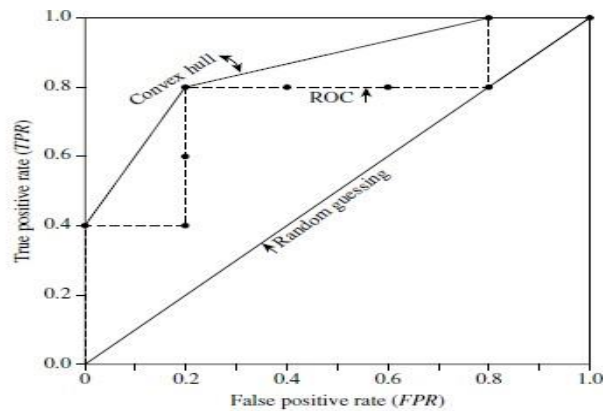


Figure 8.19 ROC curve for the data in Figure 8.18.

Next, threshold t is set to 0.8, the probability value for tuple 2, so this tuple is now also considered positive, while tuples 3 through 10 are considered negative. The actual class label of tuple 2 is positive, thus now $TP = 2$. The rest of the row can easily be computed, resulting in the point (0.4,0). Next, we examine the class label of tuple 3 and let t be 0.7, the probability value returned by the classifier for that tuple. Thus, tuple 3 is considered positive, yet its actual label is negative, and so it is a false positive. Thus, TP stays the same and FP increments so that $FP = 1$. The rest of the values in the row can also be easily computed, yielding the point (0.4, 0.2). The resulting ROC graph, from examining each tuple, is the jagged line shown in Figure 8.19.

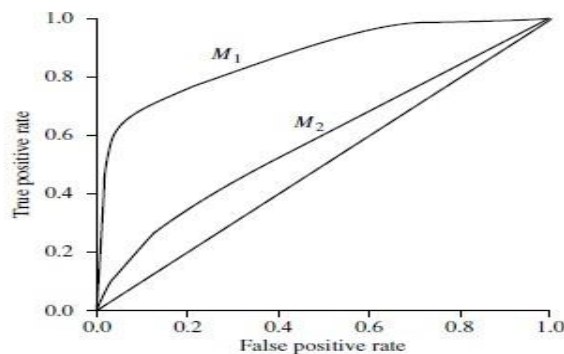


Figure 8.20 ROC curves of two classification models, M_1 and M_2 . The diagonal shows where, for every true positive, we are equally likely to encounter a false positive. The closer an ROC curve is to the diagonal line, the less accurate the model is. Thus, M_1 is more accurate here.

The diagonal line representing random guessing is also shown. Thus, the closer the ROC curve of a model is to the diagonal line, the less accurate the model. If the model is really good, initially we are more likely to encounter true positives as we move down the ranked list. Thus, the curve moves steeply up from zero. Later, as we start to encounter fewer and fewer true positives, and more and more false positives, the curve eases off and becomes more horizontal.

Techniques to Improve Classification Accuracy

In this section, you will learn some tricks for increasing classification accuracy. We focus on *ensemble methods*. An ensemble for classification is a composite model, made up of a combination of classifiers. The individual classifiers vote, and a class label prediction is returned by the ensemble based on the collection of votes. Ensembles tend to be more accurate than their component classifiers. Traditional learning models assume that the data classes are well distributed. In many real-world data domains, however, the data are class-imbalanced, where the main class of interest is represented by only a few tuples. This is known as the *class imbalance problem*.

Introducing Ensemble Methods

Bagging, *boosting*, and *random forests* are examples of **ensemble methods** (Figure 8.21). An ensemble combines a series of k learned models (or *base classifiers*), $M_1, M_2, \dots, M_k$, with the aim of creating an improved composite classification model, M^* . A given data set, D , is used to create k training sets, $D_1, D_2, \dots, D_k$, where D_i ($1 \leq i \leq k$) is used to generate classifier M_i . Given a new data tuple to classify, the base classifiers each vote by returning a class prediction. The ensemble returns a class prediction based on the votes of the base classifiers.

An ensemble tends to be more accurate than its base classifiers. For example, consider an ensemble that performs majority voting. That is, given a tuple X to classify, it collects the class label predictions returned from the base classifiers and outputs the class in majority. The base classifiers may make mistakes, but the ensemble will misclassify X only if over half of the base classifiers are in error. Ensembles yield better results when there is significant diversity among the models. That is, ideally, there is little correlation among classifiers. The classifiers should also perform better than random guessing. Each base classifier can be allocated to a different CPU and so ensemble methods are parallelizable.

To help illustrate the power of an ensemble, consider a simple two-class problem described by two attributes, x_1 and x_2 . The problem has a linear decision boundary. Figure 8.22(a) shows the decision boundary of a decision tree classifier on the problem. Figure 8.22(b) shows the decision boundary of an ensemble of decision tree classifiers on the same problem. Although the ensemble's decision boundary is still piecewise constant, it has a finer resolution and is better than that of a single tree.

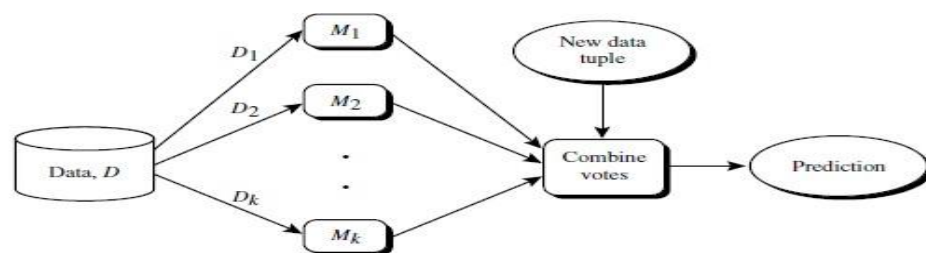


Figure 8.21 Increasing classifier accuracy: Ensemble methods generate a set of classification models, $M_1, M_2, \dots, M_k$. Given a new data tuple to classify, each classifier “votes” for the class label of that tuple. The ensemble combines the votes to return a class prediction.

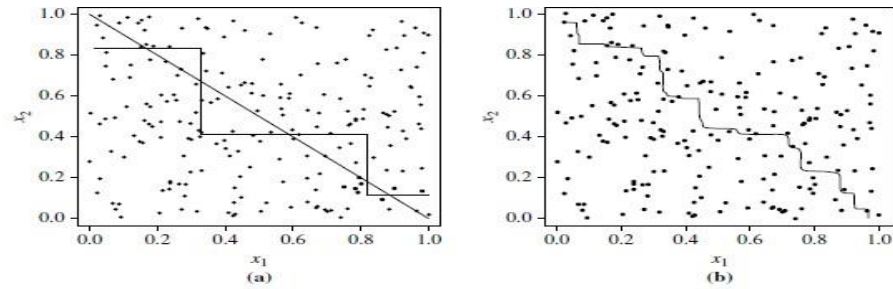


Figure 8.22 Decision boundary by (a) a single decision tree and (b) an ensemble of decision trees for a linearly separable problem (i.e., where the actual decision boundary is a straight line). The decision tree struggles with approximating a linear boundary. The decision boundary of the ensemble is closer to the true boundary. *Source:* From Seni and Elder [SE10]. © 2010 Morgan & Claypool Publishers; used with permission.

Bagging

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

Given a set, D , of d tuples, **bagging** works as follows. For iteration i ($i = 1, 2, \dots, k$), a training set, D_i , of d tuples is sampled with replacement from the original set of tuples, D . Note that the term *bagging* stands for *bootstrap aggregation*. Each training set is a bootstrap sample, as described in Section 8.5.4. Because sampling with replacement is used, some of the original tuples of D may not be included in D_i , whereas others may occur more than once. A classifier model, M_i , is learned for each training set, D_i . To classify an unknown tuple, X , each classifier, M_i , returns its class prediction, which counts as one vote. The bagged classifier, M^* , counts the votes and assigns the class with the most votes to X . Bagging can be applied to the prediction of continuous values by taking the average value of each prediction for a given test tuple.

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

Input:

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

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

Method:

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

To use the ensemble to classify a tuple, X :

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

Figure 8.23 Bagging.

It will not be considerably worse and is more robust to the effects of noisy data and overfitting. The increased accuracy occurs because the composite model reduces the variance of the individual classifiers.

Boosting and AdaBoost

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

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

AdaBoost (short for Adaptive Boosting) is a popular boosting algorithm. Suppose we want to boost the accuracy of a learning method. We are given D , a data set of d class-labeled tuples, $(X_1, y_1), (X_2, y_2), \dots, (X_d, y_d)$, where y_i is the class label of tuple X_i . Initially, AdaBoost assigns each training tuple an equal weight of $1/d$. Generating k classifiers for the ensemble requires k rounds through the rest of the algorithm. In round i , the tuples from D are sampled to form a training set, D_i , of size d . Sampling

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

If a tuple was incorrectly classified, its weight is increased. If a tuple was correctly classified, its weight is decreased. A tuple's weight reflects how difficult it is to classify—the higher the weight, the more often it has been misclassified. These weights will be used to generate the training samples for the classifier of the next round. The basic idea is that when we build a classifier, we want it to focus more on the misclassified tuples of the previous round. Some classifiers may be better at classifying some “difficult” tuples than others. In this way, we build a series of classifiers that complement each other.

Therefore, sometimes the resulting “boosted” model may be less accurate than a single model derived from the same data. Bagging is less susceptible to model overfitting. While both can significantly improve accuracy in comparison to a single model, boosting tends to achieve greater accuracy.

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

Input:

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

Output: A composite model.

Method:

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

To use the ensemble to classify tuple, X :

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

Figure 8.24 AdaBoost, a boosting algorithm.

Random Forests

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

Random forests can be built using bagging (Section 8.6.2) in tandem with random attribute selection. A training set, D , of d tuples is given. The general procedure to generate k decision trees for the ensemble is as follows. For each iteration, i ($i = 1, 2, \dots, k$), a training set, D_i , of d tuples is sampled with replacement from D . That is, each D_i is a bootstrap sample of D (Section 8.5.4), so that some tuples may occur more than once in D_i , while others may be excluded. Let F be the number of attributes to be used to determine the split at each node, where F is much smaller than the number of available attributes. To construct a decision tree classifier, M_i , randomly select, at each node, F attributes as candidates for the split at the node. The CART methodology is used to grow the trees. The trees are grown to maximum size and are not pruned. Random forests formed this way, with *random input selection*, are called Forest-RI.

Another form of random forest, called Forest-RC, uses *random linear combinations* of the input attributes. Instead of randomly selecting a subset of the attributes, it creates new attributes (or features) that are a linear combination of the existing attributes. That is, an attribute is generated by specifying L , the number of original attributes to be combined. At a given node, L attributes are randomly selected and added together with coefficients that are uniform random numbers on $[-1, 1]$. F linear combinations are generated, and a search is made over these for the best split. This form of random forest is useful when there are only a few attributes available, so as to reduce the correlation between individual classifiers.

Random forests are comparable in accuracy to AdaBoost, yet are more robust to errors and outliers. The generalization error for a forest converges as long as the number of trees in the forest is large. Thus, overfitting is not a problem. The accuracy of a random forest depends on the strength of the individual classifiers and a measure of the dependence between them. The ideal is to maintain the strength of individual classifiers without increasing their correlation. Random forests are insensitive to the number of attributes selected for consideration at each split.

Improving Classification Accuracy of Class-Imbalanced Data

Given two-class data, the data are class-imbalanced if the main class of interest (the positive class) is represented by only a few tuples, while the majority of tuples represent the negative class. For multiclass-imbalanced data, the data distribution of each class differs substantially where, again, the main class or classes of interest are rare. The class imbalance problem is closely related to cost-sensitive learning, wherein the costs of errors, per class, are not equal. In medical diagnosis, for example, it is much more costly to falsely diagnose a cancerous patient as healthy (a false negative) than to misdiagnose a healthy patient as having cancer (a false positive). A false negative error could lead to the loss of life and therefore is much more expensive than a false positive error. Other applications involving class-imbalanced data include fraud detection, the detection of oil spills from satellite radar images, and fault monitoring.

Traditional classification algorithms aim to minimize the number of errors made during classification. They assume that the costs of false positive and false negative errors are equal. By assuming a balanced distribution of classes and equal error costs, they are therefore not suitable for class-imbalanced data.

In this section, we look at general approaches for *improving* the classification accuracy of class-imbalanced data. These approaches include (1) oversampling, (2) undersampling, (3) threshold moving, and (4) ensemble techniques. The first three do not involve any changes to the construction of the classification model. That is, oversampling and undersampling change the distribution of tuples in the training set; threshold moving affects how the model makes decisions when classifying new data.

Both oversampling and undersampling change the training data distribution so that the rare (positive) class is well represented. **Oversampling** works by resampling the positive tuples so that the resulting training set contains an equal number of positive and negative tuples. **Undersampling**

works by decreasing the number of negative tuples. It randomly eliminates tuples from the majority (negative) class until there are an equal number of positive and negative tuples.

Example 8.12 Oversampling and undersampling. Suppose the original training set contains 100 positive and 1000 negative tuples. In oversampling, we replicate tuples of the rarer class to form a new training set containing 1000 positive tuples and 1000 negative tuples. In undersampling, we randomly eliminate negative tuples so that the new training set contains 100 positive tuples and 100 negative tuples.

Several variations to oversampling and undersampling exist. They may vary, for instance, in how tuples are added or eliminated. For example, the SMOTE algorithm uses oversampling where synthetic tuples are added, which are “close to” the given positive tuples in tuple space.

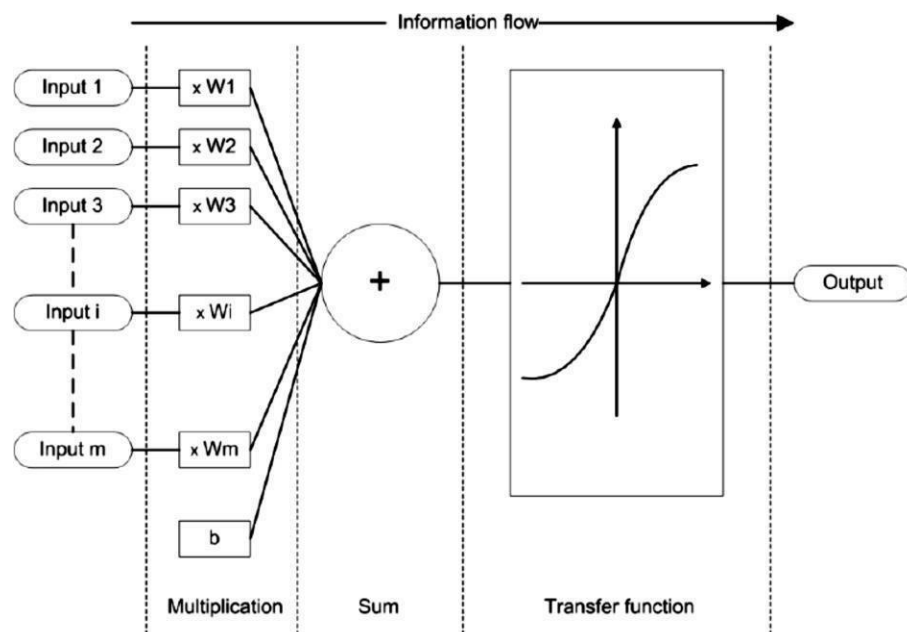
The **threshold-moving** approach to the class imbalance problem does not involve any sampling. It applies to classifiers that, given an input tuple, return a continuous output value. That is, for an input tuple, X , such a classifier returns as output a mapping, $f(X) \rightarrow [0,1]$. Rather than manipulating the training tuples, this method returns a classification decision based on the output values.

Ensemble methods have also been applied to the class imbalance problem. The individual classifiers making up the ensemble may include versions of the approaches described here such as oversampling and threshold moving.

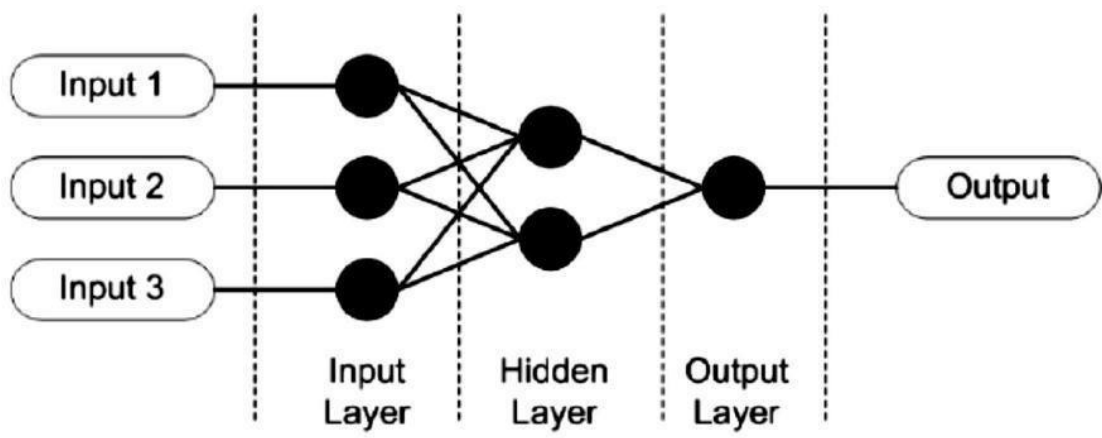
NEURAL NETWORKS

WHAT IS ARTIFICIAL NEURAL NETWORK?

An Artificial Neural Network (ANN) is a mathematical model that tries to simulate the structure and functionalities of biological neural networks. Basic building block of every artificial neural network is artificial neuron, that is, a simple mathematical model (function). Such a model has three simple sets of rules: multiplication, summation and activation. At the entrance of artificial neuron the inputs are weighted what means that every input value is multiplied with individual weight. In the middle section of artificial neuron is sum function that sums all weighted inputs and bias. At the exit of artificial neuron the sum of previously weighted inputs and bias is passing through activation function that is also called



transfer function.



● Single neuron

BIOLOGICAL NEURON STRUCTURE AND FUNCTIONS.

A neuron, or nerve cell, is an electrically excitable cell that communicates with other cells via specialized connections called synapses. It is the main component of nervous tissue. Neurons are typically classified into three types based on their function. Sensory neurons respond to stimuli such as touch, sound, or light that affect the cells of the sensory organs, and they send signals to the spinal cord or brain. Motor neurons receive signals from the brain and spinal cord to control everything from muscle contractions to glandular output. Interneurons connect neurons to other neurons within the same region of the brain or spinal cord. A group of connected neurons is called a neural circuit.

A typical neuron consists of a cell body (soma), dendrites, and a single axon. The soma is usually compact. The axon and dendrites are filaments that extrude from it. Dendrites typically branch profusely and extend a few hundred micrometers from the soma. The axon leaves the soma at a swelling called the axon hillock, and travels for as far as 1 meter in humans or more in other species. It branches but usually maintains a constant diameter. At the farthest tip of the axon's branches are axon terminals, where the neuron can transmit a signal across the synapse to another cell. Neurons may lack dendrites or have no axon. The term neurite is used to describe either a dendrite or an axon, particularly when the cell is undifferentiated.

The soma is the body of the neuron. As it contains the nucleus, most protein synthesis occurs here. The nucleus can range from 3 to 18 micrometers in diameter.

The dendrites of a neuron are cellular extensions with many branches. This overall shape and structure is referred to metaphorically as a dendritic tree. This is where the majority of input to the neuron occurs via the dendritic spine.

The axon is a finer, cable-like projection that can extend tens, hundreds, or even tens of thousands of times the diameter of the soma in length. The axon primarily carries nerve signals away from the soma, and carries some types of information back to it. Many neurons have only one axon, but this axon may—and usually will—undergo extensive branching, enabling communication with many target cells. The part of the axon where it emerges from the soma is called the axon hillock. Besides being an anatomical structure, the axon hillock also has the greatest density of voltage-dependent sodium channels. This makes it the most easily excited part of the neuron and the spike initiation zone for the axon. In electrophysiological terms, it has the most negative threshold potential.

While the axon and axon hillock are generally involved in information outflow, this region can also receive input from other neurons.

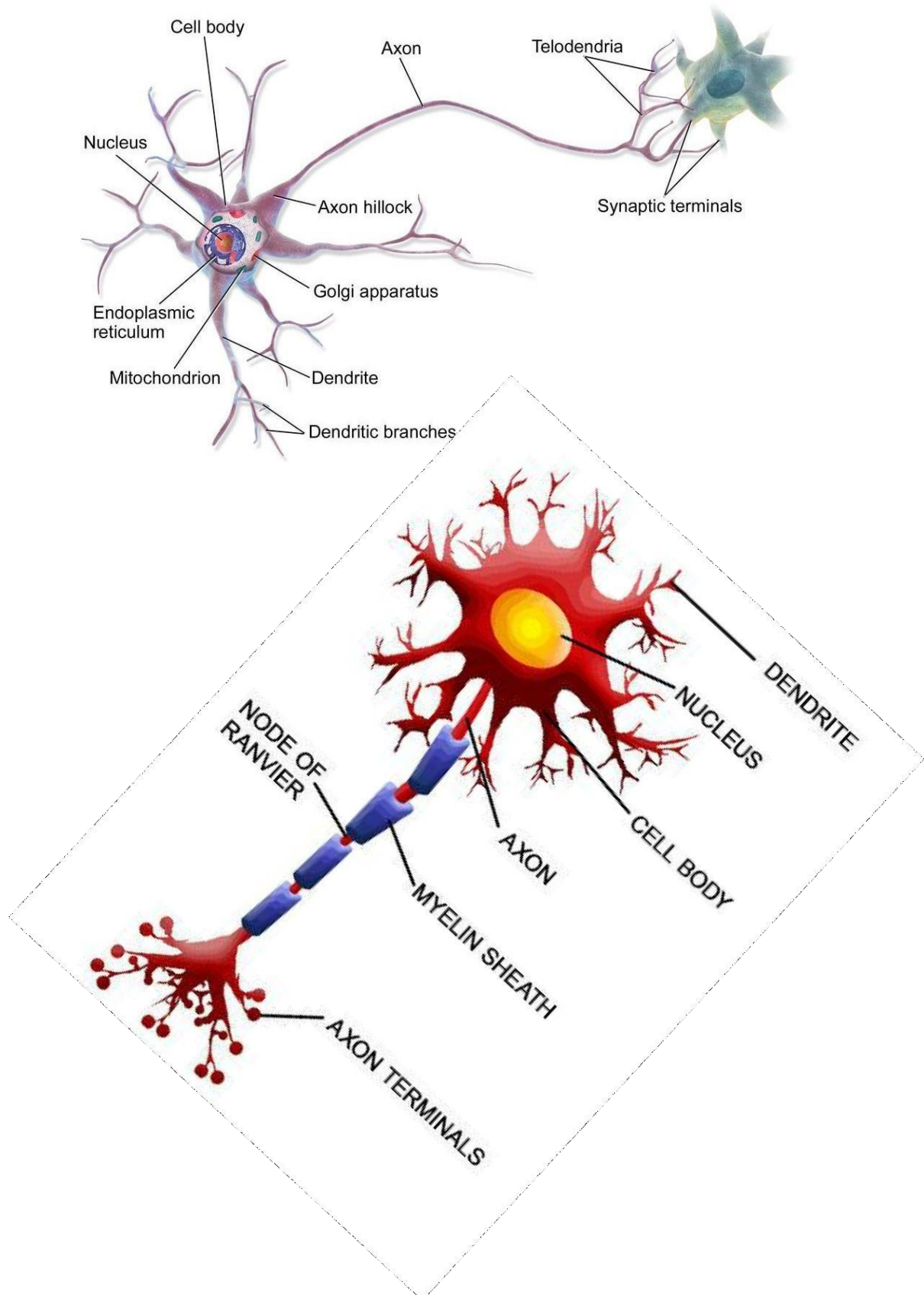
The axon terminal is found at the end of the axon farthest from the soma and contains synapses. Synaptic boutons are specialized structures where neurotransmitter chemicals are released to communicate with target neurons. In addition to synaptic boutons at the axon terminal, a neuron may have en passant boutons, which are located along the length of the axon.

Most neurons receive signals via the dendrites and soma and send out signals down the axon. At the majority of synapses, signals cross from the axon of one neuron to a dendrite of another. However, synapses can connect an axon to another axon or a dendrite to another dendrite. The signaling process is partly electrical and partly chemical. Neurons are electrically excitable, due to maintenance of voltage gradients across their membranes. If the voltage changes by a large amount over a short interval, the

neuron generates an all-or-nothing electrochemical pulse called an action potential. This potential travels

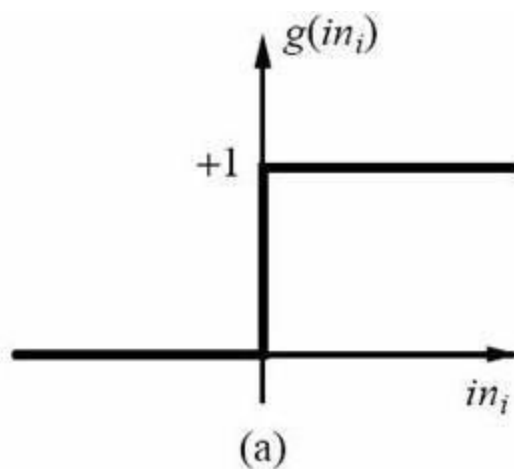
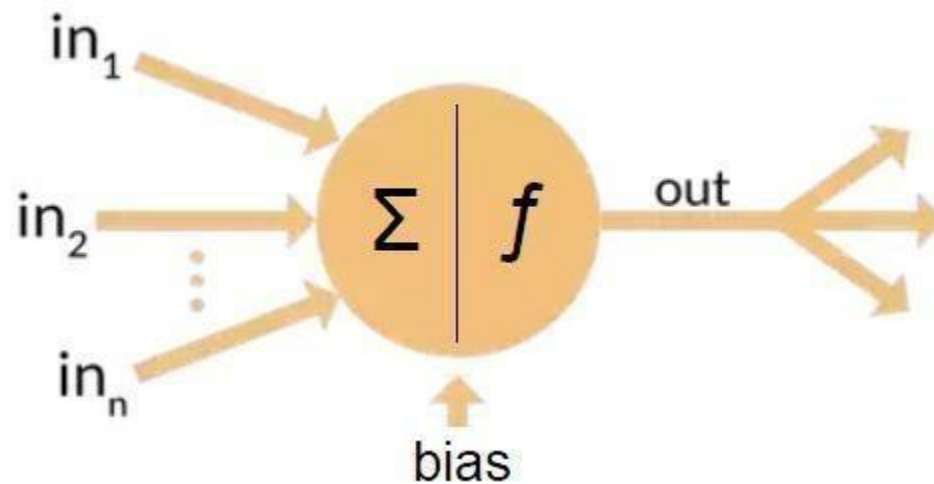
rapidly along the axon, and activates synaptic connections as it reaches them. Synaptic signals may be excitatory or inhibitory, increasing or reducing the net voltage that reaches the soma.

In most cases, neurons are generated by neural stem cells during brain development and childhood. Neurogenesis largely ceases during adulthood in most areas of the brain. However, strong evidence supports generation of substantial numbers of new neurons in the hippocampus and olfactory bulb.

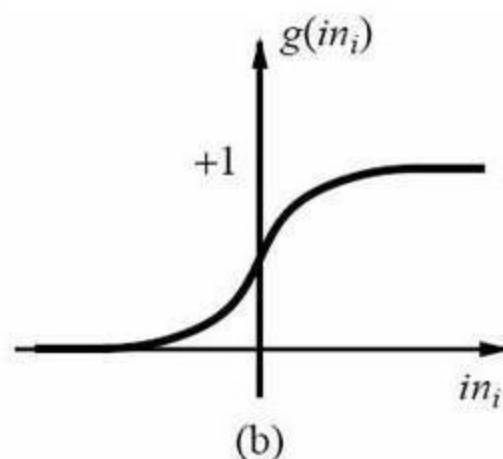


STRUCTURE AND FUNCTIONS OF ARTIFICIAL NEURON.

An artificial neuron is a mathematical function conceived as a model of biological neurons, a neural network. Artificial neurons are elementary units in an artificial neural network. The artificial neuron receives one or more inputs (representing excitatory postsynaptic potentials and inhibitory postsynaptic potentials at neural dendrites) and sums them to produce an output (or activation, representing a neuron's action potential which is transmitted along its axon). Usually each input is separately weighted, and the sum is passed through a non-linear function known as an activation function or transfer function. The transfer functions usually have a sigmoid shape, but they may also take the form of other non-linear functions, piecewise linear functions, or step functions. They are also often monotonically increasing, continuous, differentiable and bounded. The thresholding function has inspired building logic gates referred to as threshold logic; applicable to building logic circuits resembling brain processing. For example, new devices such as memristors have been extensively used to develop such logic in recent times.



step function



sigmoid function

STATE THE MAJOR DIFFERENCES BETWEEN BIOLOGICAL AND ARTIFICIAL NEURAL NETWORKS

- 1. Size:** Our brain contains about 86 billion neurons and more than a 100 synapses (connections). The number of “neurons” in artificial networks is much less than that.
- 2. Signal transport and processing:** The human brain works asynchronously, ANNs work synchronously.
- 3. Processing speed:** Single biological neurons are slow, while standard neurons in ANNs are fast.
- 4. Topology:** Biological neural networks have complicated topologies, while ANNs are often in a tree structure.
- 5. Speed:** certain biological neurons can fire around 200 times a second on average. Signals travel at different speeds depending on the type of the nerve impulse, ranging from 0.61 m/s up to 119 m/s. Signal travel speeds also vary from person to person depending on their sex, age, height, temperature, medical condition, lack of sleep etc. Information in artificial neurons is carried over by the continuous, floating point number values of synaptic weights. There are no refractory periods for artificial neural networks (periods while it is impossible to send another action potential, due to the sodium channels being lock shut) and artificial neurons do not experience “fatigue”: they are functions that can be calculated as many times and as fast as the computer architecture would allow.
- 6. Fault-tolerance:** biological neuron networks due to their topology are also fault-tolerant. Artificial neural networks are not modeled for fault tolerance or self regeneration (similarly to fatigue, these ideas are not applicable to matrix operations), though recovery is possible by saving the current state (weight values) of the model and continuing the training from that save state.
- 7. Power consumption:** the brain consumes about 20% of all the human body’s energy — despite it’s large cut, an adult brain operates on about 20 watts (barely enough to dimly light a bulb) being extremely efficient. Taking into account how humans can still operate for a while, when only given some c-vitamin rich lemon juice and beef tallow, this is quite remarkable. For benchmark: a single Nvidia GeForce Titan X GPU runs on 250 watts alone, and requires a power supply. Our machines are way less efficient than biological systems. Computers also generate a lot of heat when used, with consumer GPUs operating safely between 50–80°Celsius instead of 36.5–37.5 °C.
- 8. Learning:** we still do not understand how brains learn, or how redundant connections store and recall information. By learning, we are building on information that is already stored in the brain. Our knowledge deepens by repetition and during sleep, and tasks that once required a focus can be executed automatically once mastered. Artificial neural networks in the other hand, have a predefined model, where no further neurons or connections can be added or removed. Only the weights of the connections (and biases representing thresholds) can change during training. The networks start with random weight values and will slowly try to reach a point where further changes in the weights would no longer improve performance. Biological networks usually don't stop / start learning. ANNs have different fitting (train) and prediction (evaluate) phases.
- 9. Field of application:** ANNs are specialized. They can perform one task. They might be perfect at playing chess, but they fail at playing go (or vice versa). Biological neural networks can learn completely new tasks.
- 10. Training algorithm:** ANNs use Gradient Descent for learning. Human brains use something different (but we don't know what).

BRIEFLY EXPLAIN THE BASIC BUILDING BLOCKS OF ARTIFICIAL NEURAL NETWORKS.

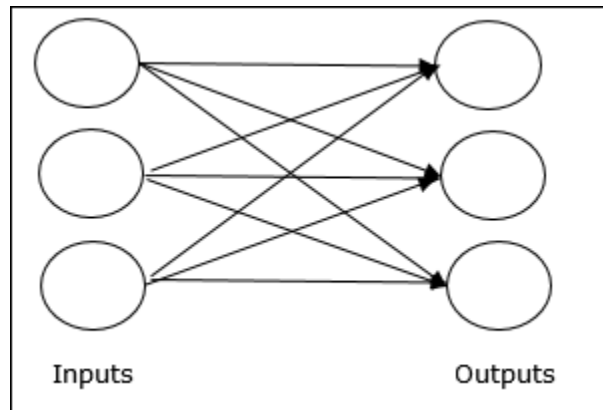
Processing of ANN depends upon the following three building blocks:

1. Network Topology
2. Adjustments of Weights or Learning
3. Activation Functions

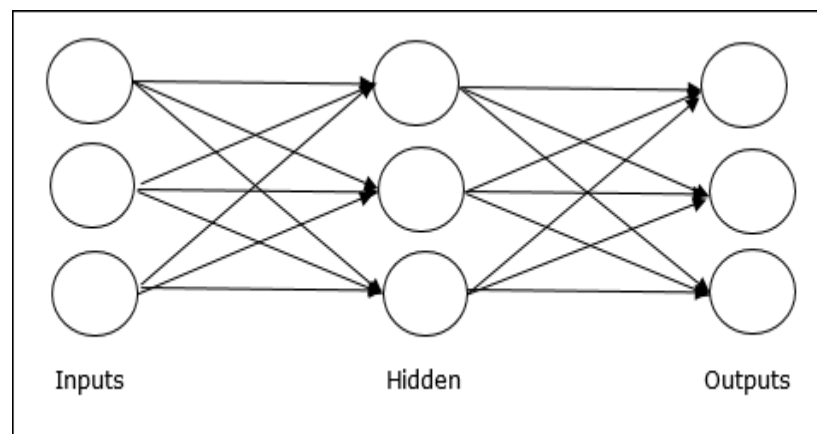
1. Network Topology: A network topology is the arrangement of a network along with its nodes and connecting lines. According to the topology, ANN can be classified as the following kinds:

A. Feed forward Network: It is a non-recurrent network having processing units/nodes in layers and all the nodes in a layer are connected with the nodes of the previous layers. The connection has different weights upon them. There is no feedback loop means the signal can only flow in one direction, from input to output. It may be divided into the following two types:

- **Single layer feed forward network:** The concept is of feed forward ANN having only one weighted layer. In other words, we can say the input layer is fully connected to the output layer.
- **Multilayer feed forward network:** The concept is of feed forward ANN having more than one

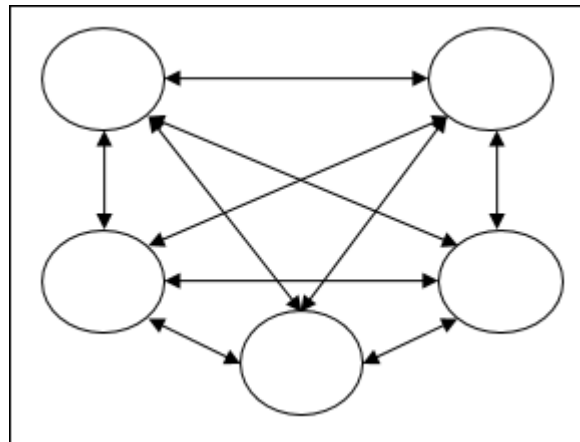


weighted layer. As this network has one or more layers between the input and the output layer, it is called hidden layers.

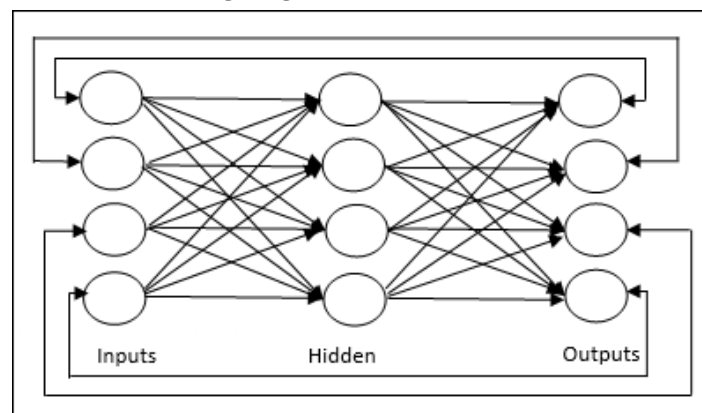


B. Feedback Network: As the name suggests, a feedback network has feedback paths, which means the signal can flow in both directions using loops. This makes it a non-linear dynamic system, which changes continuously until it reaches a state of equilibrium. It may be divided into the following types:

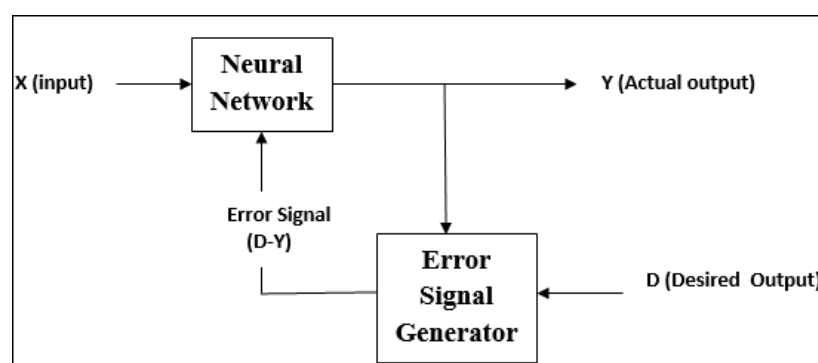
- **Recurrent networks:** They are feedback networks with closed loops. Following are the two types of recurrent networks.
- **Fully recurrent network:** It is the simplest neural network architecture because all nodes are connected to all other nodes and each node works as both input and output.



- **Jordan network** – It is a closed loop network in which the output will go to the input again as feedback as shown in the following diagram.

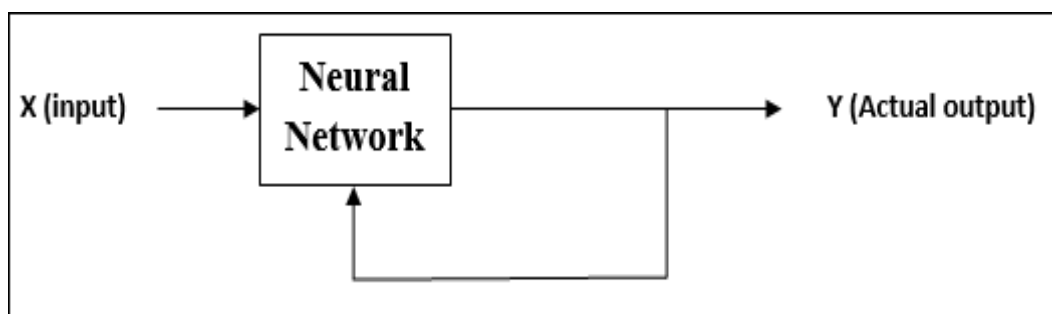


2. Adjustments of Weights or Learning: Learning, in artificial neural network, is the method of modifying the weights of connections between the neurons of a specified network. Learning in ANN can be classified into three categories namely supervised learning, unsupervised learning, and reinforcement learning.

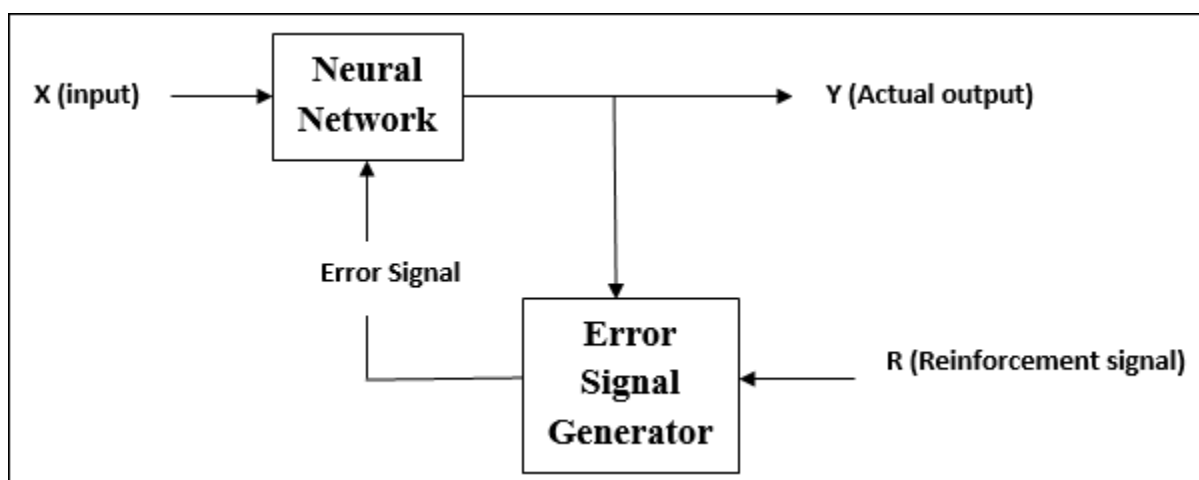


Supervised Learning: As the name suggests, this type of learning is done under the supervision of a teacher. This learning process is dependent. During the training of ANN under supervised learning, the input vector is presented to the network, which will give an output vector. This output vector is compared with the desired output vector. An error signal is generated, if there is a difference between the actual output and the desired output vector. On the basis of this error signal, the weights are adjusted until the actual output is matched with the desired output.

Unsupervised Learning: As the name suggests, this type of learning is done without the supervision of a teacher. This learning process is independent. During the training of ANN under unsupervised learning, the input vectors of similar type are combined to form clusters. When a new input pattern is applied, then the neural network gives an output response indicating the class to which the input pattern belongs. There is no feedback from the environment as to what should be the desired output and if it is correct or incorrect. Hence, in this type of learning, the network itself must discover the patterns and features from the input data, and the relation for the input data over the output.



Reinforcement Learning: As the name suggests, this type of learning is used to reinforce or strengthen the network over some critic information. This learning process is similar to supervised learning, however we might have very less information. During the training of network under reinforcement learning, the network receives some feedback from the environment. This makes it somewhat similar to supervised learning. However, the feedback obtained here is evaluative not instructive, which means there is no teacher as in supervised learning. After receiving the feedback, the network performs adjustments of the weights to get better critic information in future.



- 3. Activation Functions:** An activation function is a mathematical equation that determines the output of each element (perceptron or neuron) in the neural network. It takes in the input from each neuron and transforms it into an output, usually between one and zero or between -1 and one. It may be defined as the extra force or effort applied over the input to obtain an exact output. In ANN, we can also apply activation functions over the input to get the exact output. Followings are some activation functions of interest:

i) **Linear Activation Function:** It is also called the identity function as it performs no input editing. It can be defined as: $F(x) = x$

ii) **Sigmoid Activation Function:** It is of two type as follows –

- **Binary sigmoidal function:** This activation function performs input editing between 0 and 1. It is positive in nature. It is always bounded, which means its output cannot be less than 0 and more than 1. It is also strictly increasing in nature, which means more the input higher would be the output. It can be defined as

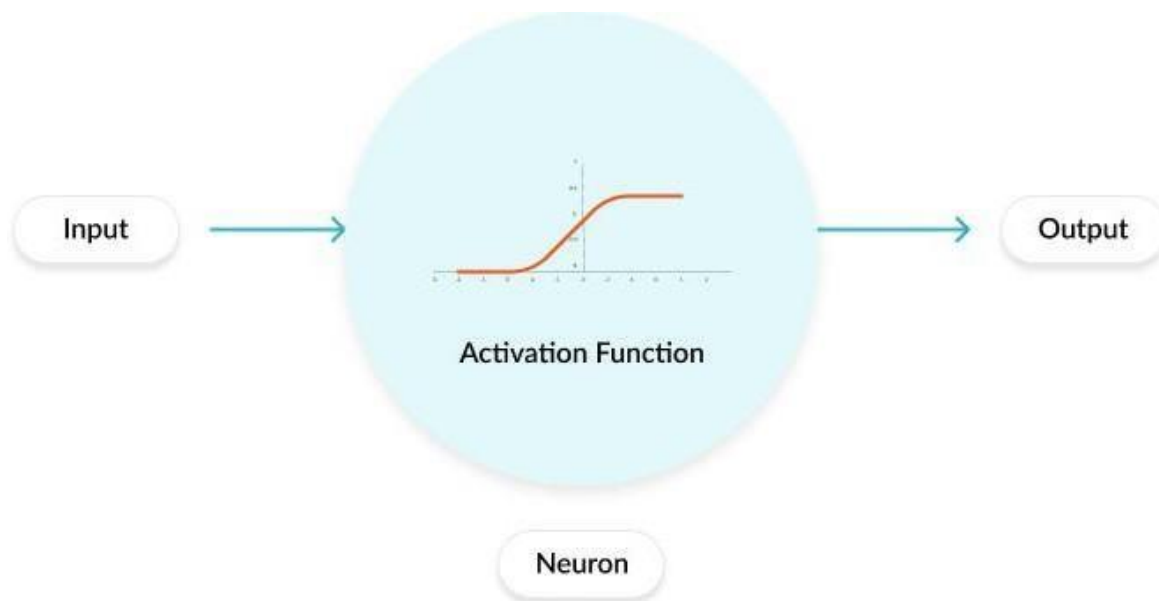
$$F(x) = \text{sigm}(x) = \frac{1}{1 + \exp(-x)}$$

- **Bipolar sigmoidal function:** This activation function performs input editing between -1 and 1. It can be positive or negative in nature. It is always bounded, which means its output cannot be less than -1 and more than 1. It is also strictly increasing in nature like sigmoid function. It can be defined as

$$F(x) = \text{sigm}(x) = \frac{1}{1 + \exp(-x)} - 0.5 = \frac{1 - \exp(-x)}{1 + \exp(-x)}$$

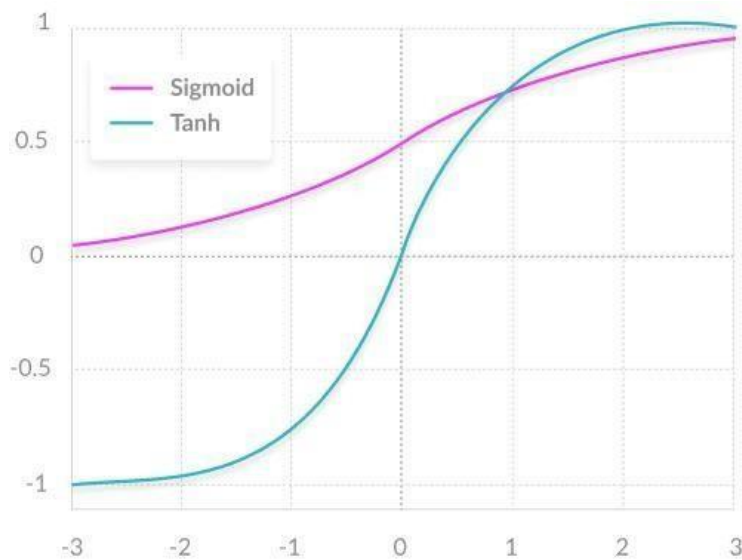
WHAT IS A NEURAL NETWORK ACTIVATION FUNCTION?

In a neural network, inputs, which are typically real values, are fed into the neurons in the network. Each neuron has a weight, and the inputs are multiplied by the weight and fed into the activation function. Each neuron's output is the input of the neurons in the next layer of the network, and so the inputs cascade through multiple activation functions until eventually, the output layer generates a prediction. Neural networks rely on nonlinear activation functions—the derivative of the activation function helps the network learn through the backpropagation process.



SOME COMMON ACTIVATION FUNCTIONS INCLUDE THE FOLLOWING:

1. **The sigmoid function** has a smooth gradient and outputs values between zero and one. For very high or low values of the input parameters, the network can be very slow to reach a prediction, called the *vanishing gradient* problem.
2. **The TanH function** is zero-centered making it easier to model inputs that are strongly negative or strongly positive or neutral.
3. **The ReLu function** is highly computationally efficient but is not able to process inputs that approach zero or negative.
4. **The Leaky ReLu function** has a small positive slope in its negative area, enabling it to process zero or negative values.
5. **The Parametric ReLu function** allows the negative slope to be learned, performing backpropagation to learn the most effective slope for zero and negative input values.
6. **Softmax** is a special activation function use for output neurons. It normalizes outputs for each class between 0 and 1, and returns the probability that the input belongs to a specific class.
7. **Swish** is a new activation function discovered by Google researchers. It performs better than ReLu with a similar level of computational efficiency.



Two common neural network activation functions - Sigmoid and Tanh

APPLICATIONS OF ANN

1. **Data Mining:** Discovery of meaningful patterns (knowledge) from large volumes of data.
2. **Expert Systems:** A computer program for decision making that simulates thought process of a human expert.
3. **Fuzzy Logic:** Theory of approximate reasoning.
4. **Artificial Life:** Evolutionary Computation, Swarm Intelligence.
5. **Artificial Immune System:** A computer program based on the biological immune system.
6. **Medical:** At the moment, the research is mostly on modelling parts of the human body and recognizing diseases from various scans (e.g. cardiograms, CAT scans, ultrasonic scans, etc.). Neural networks are ideal in recognizing diseases using scans since there is no need to provide a specific algorithm on how to identify the disease. Neural networks learn by example so the details of how to recognize the disease are not needed. What is needed is a set of examples that are representative of all the variations of the disease. The quantity of examples is not as important as the 'quality'. The examples need to be selected very carefully if the system is to perform reliably and efficiently.

7. Computer Science: Researchers in quest of artificial intelligence have created spin offs like dynamic programming, object oriented programming, symbolic programming, intelligent storage management systems and many more such tools. The primary goal of creating an artificial intelligence still remains a distant dream but people are getting an idea of the ultimate path, which could lead to it.
8. Aviation: Airlines use expert systems in planes to monitor atmospheric conditions and system status. The plane can be put on autopilot once a course is set for the destination.
9. Weather Forecast: Neural networks are used for predicting weather conditions. Previous data is fed to a neural network, which learns the pattern and uses that knowledge to predict weather patterns.
10. Neural Networks in business: Business is a diverted field with several general areas of specialization such as accounting or financial analysis. Almost any neural network application would fit into one business area or financial analysis.
11. There is some potential for using neural networks for business purposes, including resource allocation and scheduling.
12. There is also a strong potential for using neural networks for database mining, which is, searching for patterns implicit within the explicitly stored information in databases. Most of the funded work in this area is classified as proprietary. Thus, it is not possible to report on the full extent of the work going on. Most work is applying neural networks, such as the Hopfield-Tank network for optimization and scheduling.
13. Marketing: There is a marketing application which has been integrated with a neural network system. The Airline Marketing Tactician (a trademark abbreviated as AMT) is a computer system made of various intelligent technologies including expert systems. A feed forward neural network is integrated with the AMT and was trained using back-propagation to assist the marketing control of airline seat allocations. The adaptive neural approach was amenable to rule expression. Additionally, the application's environment changed rapidly and constantly, which required a continuously adaptive solution.
14. Credit Evaluation: The HNC company, founded by Robert Hecht-Nielsen, has developed several neural network applications. One of them is the Credit Scoring system which increases the profitability of the existing model up to 27%. The HNC neural systems were also applied to mortgage screening. A neural network automated mortgage insurance under writing system was developed by the Nestor Company. This system was trained with 5048 applications of which 2597 were certified. The data related to property and borrower qualifications. In a conservative model the system agreed on the under writers on 97% of the cases. In the liberal model the system agreed 84% of the cases. This is system run on an Apollo DN3000 and used 250K memory while processing a case file in approximately 1 sec.

ADVANTAGES OF ANN

1. Adaptive learning: An ability to learn how to do tasks based on the data given for training or initial experience.
2. Self-Organisation: An ANN can create its own organisation or representation of the information it receives during learning time.
3. Real Time Operation: ANN computations may be carried out in parallel, and special hardware devices are being designed and manufactured which take advantage of this capability.
4. Pattern recognition: is a powerful technique for harnessing the information in the data and generalizing about it. Neural nets learn to recognize the patterns which exist in the data set.
5. The system is developed through learning rather than programming.. Neural nets teach themselves the patterns in the data freeing the analyst for more interesting work.

6. Neural networks are flexible in a changing environment. Although neural networks may take some time to learn a sudden drastic change they are excellent at adapting to constantly changing information.
7. Neural networks can build informative models whenever conventional approaches fail. Because neural networks can handle very complex interactions they can easily model data which is too difficult to model with traditional approaches such as inferential statistics or programming logic.
8. Performance of neural networks is at least as good as classical statistical modelling, and better on most problems. The neural networks build models that are more reflective of the structure of the data in significantly less time.

LIMITATIONS OF ANN

In this technological era everything has Merits and some Demerits in others words there is a Limitation with every system which makes this ANN technology weak in some points. The various Limitations of ANN are:-

- 1) ANN is not a daily life general purpose problem solver.
- 2) There is no structured methodology available in ANN.
- 3) There is no single standardized paradigm for ANN development.
- 4) The Output Quality of an ANN may be unpredictable.
- 5) Many ANN Systems does not describe how they solve problems.
- 6) Black box Nature
- 7) Greater computational burden.
- 8) Proneness to over fitting.
- 9) Empirical nature of model development.

ARTIFICIAL NEURAL NETWORK CONCEPTS/TERMINOLOGY

Here is a glossary of basic terms you should be familiar with before learning the details of neural networks.

Inputs: Source data fed into the neural network, with the goal of making a decision or prediction about the data. Inputs to a neural network are typically a set of real values; each value is fed into one of the neurons in the input layer.

Training Set: A set of inputs for which the correct outputs are known, used to train the neural network.

Outputs : Neural networks generate their predictions in the form of a set of real values or boolean decisions. Each output value is generated by one of the neurons in the output layer.

Neuron/perceptron: The basic unit of the neural network. Accepts an input and generates a prediction.

Each neuron accepts part of the input and passes it through the activation function. Common activation functions are sigmoid, TanH and ReLu. Activation functions help generate output values within an acceptable range, and their non-linear form is crucial for training the network.

Weight Space: Each neuron is given a numeric weight. The weights, together with the activation function, define each neuron's output. Neural networks are trained by fine-tuning weights, to discover the optimal set of weights that generates the most accurate prediction.

Forward Pass: The forward pass takes the inputs, passes them through the network and allows each neuron to react to a fraction of the input. Neurons generate their outputs and pass them on to the next layer, until eventually the network generates an output.

Error Function: Defines how far the actual output of the current model is from the correct output. When training the model, the objective is to minimize the error function and bring output as close as

possible to the correct value.

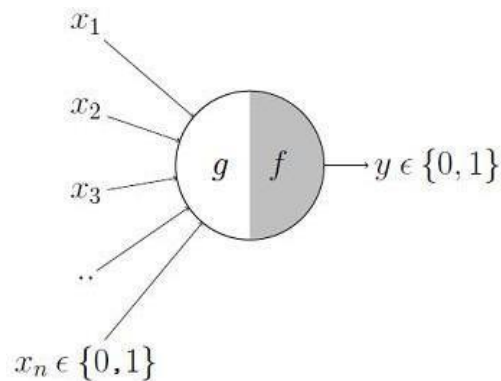
Backpropagation: In order to discover the optimal weights for the neurons, we perform a backward pass, moving back from the network's prediction to the neurons that generated that prediction. This is called backpropagation. Backpropagation tracks the derivatives of the activation functions in each successive neuron, to find weights that bring the loss function to a minimum, which will generate the best prediction. This is a mathematical process called *gradient descent*.

Bias and Variance: When training neural networks, like in other machine learning techniques, we try to balance between bias and variance. Bias measures how well the model fits the training set—able to correctly predict the known outputs of the training examples. Variance measures how well the model works with unknown inputs that were not available during training. Another meaning of bias is a “bias neuron” which is used in every layer of the neural network. The bias neuron holds the number 1, and makes it possible to move the activation function up, down, left and right on the number graph.

Hyperparameters: A hyper parameter is a setting that affects the structure or operation of the neural network. In real deep learning projects, tuning hyper parameters is the primary way to build a network that provides accurate predictions for a certain problem. Common hyper parameters include the number of hidden layers, the activation function, and how many times (epochs) training should be repeated.

McCULLOGH-PITTS MODEL

In 1943 two electrical engineers, Warren McCulloch and Walter Pitts, published the first paper describing what we would call a neural network.



It may be divided into 2 parts. The first part, g takes an input, performs an aggregation and based on the aggregated value the second part, f makes a decision. Let us suppose that I want to predict my own decision, whether to watch a random football game or not on TV. The inputs are all boolean i.e., $\{0,1\}$ and my output variable is also boolean $\{0: \text{Will watch it}, 1: \text{Won't watch it}\}$.

So, x_1 could be ‘is Indian Premier League On’ (I like Premier League more)

x_2 could be ‘is it a knockout game (I tend to care less about the league level matches)

x_3 could be ‘is Not Home’ (Can't watch it when I'm in College. Can I?)

x_4 could be ‘is my favorite team playing’ and so on.

These inputs can either be excitatory or inhibitory. Inhibitory inputs are those that have maximum effect on the decision making irrespective of other inputs i.e., if x_3 is 1 (not home) then my output will always be 0 i.e., the neuron will never fire, so x_3 is an inhibitory input. Excitatory inputs are NOT the ones that will make the neuron fire on their own but they might fire it when combined together. Formally, this is what is going on:

$$g(x_1, x_2, x_3, \dots, x_n) = g(\mathbf{x}) = \sum_{i=1}^n x_i$$

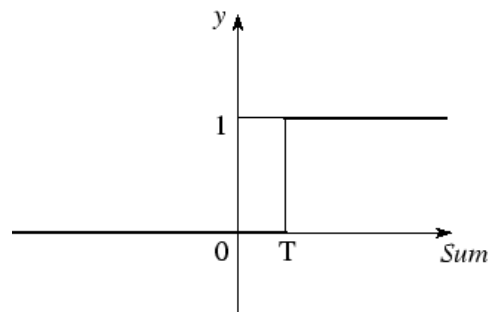
$$y = f(g(\mathbf{x})) = \begin{cases} 1 & \text{if } g(\mathbf{x}) \geq \theta \\ 0 & \text{if } g(\mathbf{x}) < \theta \end{cases}$$

We can see that $g(\mathbf{x})$ is just doing a sum of the inputs — a simple aggregation. And θ here is called thresholding parameter. For example, if I always watch the game when the sum turns out to be 2 or more, the θ is 2 here. This is called the Thresholding Logic.

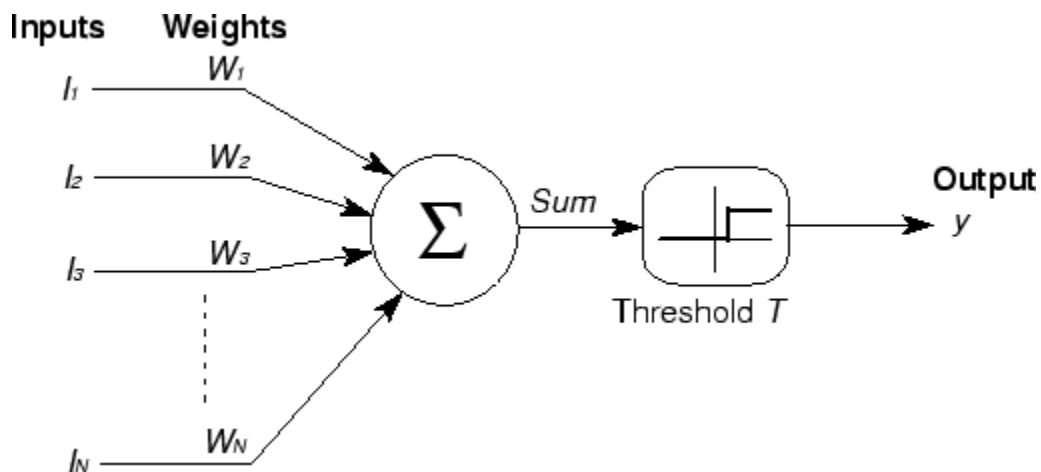
The McCulloch-Pitts neural model is also known as linear threshold gate. It is a neuron of a set of inputs $I_1, I_2, I_3, \dots, I_m$ and one output 'y'. The linear threshold gate simply classifies the set of inputs into two different classes. Thus the output y is binary. Such a function can be described mathematically using these equations:

$$Sum = \sum_{i=1}^N I_i W_i, \quad y = f(Sum).$$

Where, $W_1, W_2, W_3, \dots, W_m$ are weight values normalized in the range of either $(0, 1)$ or $(-1, 1)$ and associated with each input line, Sum is the weighted sum, and T is a threshold constant. The function f is a linear step function at threshold T as shown in figure 2.3. The symbolic representation of the linear threshold gate is shown in figure below.



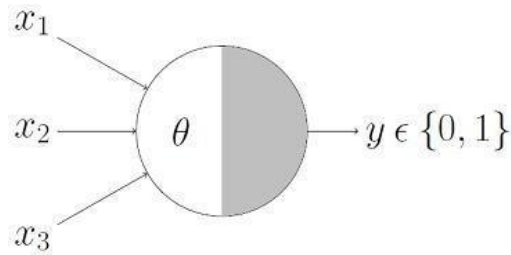
Linear Threshold Function



Symbolic Illustration of Linear Threshold Gate

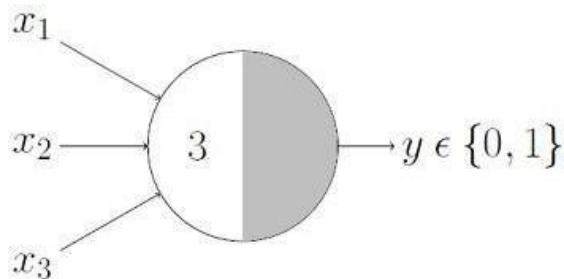
BOOLEAN FUNCTIONS USING MCCULLOGH-PITTS NEURON

In any Boolean function, all inputs are Boolean and the output is also Boolean. So essentially, the neuron is just trying to learn a Boolean function.

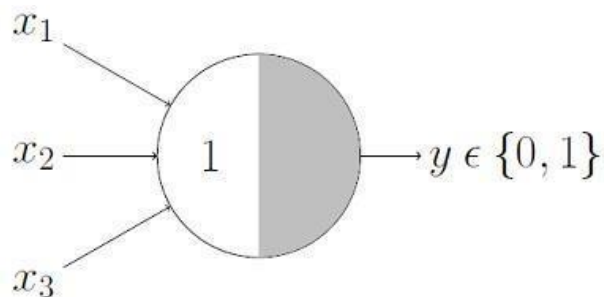


This representation just denotes that, for the boolean inputs x_1 , x_2 and x_3 if the $g(x)$ i.e., $\text{sum} \geq \theta$, the neuron will fire otherwise, it won't.

AND Function



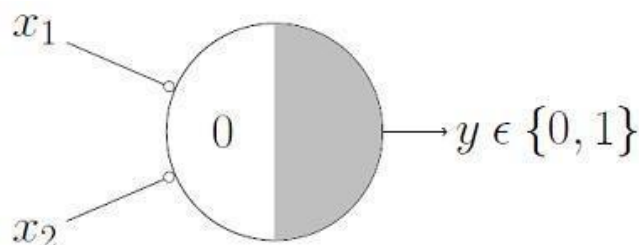
An AND function neuron would only fire when ALL the inputs are ON i.e., $g(x) \geq 3$ here.



OR Function

For an OR function neuron would fire if ANY of the inputs is ON i.e., $g(x) \geq 1$ here.

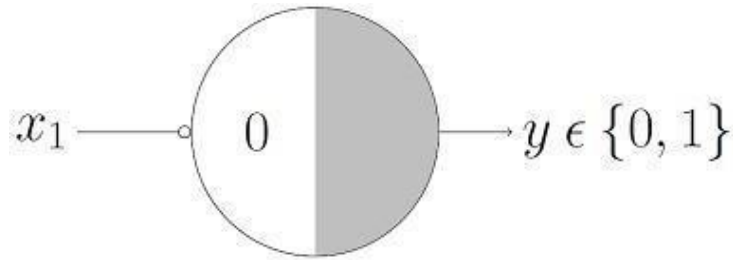
NOR Function



For a NOR neuron to fire, we want ALL the inputs to be 0 so the thresholding parameter should also be

0 and we take them all as inhibitory input.

NOT Function



For a NOT neuron, 1 outputs 0 and 0 outputs 1. So we take the input as an inhibitory input and set the thresholding parameter to 0.

We can summarize these rules with the McCulloch-Pitts output rule as:

The McCulloch-Pitts model of a neuron is simple yet has substantial computing potential. It also has a precise mathematical definition. However, this model is so simplistic that it only generates a binary output and also the weight and threshold values are fixed. The neural computing algorithm has diverse features for various applications. Thus, we need to obtain the neural model with more flexible computational features.

WHAT ARE THE LEARNING RULES IN ANN?

Learning rule is a method or a mathematical logic. It helps a Neural Network to learn from the existing conditions and improve its performance. Thus learning rules update the weights and bias levels of a network when a network simulates in a specific data environment. Applying learning rule is an iterative process. It helps a neural network to learn from the existing conditions and improve its performance.

The different learning rules in the Neural network are:

1. Hebbian learning rule – It identifies, how to modify the weights of nodes of a network.
2. Perceptron learning rule – Network starts its learning by assigning a random value to each weight.
3. Delta learning rule – Modification in synaptic weight of a node is equal to the multiplication of error and the input.
4. Correlation learning rule – The correlation rule is the supervised learning.
5. Outstar learning rule – We can use it when it assumes that nodes or neurons in a network arranged in a layer.

- 1. Hebbian Learning Rule:** The Hebbian rule was the first learning rule. In 1949 Donald Hebb developed it as learning algorithm of the unsupervised neural network. We can use it to identify how to improve the weights of nodes of a network. The Hebb learning rule assumes that – If two neighbor neurons activated and deactivated at the same time, then the weight connecting these neurons should increase. At the start, values of all weights are set to zero. This learning rule can be used for both soft- and hard-activation functions. Since desired responses of neurons are not used in the learning procedure, this is the unsupervised learning rule. The absolute values of the weights are usually proportional to the learning time, which is undesired.

$$W_{ij} = x_i * x_j$$

Mathematical Formula of Hebb Learning Rule.

- 2. Perceptron Learning Rule:** Each connection in a neural network has an associated weight, which changes in the course of learning. According to it, an example of supervised learning, the network starts its learning by assigning a random value to each weight. Calculate the output value on the basis of a set of records for which we can know the expected output value. This is the learning sample that indicates the entire definition. As a result, it is called a learning sample. The network then compares the calculated output value with the expected value. Next calculates an error function E , which can be the sum of squares of the errors occurring for each individual in the learning sample which can be computed as:

$$\sum \sum (E_{ij} - O_{ij})^2$$

Mathematical Formula of Perceptron Learning Rule

Perform the first summation on the individuals of the learning set, and perform the second summation on the output units. E_{ij} and O_{ij} are the expected and obtained values of the j^{th} unit for the i^{th} individual. The network then adjusts the weights of the different units, checking each time to see if the error function has increased or decreased. As in a conventional regression, this is a matter of solving a problem of least squares. Since assigning the weights of nodes according to users, it is an example of supervised learning.

- 3. Delta Learning Rule:** Developed by Widrow and Hoff, the delta rule, is one of the most common learning rules. It depends on supervised learning. This rule states that the modification in synaptic weight of a node is equal to the multiplication of error and the input. In Mathematical form the delta rule is as follows:

$$\Delta w = \eta (t - y) x_i$$

Mathematical Formula of Delta Learning Rule

For a given input vector, compare the output vector is the correct answer. If the difference is zero, no learning takes place; otherwise, adjusts its weights to reduce this difference. The change in weight from u_i to u_j is: $\Delta w_{ij} = r * a_i * e_j$. where r is the learning rate, a_i represents the activation of u_i and e_j is the difference between the expected output and the actual output of u_j . If the set of input patterns form an independent set then learn arbitrary associations using the delta rule.

It has seen that for networks with linear activation functions and with no hidden units. The error squared vs. the weight graph is a paraboloid in n -space. Since the proportionality constant is negative, the graph of such a function is concave upward and has the least value. The vertex of this paraboloid represents the point where it reduces the error. The weight vector corresponding to this point is then the ideal weight vector. We can use the delta learning rule with both single output unit and several output units. While applying the delta rule assume that the error can be directly measured. The aim of applying the delta rule is to reduce the difference between the actual and expected output that is the error.

- 4. Correlation Learning Rule:** The correlation learning rule based on a similar principle as the Hebbian learning rule. It assumes that weights between responding neurons should be more positive, and weights between neurons with opposite reaction should be more negative. Contrary to the Hebbian rule, the correlation rule is the supervised learning, instead of an actual. The response, o_j , the desired response, d_j , uses for the weight-change calculation. In Mathematical form the correlation learning rule is as follows:

$$\Delta w_{ij} = \eta x_i d_j$$

Mathematical Formula of Correlation Learning Rule

Where d_j is the desired value of output signal. This training algorithm usually starts with the initialization of weights to zero. Since assigning the desired weight by users, the correlation learning rule is an example of supervised learning.

5. Out Star Learning Rule: We use the Out Star Learning Rule when we assume that nodes or neurons in a network arranged in a layer. Here the weights connected to a certain node should be equal to the desired outputs for the neurons connected through those weights. The out star rule produces the desired response t for the layer of n nodes. Apply this type of learning for all nodes in a particular layer. Update the weights for nodes are as in Kohonen neural networks. In Mathematical form, express the out star learning as follows:

$$w_{jk} = \begin{cases} \eta(y_k - w_{jk}) & \text{if node } j \text{ wins the competition} \\ 0 & \text{if node } j \text{ loses the competition} \end{cases}$$

Mathematical Formula of Out Star Learning Rule

This is a supervised training procedure because desired outputs must be known.

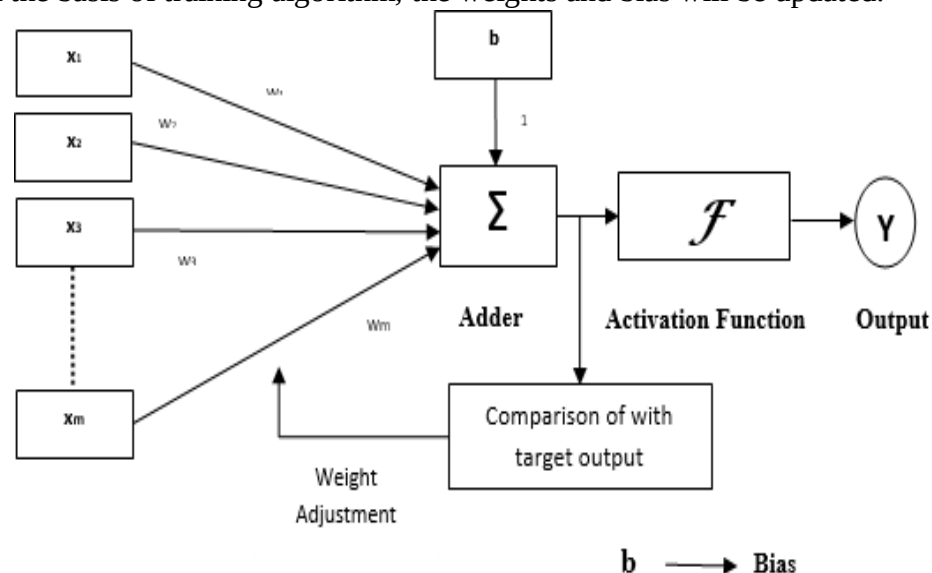
BRIEFLY EXPLAIN THE ADALINE MODEL OF ANN.

ADALINE (Adaptive Linear Neuron or later Adaptive Linear Element) is an early single-layer artificial neural network and the name of the physical device that implemented this network. The network uses memistors. It was developed by Professor Bernard Widrow and his graduate student Ted Hoff at Stanford University in 1960. It is based on the McCulloch–Pitts neuron. It consists of a weight, a bias and a summation function. The difference between Adaline and the standard (McCulloch–Pitts) perceptron is that in the learning phase, the weights are adjusted according to the weighted sum of the inputs (the net). In the standard perceptron, the net is passed to the activation (transfer) function and the function's output is used for adjusting the weights. Some important points about Adaline are as follows:

- It uses bipolar activation function.
- It uses delta rule for training to minimize the Mean-Squared Error (MSE) between the actual output and the desired/target output.
- The weights and the bias are adjustable.

Architecture of ADALINE network: The basic structure of Adaline is similar to perceptron having an extra feedback loop with the help of which the actual output is compared with the desired/target output. After comparison on the basis of training algorithm, the weights and bias will be updated.

Architecture of ADALINE: The basic structure of Adaline is similar to perceptron having an extra feedback loop with the help of which the actual output is compared with the desired/target output. After comparison on the basis of training algorithm, the weights and bias will be updated.



Training Algorithm of ADALINE:

Step 1 – Initialize the following to start the training:

- Weights
- Bias
- Learning rate α

For easy calculation and simplicity, weights and bias must be set equal to 0 and the learning rate must be set equal to 1.

Step 2 – Continue step 3-8 when the stopping condition is not

true. Step 3 – Continue step 4-6 for every bipolar training pair s :

t .

Step 4 – Activate each input unit as follows:

$$X_i = S_i \text{ (i=1 to n)}$$

Step 5 – Obtain the net input with the following relation:

$$y_{in} = b + \sum_i^n x_i w_i$$

Here 'b' is bias and 'n' is the total number of input neurons.

Step 6 – Apply the following activation function to obtain the final output:

$$f(y_{in}) = \begin{cases} 1, & \text{if } y_{in} \geq 0 \\ -1, & \text{if } y_{in} < 0 \end{cases}$$

Step 7 – Adjust the weight and bias as follows :

$$\begin{aligned} \text{Case 1 – if } y \neq t \text{ then, } w_i(\text{new}) &= w_i(\text{old}) + \alpha(t - y_{in})x_i \\ \mathbf{b(\text{new})} &= \mathbf{b(\text{old}) + \alpha(t - y_{in})} \end{aligned}$$

$$\text{Case 2 – if } y = t \text{ then, } w_i(\text{new}) = w_i(\text{old})$$

$$\mathbf{b(\text{new}) = b(\text{old})}$$

Here 'y' is the actual output and 't' is the desired/target output. $(t - y_{in})$ is the computed error.

Step 8 – Test for the stopping condition, which will happen when there is no change in weight or the highest weight change occurred during training is smaller than the specified tolerance.

EXPLAIN MULTIPLE ADAPTIVE LINEAR NEURONS (MADALINE).

Madaline which stands for Multiple Adaptive Linear Neuron, is a network which consists of many Adalines in parallel. It will have a single output unit. Three different training algorithms for MADALINE networks called Rule I, Rule II and Rule III have been suggested, which cannot be learned using backpropagation. The first of these dates back to 1962 and cannot adapt the weights of the hidden-output connection.[10] The second training algorithm improved on Rule I and was described in 1988.[8] The third "Rule" applied to a modified network with sigmoid activations instead of signum; it was later found to be equivalent to backpropagation. The Rule II training algorithm is based on a principle called "minimal disturbance". It proceeds by looping over training examples, then for each example, it:

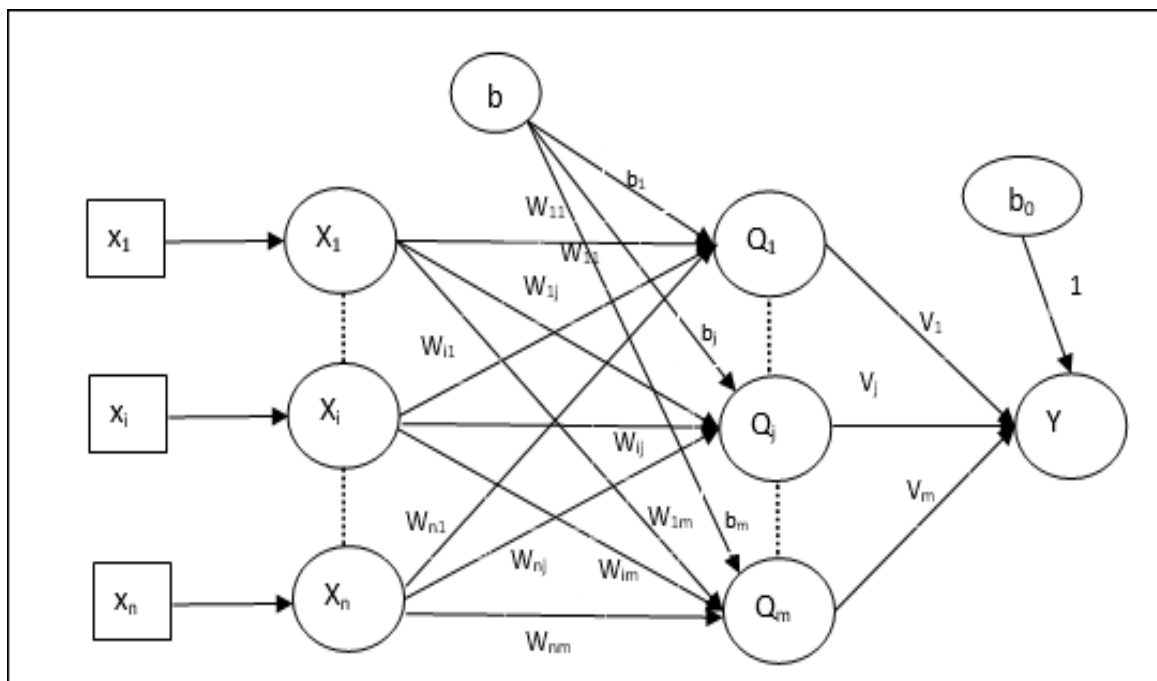
- finds the hidden layer unit (ADALINE classifier) with the lowest confidence in its prediction, tentatively flips the sign of the unit,
- accepts or rejects the change based on whether the network's error is reduced,
- stops when the error is zero.

Some important points about Madaline are as follows:

- It is just like a multilayer perceptron, where Adaline will act as a hidden unit between the input and the Madaline layer.
- The weights and the bias between the input and Adaline layers, as in we see in the Adaline architecture, are adjustable.
- The Adaline and Madaline layers have fixed weights and bias of 1.
- Training can be done with the help of Delta rule.

BRIEFLY EXPLAIN THE ARCHITECTURE OF MADALINE

MADALINE (Many ADALINE) is a three-layer (input, hidden, output), fully connected, feed-forward artificial neural network architecture for classification that uses ADALINE units in its hidden and output layers, i.e. its activation function is the sign function. The three-layer network uses memistors. The architecture of Madaline consists of "n" neurons of the input layer, "m" neurons of the Adaline layer, and 1 neuron of the Madaline layer. The Adaline layer can be considered as the hidden layer as it is between the input layer and the output layer, i.e. the Madaline layer.



Training Algorithm of MADALINE

By now we know that only the weights and bias between the input and the Adaline layer are to be adjusted, and the weights and bias between the Adaline and the Madaline layer are fixed.

Step 1 – Initialize the following to start the training:

- Weights
- Bias
- Learning rate α

For easy calculation and simplicity, weights and bias must be set equal to 0 and the learning rate must be set equal to 1.

Step 2 – Continue step 3-8 when the stopping condition is not true.

Step 3 – Continue step 4-6 for every bipolar training pair s:t.

Step 4 – Activate each input unit as follows:

$$x_i = s_i (i = 1 \text{ to } n)$$

Step 5 – Obtain the net input at each hidden layer, i.e. the Adaline layer with the following relation:

$$O_{inj} = b_j + \sum_{i=1}^n x_i w_{ij} \quad j = 1 \text{ to } m$$

Here 'b' is bias and 'n' is the total number of input neurons.

Step 6 – Apply the following activation function to obtain the final output at the Adaline and the

Madaline Layer:

$$f(y_{in}) = \begin{cases} 1, & \text{if } x \geq 0 \\ -1, & \text{if } x < 0 \end{cases}$$

Output at the hidden Adaline unit

$Q_j = f(Q_{inj})$ Final output of the network $y = f(y_{in})$

i.e.

$$y_{inj} = b_0 + \sum_{j=1}^m Q_j v_j$$

Step 7 – Calculate the error and adjust the weights as follows –

Case 1 – if $y \neq t$ and $t = 1$ then,

$$\begin{aligned} w_{ij}(\text{new}) &= w_{ij}(\text{old}) + \alpha(1 - Q_{inj})x_i \\ b_j(\text{new}) &= b_j(\text{old}) + \alpha(1 - Q_{inj}) \end{aligned}$$

In this case, the weights would be updated on Q_j where the net input is close to 0 because $t = 1$.

Case 2 – if $y \neq t$ and $t = -1$ then,

$$\begin{aligned} w_{ik}(\text{new}) &= w_{ik}(\text{old}) + \alpha(-1 - Q_{ink})x_i \\ b_k(\text{new}) &= b_k(\text{old}) + \alpha(-1 - Q_{ink}) \end{aligned}$$

In this case, the weights would be updated on Q_k where the net input is positive because $t = -1$. Here 'y' is the actual output and 't' is the desired/target output.

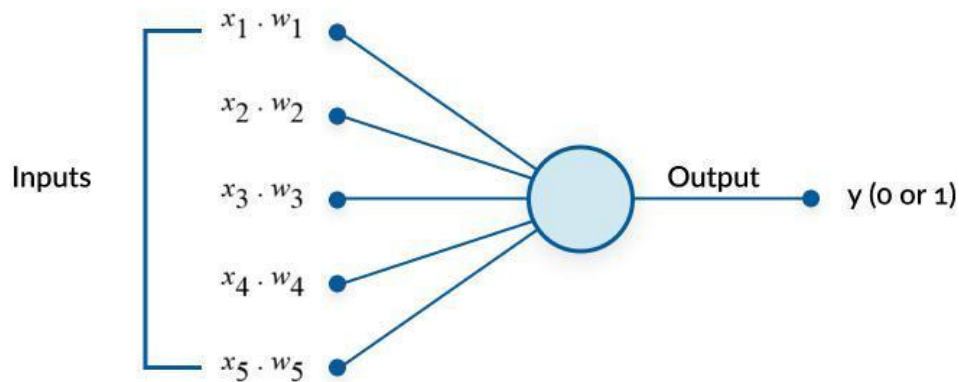
Case 3 – if $y = t$, then there would be no change in weights.

Step 8 – Test for the stopping condition, which will happen when there is no change in weight or the highest weight change occurred during training is smaller than the specified tolerance.

WHAT IS A PERCEPTRON?

A perceptron is a binary classification algorithm modeled after the functioning of the human brain—it was intended to emulate the neuron. The perceptron, while it has a simple structure, has the ability to learn.

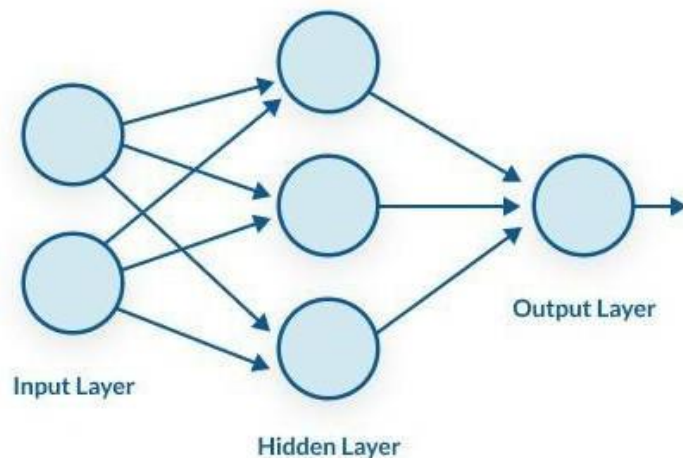
Perceptron Input And Output



What is Multilayer Perceptron?

A multilayer perceptron (MLP) is a group of perceptrons, organized in multiple layers, that can accurately answer complex questions. Each perceptron in the first layer (on the left) sends signals to all the perceptrons in the second layer, and so on. An MLP contains an input layer, at least one hidden layer, and an output layer.

Perceptron Input And Output



The perceptron learns as follows:

1. Takes the inputs which are fed into the perceptrons in the input layer, multiplies them by their weights, and computes the sum.
2. Adds the number one, multiplied by a “bias weight”. This is a technical step that makes it possible to move the output function of each perceptron (the activation function) up, down, left and right on the number graph.
3. Feeds the sum through the activation function—in a simple perceptron system, the activation function is a step function.
4. The result of the step function is the output.

A multilayer perceptron is quite similar to a modern neural network. By adding a few ingredients, the perceptron architecture becomes a full-fledged deep learning system:

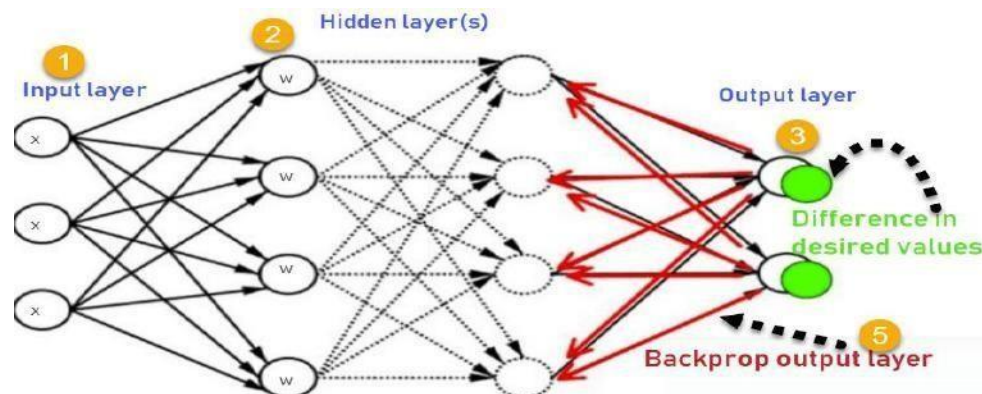
- **Activation functions and other hyperparameters:** a full neural network uses a variety of activation functions which output real values, not boolean values like in the classic perceptron. It is more flexible in terms of other details of the learning process, such as the number of training iterations (iterations and epochs), weight initialization schemes, regularization, and so on. All these can be tuned as hyperparameters.
- **Backpropagation:** a full neural network uses the backpropagation algorithm, to perform iterative backward passes which try to find the optimal values of perceptron weights, to generate the most accurate prediction.
- **Advanced architectures:** full neural networks can have a variety of architectures that can help solve specific problems. A few examples are Recurrent Neural Networks (RNN), Convolutional Neural Networks (CNN), and Generative Adversarial Networks (GAN).

WHAT IS BACKPROPAGATION AND WHY IS IT IMPORTANT?

After a neural network is defined with initial weights, and a forward pass is performed to generate the initial prediction, there is an error function which defines how far away the model is from the true prediction. There are many possible algorithms that can minimize the error function—for example, one could do a brute force search to find the weights that generate the smallest error. However, for large neural networks, a training algorithm is needed that is very computationally efficient. Backpropagation is that algorithm—it can discover the optimal weights relatively quickly, even for a network with millions of weights.

HOW BACKPROPAGATION WORKS?

1. **Forward pass**—weights are initialized and inputs from the training set are fed into the network. The forward pass is carried out and the model generates its initial prediction.
2. **Error function**—the error function is computed by checking how far away the prediction is from the known true value.
3. **Backpropagation with gradient descent**—the backpropagation algorithm calculates how much the output values are affected by each of the weights in the model. To do this, it calculates partial derivatives, going back from the error function to a specific neuron and its weight. This provides complete traceability from total errors, back to a specific weight which contributed to that error. The result of backpropagation is a set of weights that minimize the error function.
4. **Weight update**—weights can be updated after every sample in the training set, but this is usually not practical. Typically, a batch of samples is run in one big forward pass, and then backpropagation performed on the aggregate result. The *batch size* and number of batches used in training, called *iterations*, are important hyperparameters that are tuned to get the best results. Running the entire training set through the backpropagation process is called an *epoch*.



Training algorithm of BPNN:

1. Inputs X , arrive through the pre connected path
2. Input is modeled using real weights W . The weights are usually randomly selected.
3. Calculate the output for every neuron from the input layer, to the hidden layers, to the output layer.
4. Calculate the error in the outputs

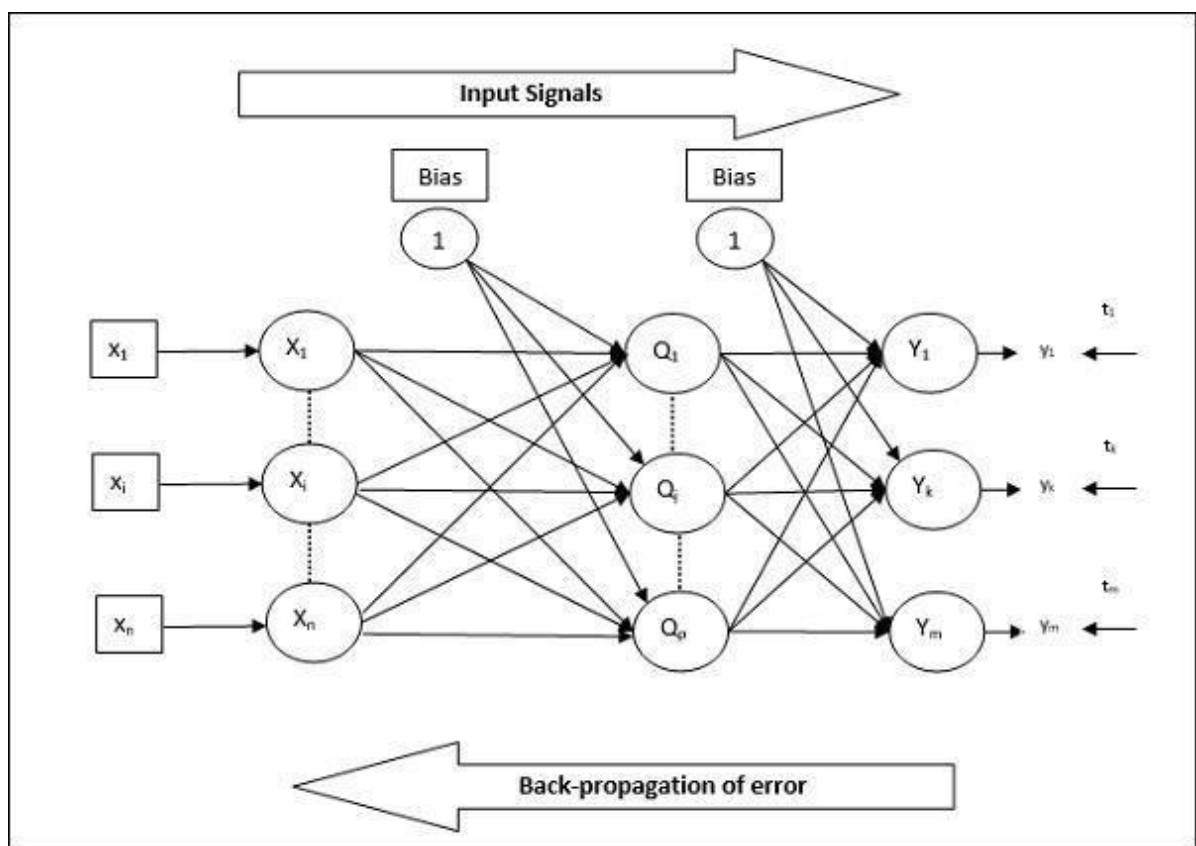
$$\text{Error}_B = \text{Actual Output} - \text{Desired Output}$$

5. Travel back from the output layer to the hidden layer to adjust the weights such that the error is decreased.

Keep repeating the process until the desired output is achieved

Architecture of back propagation network:

As shown in the diagram, the architecture of BPN has three interconnected layers having weights on them. The hidden layer as well as the output layer also has bias, whose weight is always 1, on them. As is clear from the diagram, the working of BPN is in two phases. One phase sends the signal from the input layer to the output layer, and the other phase back propagates the error from the output layer to the input layer.

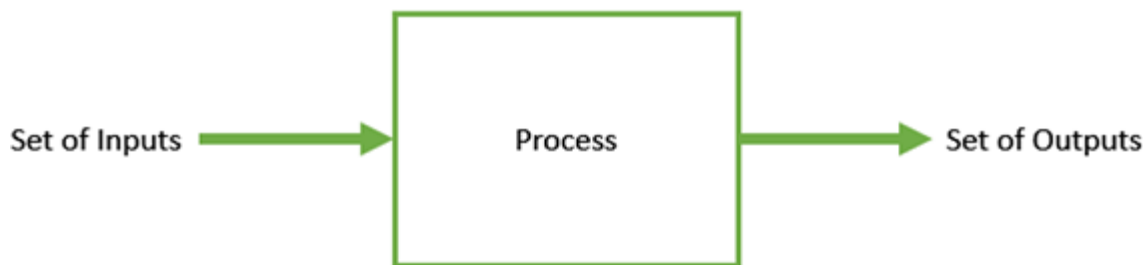


Genetic algorithm

Genetic Algorithm (GA) is a search-based optimization technique based on the principles of **Genetics and Natural Selection**. It is frequently used to find optimal or near-optimal solutions to difficult problems which otherwise would take a lifetime to solve. It is frequently used to solve optimization problems, in research, and in machine learning.

Introduction to Optimization

Optimization is the process of **making something better**. In any process, we have a set of inputs and a set of outputs as shown in the following figure.



Optimization refers to finding the values of inputs in such a way that we get the “best” output values. The definition of “best” varies from problem to problem, but in mathematical terms, it refers to maximizing or minimizing one or more objective functions, by varying the input parameters.

The set of all possible solutions or values which the inputs can take make up the search space. In this search space, lies a point or a set of points which gives the optimal solution. The aim of optimization is to find that point or set of points in the search space

What are Genetic Algorithms?

Nature has always been a great source of inspiration to all mankind. Genetic Algorithms (GAs) are search based algorithms based on the concepts of natural selection and genetics. GAs are a subset of a much larger branch of computation known as **Evolutionary Computation**.

GAs were developed by John Holland and his students and colleagues at the University of Michigan, most notably David E. Goldberg and has since been tried on various optimization problems with a high degree of success.

In GAs, we have a **pool or a population of possible solutions** to the given problem. These solutions then undergo recombination and mutation (like in natural genetics), producing new children, and the process is repeated over various generations. Each individual (or candidate solution) is assigned a fitness value (based on its objective function value) and the fitter individuals are given a higher chance to mate and yield more “fitter” individuals. This is in line with the Darwinian Theory of “Survival of the Fittest”.

In this way we keep “evolving” better individuals or solutions over generations, till we reach a stopping criterion.

Genetic Algorithms are sufficiently randomized in nature, but they perform much better than random local search (in which we just try various random solutions, keeping track of the best so far), as they exploit historical information as well.

Advantages of GAs

GAs have various advantages which have made them immensely popular. These include –

- Does not require any derivative information (which may not be available for many real-world problems).
- Is faster and more efficient as compared to the traditional methods.
- Has very good parallel capabilities.
- Optimizes both continuous and discrete functions and also multi-objective problems.
- Provides a list of “good” solutions and not just a single solution.
- Always gets an answer to the problem, which gets better over the time.
- Useful when the search space is very large and there are a large number of parameters involved.

Limitations of GAs

Like any technique, GAs also suffer from a few limitations. These include –

- GAs are not suited for all problems, especially problems which are simple and for which derivative information is available.
- Fitness value is calculated repeatedly which might be computationally expensive for some problems.
- Being stochastic, there are no guarantees on the optimality or the quality of the solution.
- If not implemented properly, the GA may not converge to the optimal solution.

GA – Motivation

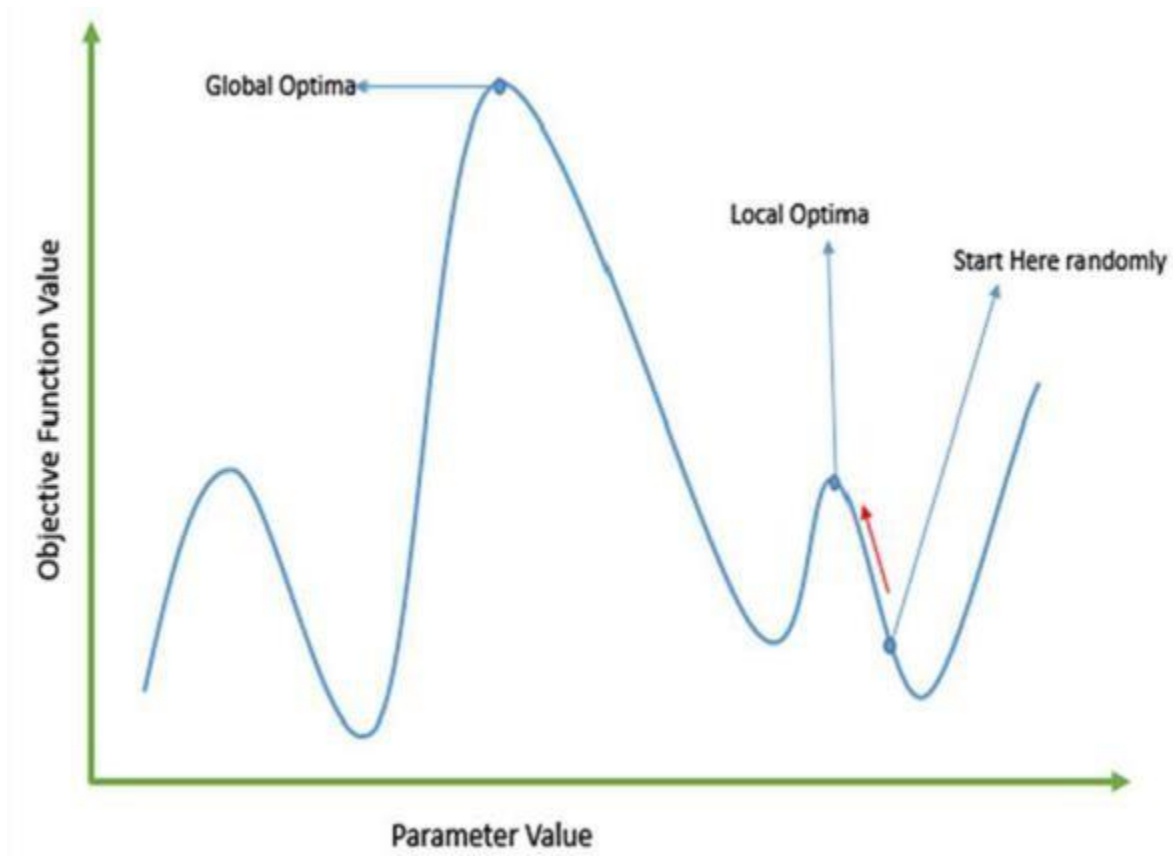
Genetic Algorithms have the ability to deliver a “good-enough” solution “fast-enough”. This makes genetic algorithms attractive for use in solving optimization problems. The reasons why GAs are needed are as follows –

Solving Difficult Problems

In computer science, there is a large set of problems, which are **NP-Hard**. What this essentially means is that, even the most powerful computing systems take a very long time (even years!) to solve that problem. In such a scenario, GAs prove to be an efficient tool to provide **usable near-optimal solutions** in a short amount of time.

Failure of Gradient Based Methods

Traditional calculus based methods work by starting at a random point and by moving in the direction of the gradient, till we reach the top of the hill. This technique is efficient and works very well for single-peaked objective functions like the cost function in linear regression. But, in most real-world situations, we have a very complex problem called as landscapes, which are made of many peaks and many valleys, which causes such methods to fail, as they suffer from an inherent tendency of getting stuck at the local optima as shown in the following figure.



Getting a Good Solution Fast

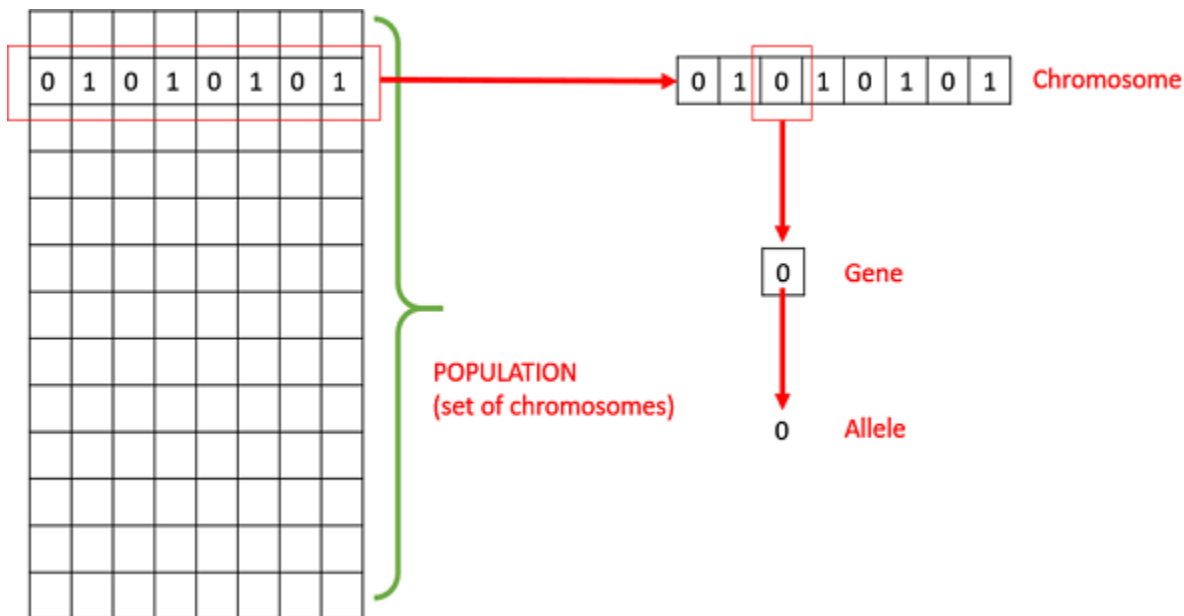
Some difficult problems like the Travelling Salesperson Problem (TSP), have real-world applications like path finding and VLSI Design. Now imagine that you are using your GPS Navigation system, and it takes a few minutes (or even a few hours) to compute the “optimal” path from the source to destination. Delay in such real world applications is not acceptable and therefore a “good-enough” solution, which is delivered “fast” is what is required.

This section introduces the basic terminology required to understand GAs. Also, a generic structure of GAs is presented in both **pseudo-code** and **graphical forms**. The reader is advised to properly understand all the concepts introduced in this section and keep them in mind when reading other sections of this tutorial as well.

Basic Terminology

Before beginning a discussion on Genetic Algorithms, it is essential to be familiar with some basic terminology which will be used throughout this tutorial.

- **Population** – It is a subset of all the possible (encoded) solutions to the given problem. The population for a GA is analogous to the population for human beings except that instead of human beings, we have Candidate Solutions representing human beings.
- **Chromosomes** – A chromosome is one such solution to the given problem.
- **Gene** – A gene is one element position of a chromosome.
- **Allele** – It is the value a gene takes for a particular chromosome.

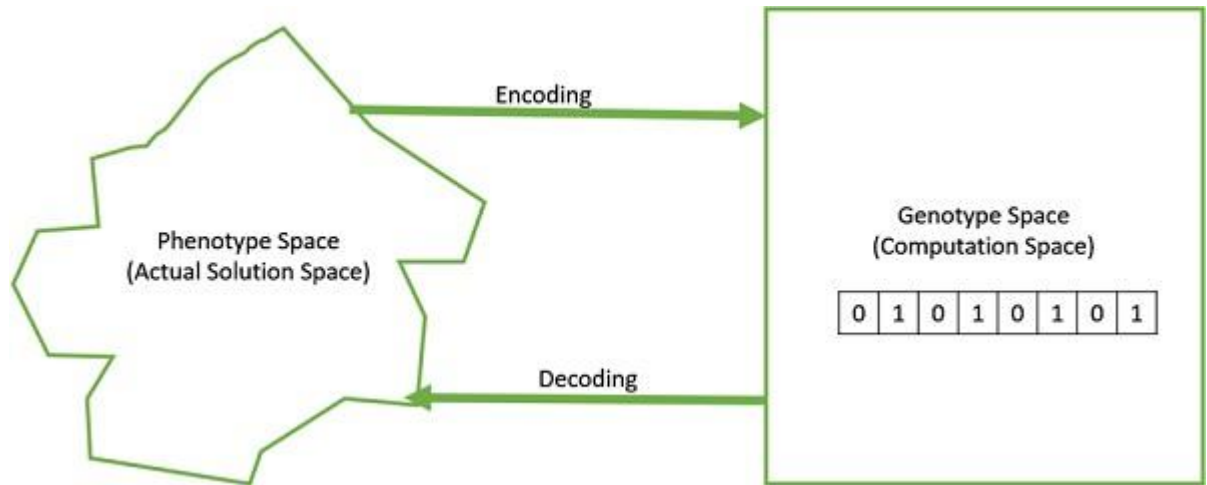


- **Genotype** – Genotype is the population in the computation space. In the computation space, the solutions are represented in a way which can be easily understood and manipulated using a computing system.
- **Phenotype** – Phenotype is the population in the actual real world solution space in which solutions are represented in a way they are represented in real world situations.
- **Decoding and Encoding** – For simple problems, the **phenotype and genotype** spaces are the same. However, in most of the cases, the phenotype and genotype spaces are different. Decoding is a process of transforming a solution from the genotype to the phenotype space, while encoding is a process of transforming from the phenotype to genotype space. Decoding should be fast as it is carried out repeatedly in a GA during the fitness value calculation.

For example, consider the 0/1 Knapsack Problem. The Phenotype space consists of solutions which just contain the item numbers of the items to be picked.

However, in the genotype space it can be represented as a binary string of length n (where n is the number of items). A **0 at position x** represents that x^{th} item is picked

while a 1 represents the reverse. This is a case where genotype and phenotype spaces are different.

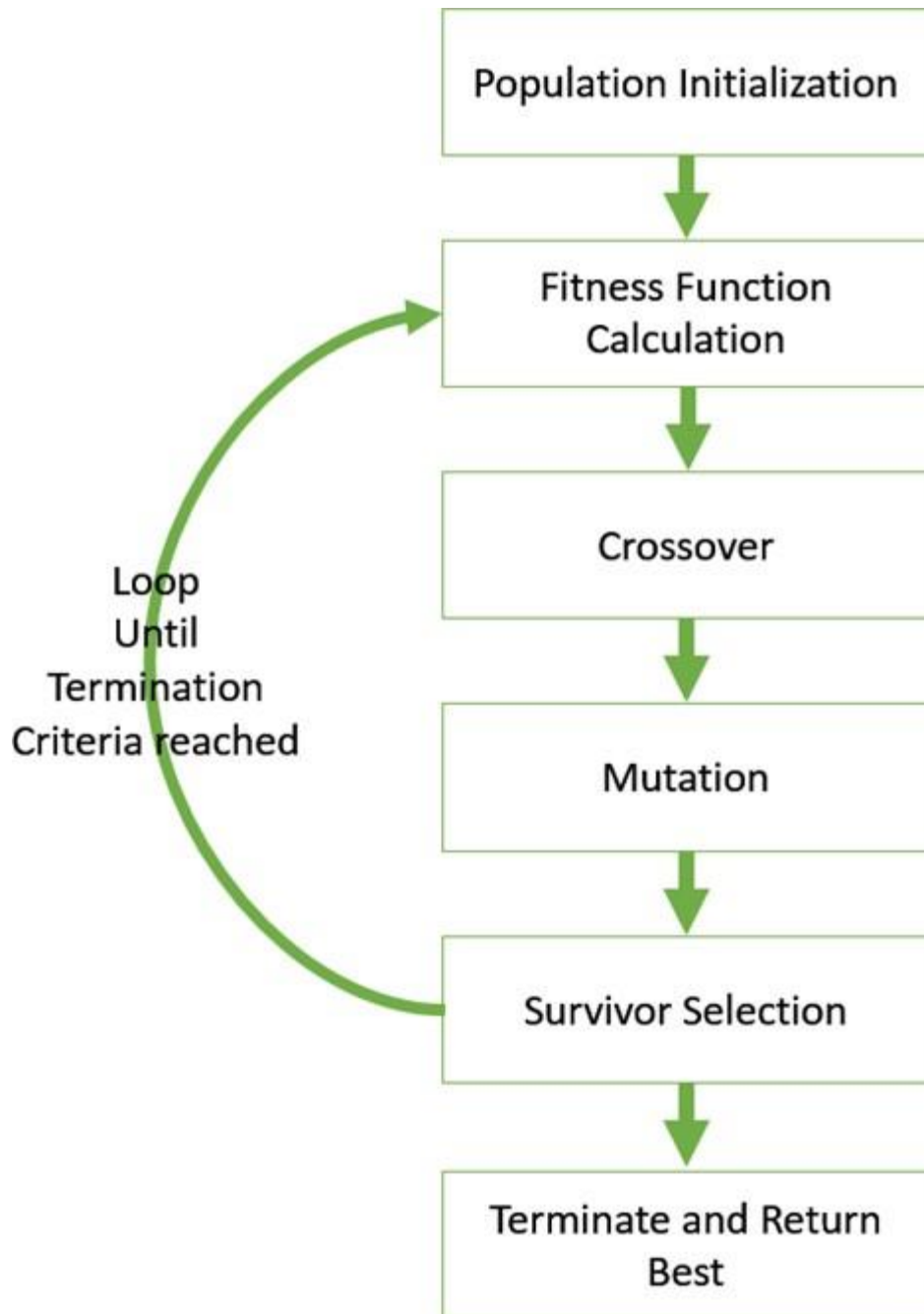


Basic Structure

The basic structure of a GA is as follows –

We start with an initial population (which may be generated at random or seeded by other heuristics), select parents from this population for mating. Apply crossover and mutation operators on the parents to generate new off-springs. And finally these off-springs replace the existing individuals in the population and the process repeats. In this way genetic algorithms actually try to mimic the human evolution to some extent.

Each of the following steps are covered as a separate chapter later in this tutorial.



A generalized pseudo-code for a GA is explained in the following program –

```
GA()  
initialize population  
find fitness of population  
  
while (termination criteria is reached) do  
parent selection  
crossover with probability pc
```

```
mutation with probability pm
decode and fitness calculation
survivor selection
find best
return best
```

One of the most important decisions to make while implementing a genetic algorithm is deciding the representation that we will use to represent our solutions. It has been observed that improper representation can lead to poor performance of the GA.

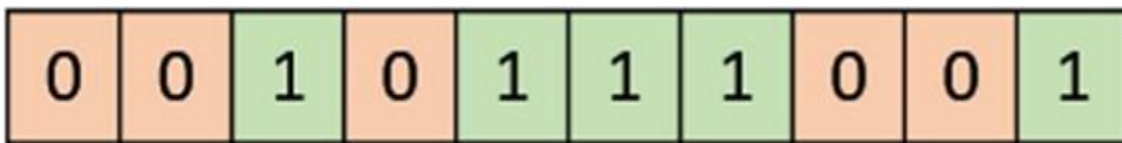
Therefore, choosing a proper representation, having a proper definition of the mappings between the phenotype and genotype spaces is essential for the success of a GA.

In this section, we present some of the most commonly used representations for genetic algorithms. However, representation is highly problem specific and the reader might find that another representation or a mix of the representations mentioned here might suit his/her problem better.

Binary Representation

This is one of the simplest and most widely used representation in GAs. In this type of representation the genotype consists of bit strings.

For some problems when the solution space consists of Boolean decision variables – yes or no, the binary representation is natural. Take for example the 0/1 Knapsack Problem. If there are n items, we can represent a solution by a binary string of n elements, where the x^{th} element tells whether the item x is picked (1) or not (0).



For other problems, specifically those dealing with numbers, we can represent the numbers with their binary representation. The problem with this kind of encoding is that different bits have different significance and therefore mutation and crossover operators can have undesired consequences. This can be resolved to some extent by using **Gray Coding**, as a change in one bit does not have a massive effect on the solution.

Real Valued Representation

For problems where we want to define the genes using continuous rather than discrete variables, the real valued representation is the most natural. The precision of these real valued or floating point numbers is however limited to the computer.

0.5	0.2	0.6	0.8	0.7	0.4	0.3	0.2	0.1	0.9
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

Integer Representation

For discrete valued genes, we cannot always limit the solution space to binary „yes“ or „no“. Forexample, if we want to encode the four distances – North, South, East and West, we can encode them as **{0,1,2,3}**. In such cases, integer representation is desirable.

1	2	3	4	3	2	4	1	2	1
---	---	---	---	---	---	---	---	---	---

Permutation Representation

In many problems, the solution is represented by an order of elements. In such cases permutation representation is the most suited.

A classic example of this representation is the travelling salesman problem (TSP). In this the salesman has to take a tour of all the cities, visiting each city exactly once and come back to the starting city. The total distance of the tour has to be minimized. The solution to this TSP is naturally an ordering or permutation of all the cities and therefore using a permutation representation makes sense for this problem.

1	5	9	8	7	4	2	3	6	0
---	---	---	---	---	---	---	---	---	---

Population is a subset of solutions in the current generation. It can also be defined as a set of chromosomes. There are several things to be kept in mind when dealing with GA population –

- The diversity of the population should be maintained otherwise it might lead to premature convergence.
- The population size should not be kept very large as it can cause a GA to slow down, while a smaller population might not be enough for a good mating pool. Therefore, an optimal population size needs to be decided by trial and error.

The population is usually defined as a two dimensional array of - **size population, size x, chromosome size**.

Population Initialization

There are two primary methods to initialize a population in a GA. They are –

- **Random Initialization** – Populate the initial population with completely random solutions.
- **Heuristic initialization** – Populate the initial population using a known heuristic for the problem.

It has been observed that the entire population should not be initialized using a heuristic, as it can result in the population having similar solutions and very little diversity. It has been experimentally observed that the random solutions are the ones to drive the population to optimality. Therefore, with heuristic initialization, we just seed the population with a couple of good solutions, filling up the rest with random solutions rather than filling the entire population with heuristic based solutions.

It has also been observed that heuristic initialization in some cases, only effects the initial fitness of the population, but in the end, it is the diversity of the solutions which lead to optimality.

Population Models

There are two population models widely in use –

Steady State

In steady state GA, we generate one or two off-springs in each iteration and they replace one or two individuals from the population. A steady state GA is also known as **Incremental GA**.

Generational

In a generational model, we generate „n“ off-springs, where n is the population size, and the entire population is replaced by the new one at the end of the iteration.

The fitness function simply defined is a function which takes a **candidate solution to the problem as input and produces as output** how “fit” or how “good” the solution is with respect to the problem in consideration.

Calculation of fitness value is done repeatedly in a GA and therefore it should be sufficiently fast. A slow computation of the fitness value can adversely affect a GA and make it exceptionally slow.

In most cases the fitness function and the objective function are the same as the objective is to either maximize or minimize the given objective function. However, for more complex problems with multiple objectives and constraints, an **Algorithm Designer** might choose to have a different fitness function.

A fitness function should possess the following characteristics –

- The fitness function should be sufficiently fast to compute.
- It must quantitatively measure how fit a given solution is or how fit individuals can be produced from the given solution.

In some cases, calculating the fitness function directly might not be possible due to the inherent complexities of the problem at hand. In such cases, we do fitness approximation to suit our needs.

The following image shows the fitness calculation for a solution of the 0/1 Knapsack. It is a simple fitness function which just sums the profit values of the items being picked (which have a 1), scanning the elements from left to right till the knapsack is full.

0	1	2	3	4	5	6	Item Number
0	1	0	1	1	0	1	Chromosome
2	9	8	5	4	0	2	Profit Values
7	5	3	1	5	9	8	Weight Values
Knapsack capacity = 15							
Total associated profit = 18							
Last item not picked as it exceeds knapsack capacity							

Parent Selection is the process of selecting parents which mate and recombine to create offsprings for the next generation. Parent selection is very crucial to the convergence rate of the GA as good parents drive individuals to a better and fitter solutions.

However, care should be taken to prevent one extremely fit solution from taking over the entire population in a few generations, as this leads to the solutions being close to one another in the solution space thereby leading to a loss of diversity. **Maintaining good diversity** in the population is extremely crucial for the success of a GA. This taking up of the entire population by one extremely fit solution is known as **premature convergence** and is an undesirable condition in a GA.

Fitness Proportionate Selection

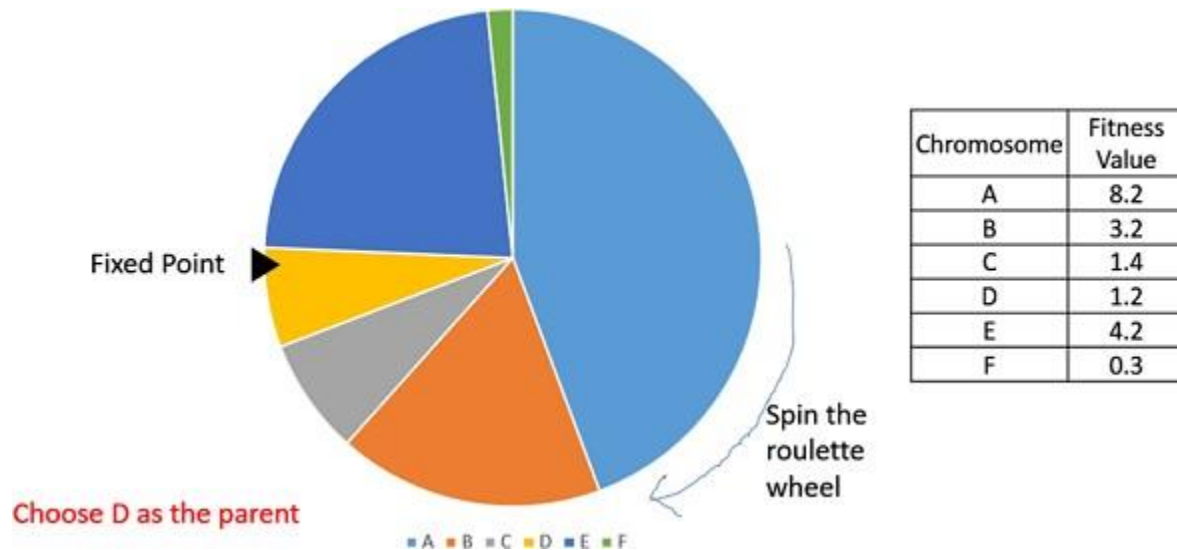
Fitness Proportionate Selection is one of the most popular ways of parent selection. In this every individual can become a parent with a probability which is proportional to its fitness. Therefore, fitter individuals have a higher chance of mating and propagating their features to the next generation. Therefore, such a selection strategy applies a selection pressure to the more fit individuals in the population, evolving better individuals over time.

Consider a circular wheel. The wheel is divided into **n pies**, where n is the number of individuals in the population. Each individual gets a portion of the circle which is proportional to its fitness value.

Two implementations of fitness proportionate selection are possible –

Roulette Wheel Selection

In a roulette wheel selection, the circular wheel is divided as described before. A fixed point is chosen on the wheel circumference as shown and the wheel is rotated. The region of the wheel which comes in front of the fixed point is chosen as the parent. For the second parent, the same process is repeated.



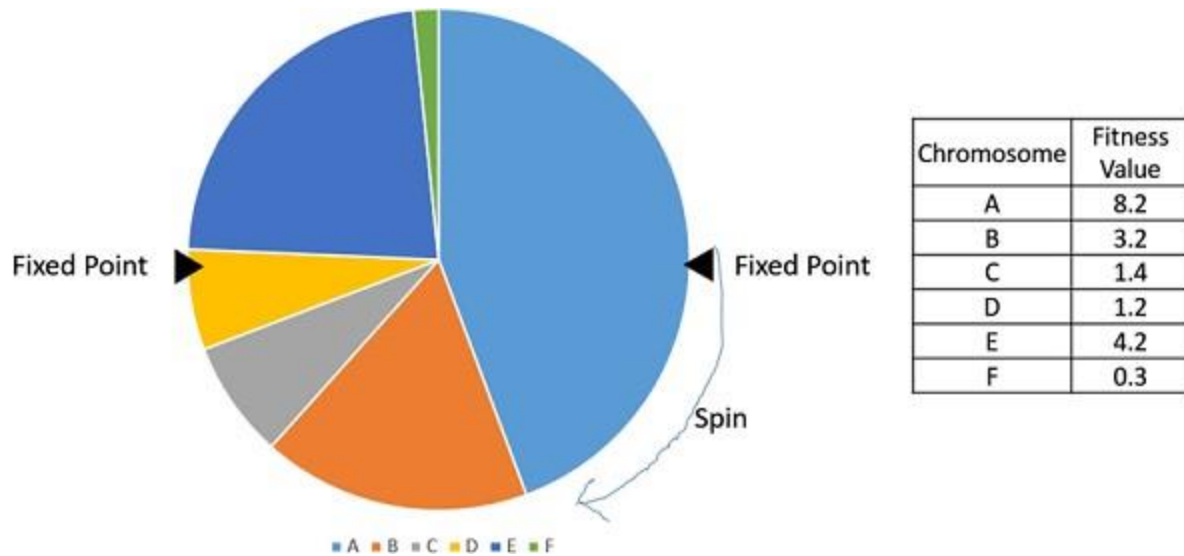
It is clear that a fitter individual has a greater pie on the wheel and therefore a greater chance of landing in front of the fixed point when the wheel is rotated. Therefore, the probability of choosing an individual depends directly on its fitness.

Implementation wise, we use the following steps –

- Calculate S = the sum of fitnesses.
- Generate a random number between 0 and S .
- Starting from the top of the population, keep adding the fitnesses to the partial sum P , till $P < S$.
- The individual for which P exceeds S is the chosen

individual. Stochastic Universal Sampling (SUS)

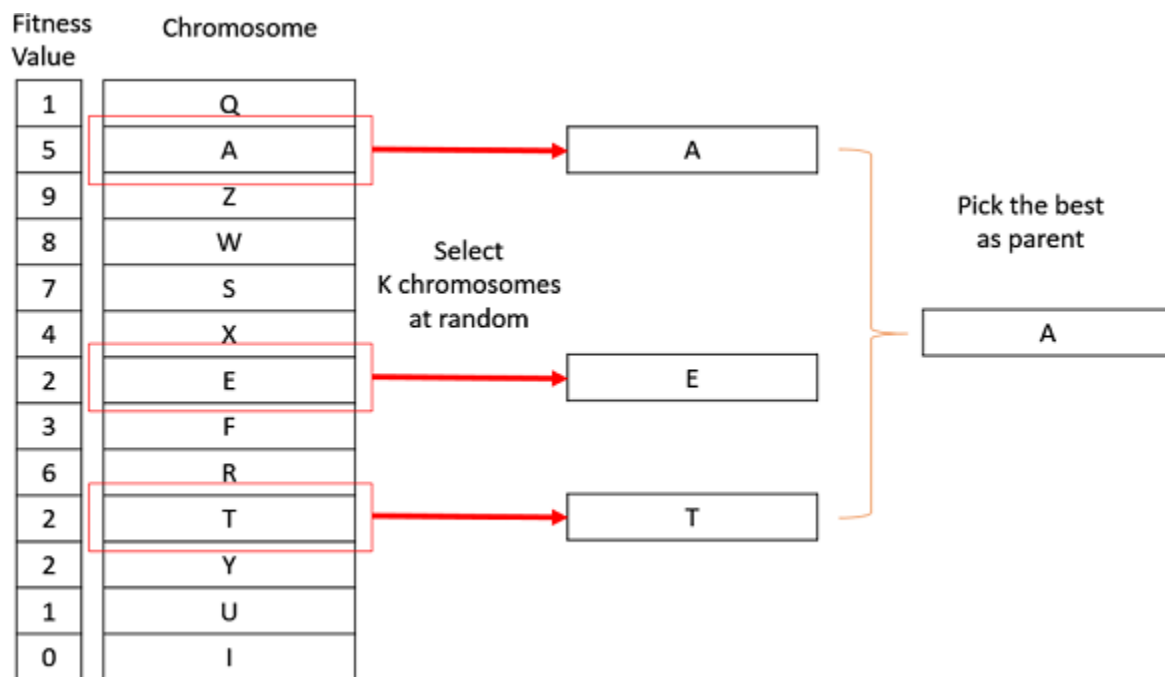
Stochastic Universal Sampling is quite similar to Roulette wheel selection, however instead of having just one fixed point, we have multiple fixed points as shown in the following image. Therefore, all the parents are chosen in just one spin of the wheel. Also, such a setup encourages the highly fit individuals to be chosen at least once.



It is to be noted that fitness proportionate selection methods don't work for cases where the fitness can take a negative value.

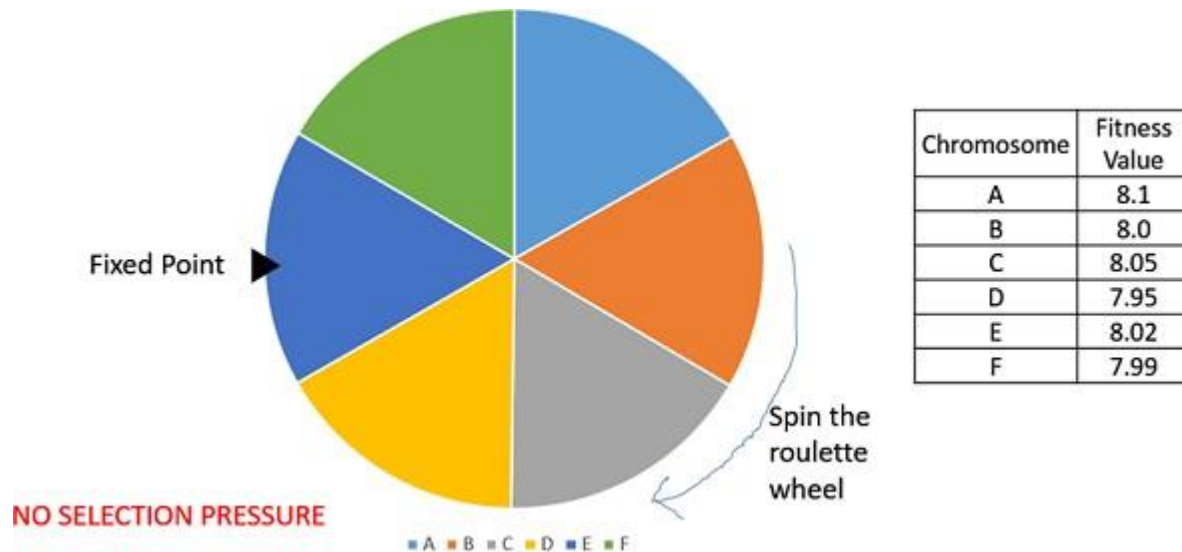
Tournament Selection

In K-Way tournament selection, we select K individuals from the population at random and select the best out of these to become a parent. The same process is repeated for selecting the next parent. Tournament Selection is also extremely popular in literature as it can even work with negative fitness values.



Rank Selection

Rank Selection also works with negative fitness values and is mostly used when the individuals in the population have very close fitness values (this happens usually at the end of the run). This leads to each individual having an almost equal share of the pie (like in case of fitness proportionate selection) as shown in the following image and hence each individual no matter how fit relative to each other has an approximately same probability of getting selected as a parent. This in turn leads to a loss in the selection pressure towards fitter individuals, making the GA to make poor parent selections in such situations.



In this, we remove the concept of a fitness value while selecting a parent. However, every individual in the population is ranked according to their fitness. The selection of the parents depends on the rank of each individual and not the fitness. The higher ranked individuals are preferred more than the lower ranked ones.

Chromosome	Fitness Value	Rank
A	8.1	1
B	8.0	4
C	8.05	2
D	7.95	6
E	8.02	3
F	7.99	5

Random Selection

In this strategy we randomly select parents from the existing population. There is no selection pressure towards fitter individuals and therefore this strategy is usually avoided.

Introduction to Crossover

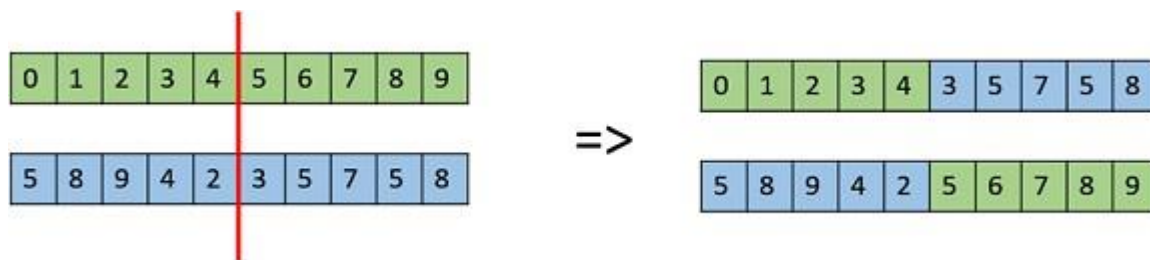
The crossover operator is analogous to reproduction and biological crossover. In this more than one parent is selected and one or more off-springs are produced using the genetic material of the parents. Crossover is usually applied in a GA with a high probability – p_c .

Crossover Operators

In this section we will discuss some of the most popularly used crossover operators. It is to be noted that these crossover operators are very generic and the GA Designer might choose to implement a problem-specific crossover operator as well.

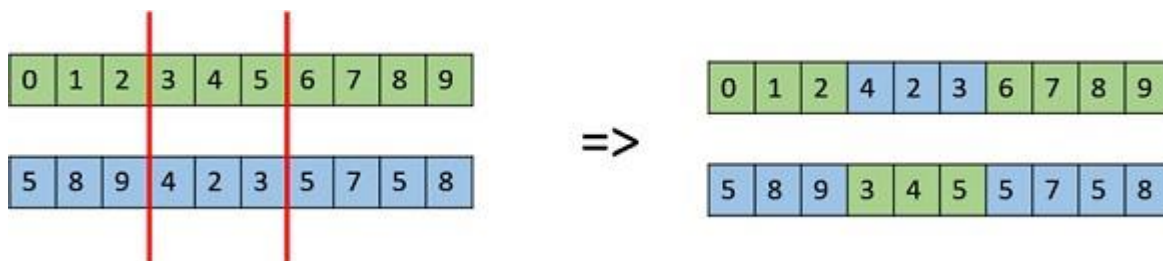
One Point Crossover

In this one-point crossover, a random crossover point is selected and the tails of its two parents are swapped to get new off-springs.



Multi Point Crossover

Multi point crossover is a generalization of the one-point crossover wherein alternating segments are swapped to get new off-springs.



Uniform Crossover

In a uniform crossover, we don't divide the chromosome into segments, rather we treat each gene separately. In this, we essentially flip a coin for each chromosome to decide whether or not

it'll be included in the off-spring. We can also bias the coin to one parent, to have more genetic material in the child from that parent.



Whole Arithmetic Recombination

This is commonly used for integer representations and works by taking the weighted average of the two parents by using the following formulae –

- $Child1 = \alpha \cdot x + (1-\alpha) \cdot y$
- $Child2 = \alpha \cdot x + (1-\alpha) \cdot y$

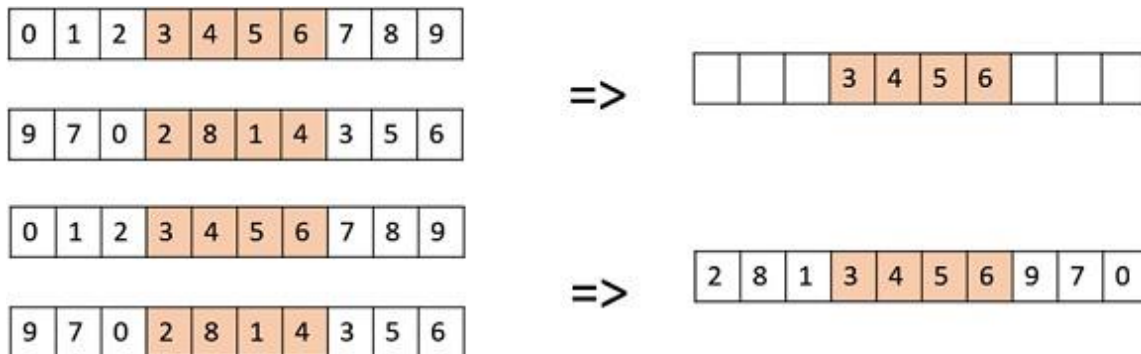
Obviously, if $\alpha = 0.5$, then both the children will be identical as shown in the following image.



Davis' Order Crossover (OX1)

OX1 is used for permutation based crossovers with the intention of transmitting information about relative ordering to the off-springs. It works as follows –

- Create two random crossover points in the parent and copy the segment between them from the first parent to the first offspring.
- Now, starting from the second crossover point in the second parent, copy the remaining unused numbers from the second parent to the first child, wrapping around the list.
- Repeat for the second child with the parent's role reversed.



Repeat the same procedure to get the second child

There exist a lot of other crossovers like Partially Mapped Crossover (PMX), Order based crossover (OX2), Shuffle Crossover, Ring Crossover, etc.

Introduction to Mutation

In simple terms, mutation may be defined as a small random tweak in the chromosome, to get a new solution. It is used to maintain and introduce diversity in the genetic population and is usually applied with a low probability – p_m . If the probability is very high, the GA gets reduced to a random search.

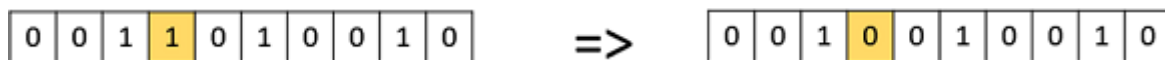
Mutation is the part of the GA which is related to the “exploration” of the search space. It has been observed that mutation is essential to the convergence of the GA while crossover is not.

Mutation Operators

In this section, we describe some of the most commonly used mutation operators. Like the crossover operators, this is not an exhaustive list and the GA designer might find a combination of these approaches or a problem-specific mutation operator more useful.

Bit Flip Mutation

In this bit flip mutation, we select one or more random bits and flip them. This is used for binary encoded GAs.

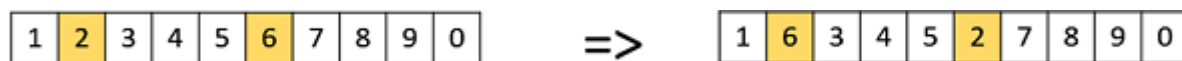


Random Resetting

Random Resetting is an extension of the bit flip for the integer representation. In this, a random value from the set of permissible values is assigned to a randomly chosen gene.

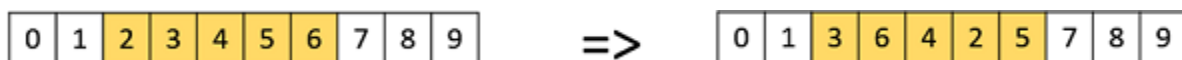
Swap Mutation

In swap mutation, we select two positions on the chromosome at random, and interchange the values. This is common in permutation based encodings.



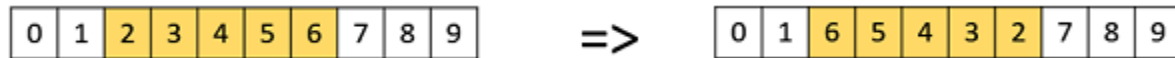
Scramble Mutation

Scramble mutation is also popular with permutation representations. In this, from the entire chromosome, a subset of genes is chosen and their values are scrambled or shuffled randomly.



Inversion Mutation

In inversion mutation, we select a subset of genes like in scramble mutation, but instead of shuffling the subset, we merely invert the entire string in the subset.



The Survivor Selection Policy determines which individuals are to be kicked out and which are to be kept in the next generation. It is crucial as it should ensure that the fitter individuals are not kicked out of the population, while at the same time diversity should be maintained in the population.

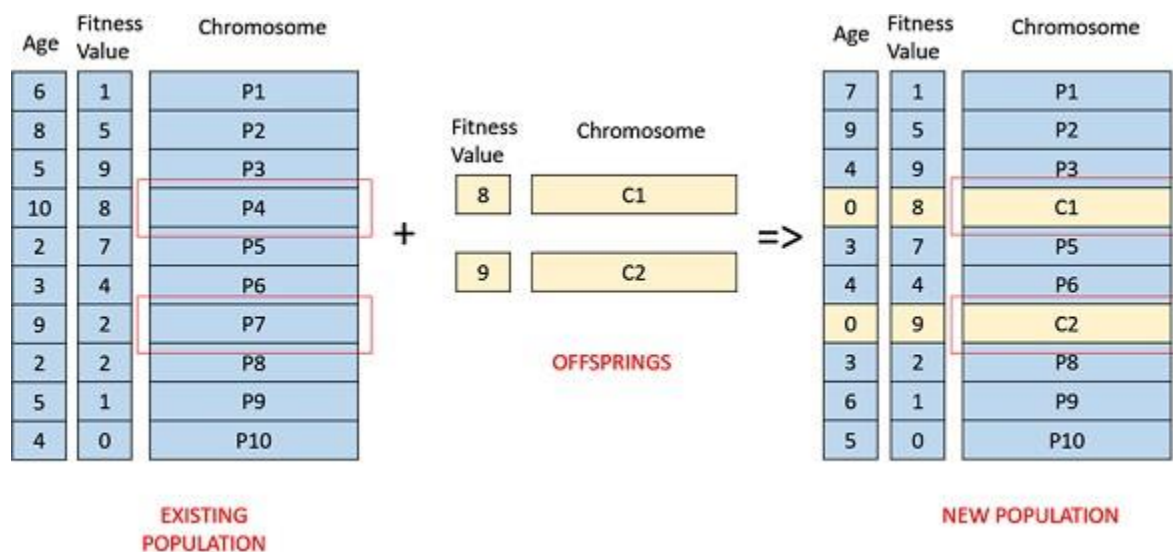
Some GAs employ **Elitism**. In simple terms, it means the current fittest member of the population is always propagated to the next generation. Therefore, under no circumstance can the fittest member of the current population be replaced.

The easiest policy is to kick random members out of the population, but such an approach frequently has convergence issues, therefore the following strategies are widely used.

Age Based Selection

In Age-Based Selection, we don't have a notion of a fitness. It is based on the premise that each individual is allowed in the population for a finite generation where it is allowed to reproduce, after that, it is kicked out of the population no matter how good its fitness is.

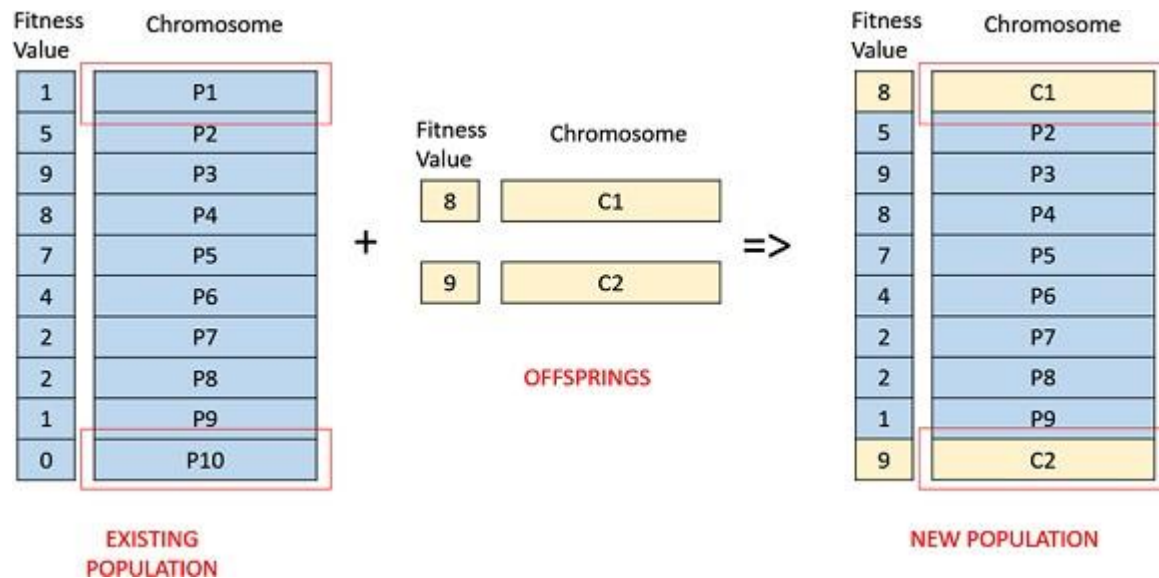
For instance, in the following example, the age is the number of generations for which the individual has been in the population. The oldest members of the population i.e. P4 and P7 are kicked out of the population and the ages of the rest of the members are incremented by one.



Fitness Based Selection

In this fitness based selection, the children tend to replace the least fit individuals in the population. The selection of the least fit individuals may be done using a variation of any of the selection policies described before – tournament selection, fitness proportionate selection, etc.

For example, in the following image, the children replace the least fit individuals P1 and P10 of the population. It is to be noted that since P1 and P9 have the same fitness value, the decision to remove which individual from the population is arbitrary.



The termination condition of a Genetic Algorithm is important in determining when a GA run will end. It has been observed that initially, the GA progresses very fast with better solutions coming in every few iterations, but this tends to saturate in the later stages where the improvements are very small. We usually want a termination condition such that our solution is close to the optimal, at the end of the run.

Usually, we keep one of the following termination conditions –

- When there has been no improvement in the population for X iterations.
- When we reach an absolute number of generations.
- When the objective function value has reached a certain pre-defined value.

For example, in a genetic algorithm we keep a counter which keeps track of the generations for which there has been no improvement in the population. Initially, we set this counter to zero. Each time we don't generate off-springs which are better than the individuals in the population, we increment the counter.

However, if the fitness any of the off-springs is better, then we reset the counter to zero. The algorithm terminates when the counter reaches a predetermined value.

Like other parameters of a GA, the termination condition is also highly problem specific and the GA designer should try out various options to see what suits his particular problem the best.

Till now in this tutorial, whatever we have discussed corresponds to the Darwinian model of evolution – natural selection and genetic variation through recombination and mutation. In nature, only the information contained in the individual's genotype can be transmitted to the next generation. This is the approach which we have been following in the tutorial so far.

However, other models of lifetime adaptation – **Lamarckian Model** and **Baldwinian Model** also do exist. It is to be noted that whichever model is the best, is open for debate and the results obtained by researchers show that the choice of lifetime adaptation is highly problem specific.

Often, we hybridize a GA with local search – like in Memetic Algorithms. In such cases, one might choose to go with either Lamarckian or Baldwinian Model to decide what to do with individuals generated after the local search.

Lamarckian Model

The Lamarckian Model essentially says that the traits which an individual acquires in his/her lifetime can be passed on to its offspring. It is named after French biologist Jean-Baptiste Lamarck.

Even though, natural biology has completely disregarded Lamarckism as we all know that only the information in the genotype can be transmitted. However, from a computation view point, it has been shown that adopting the Lamarckian model gives good results for some of the problems.

In the Lamarckian model, a local search operator examines the neighborhood (acquiring new traits), and if a better chromosome is found, it becomes the offspring.

Baldwinian Model

The Baldwinian model is an intermediate idea named after James Mark Baldwin (1896). In the Baldwin model, the chromosomes can encode a tendency of learning beneficial behaviors. This means, that unlike the Lamarckian model, we don't transmit the acquired traits to the next generation, and neither do we completely ignore the acquired traits like in the Darwinian Model.

The Baldwin Model is in the middle of these two extremes, wherein the tendency of an individual to acquire certain traits is encoded rather than the traits themselves.

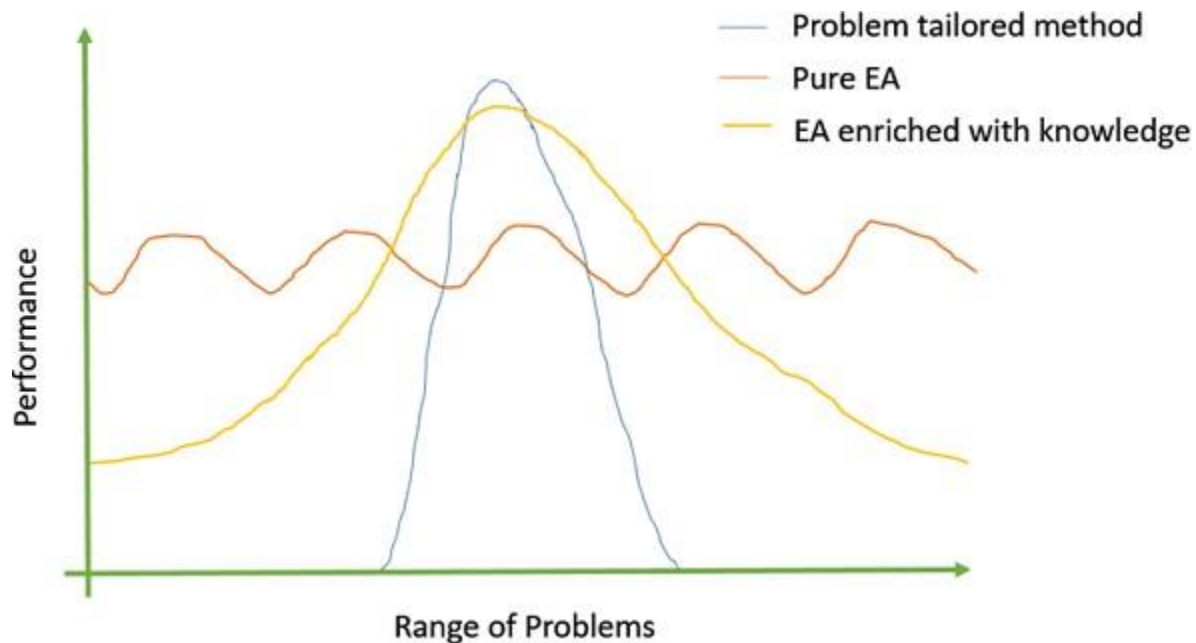
In this Baldwinian Model, a local search operator examines the neighborhood (acquiring new traits), and if a better chromosome is found, it only assigns the improved fitness to the chromosome and does not modify the chromosome itself. The change in fitness signifies the chromosome's capability to "acquire the trait", even though it is not passed directly to the future generations.

GAs are very general in nature, and just applying them to any optimization problem wouldn't give good results. In this section, we describe a few points which would help and assist a GA designer or GA implementer in their work.

Introduce problem-specific domain knowledge

It has been observed that the more problem-specific domain knowledge we incorporate into the GA; the better objective values we get. Adding problem specific information can be done by either using problem specific crossover or mutation operators, custom representations, etc.

The following image shows Michalewicz's (1990) view of the EA –



Reduce Crowding

Crowding happens when a highly fit chromosome gets to reproduce a lot, and in a few generations, the entire population is filled with similar solutions having similar fitness. This reduces diversity which is a very crucial element to ensure the success of a GA. There are numerous ways to limit crowding. Some of them are –

- **Mutation** to introduce diversity.
- Switching to **rank selection** and **tournament selection** which have more selection pressure than fitness proportionate selection for individuals with similar fitness.
- **Fitness Sharing** – In this an individual's fitness is reduced if the population already contains similar individuals.

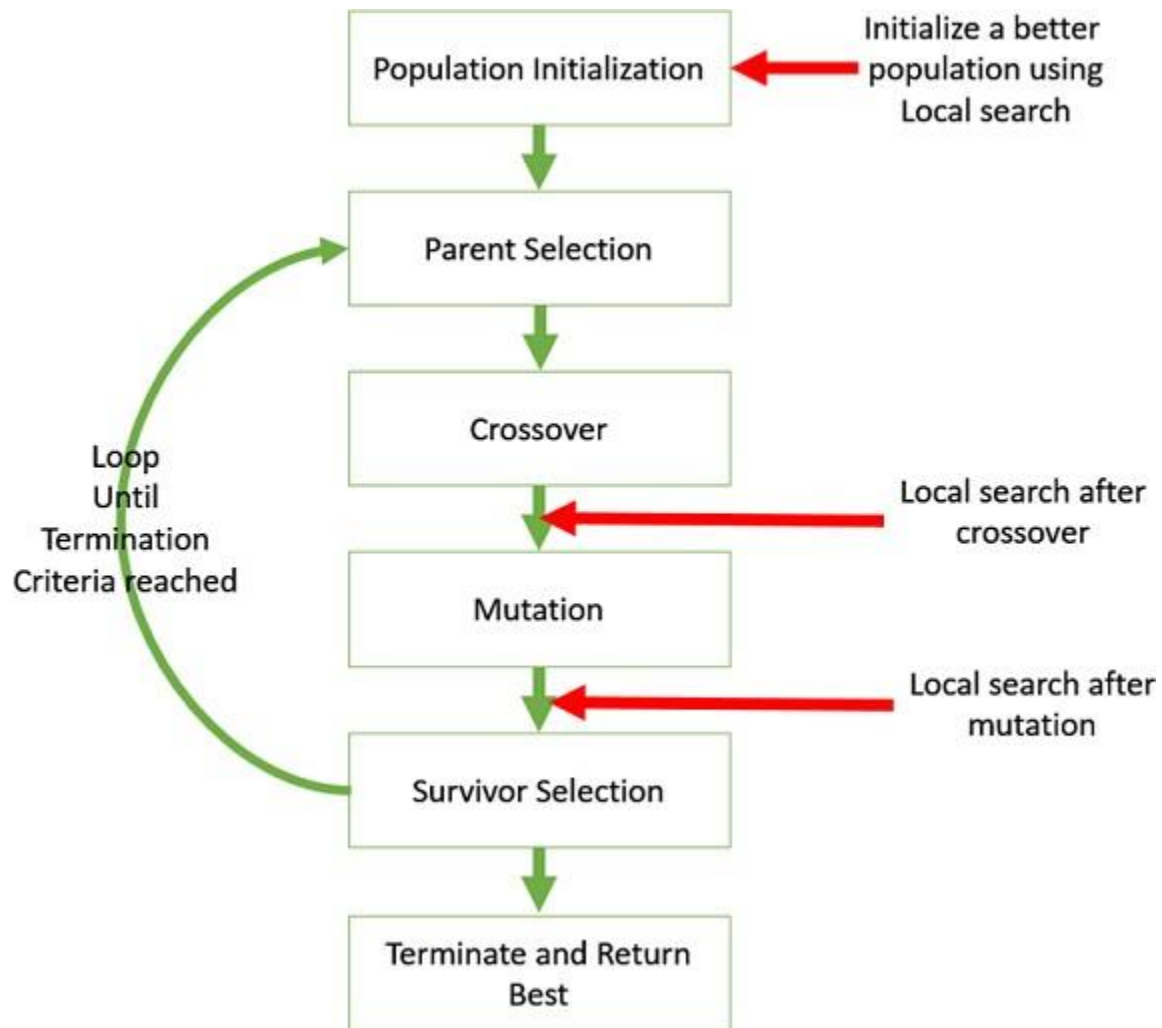
Randomization Helps!

It has been experimentally observed that the best solutions are driven by randomized chromosomes as they impart diversity to the population. The GA implementer should be careful to keep sufficient amount of randomization and diversity in the population for the best results.

Hybridize GA with Local Search

Local search refers to checking the solutions in the neighborhood of a given solution to look for better objective values.

It may be sometimes useful to hybridize the GA with local search. The following image shows the various places in which local search can be introduced in a GA.



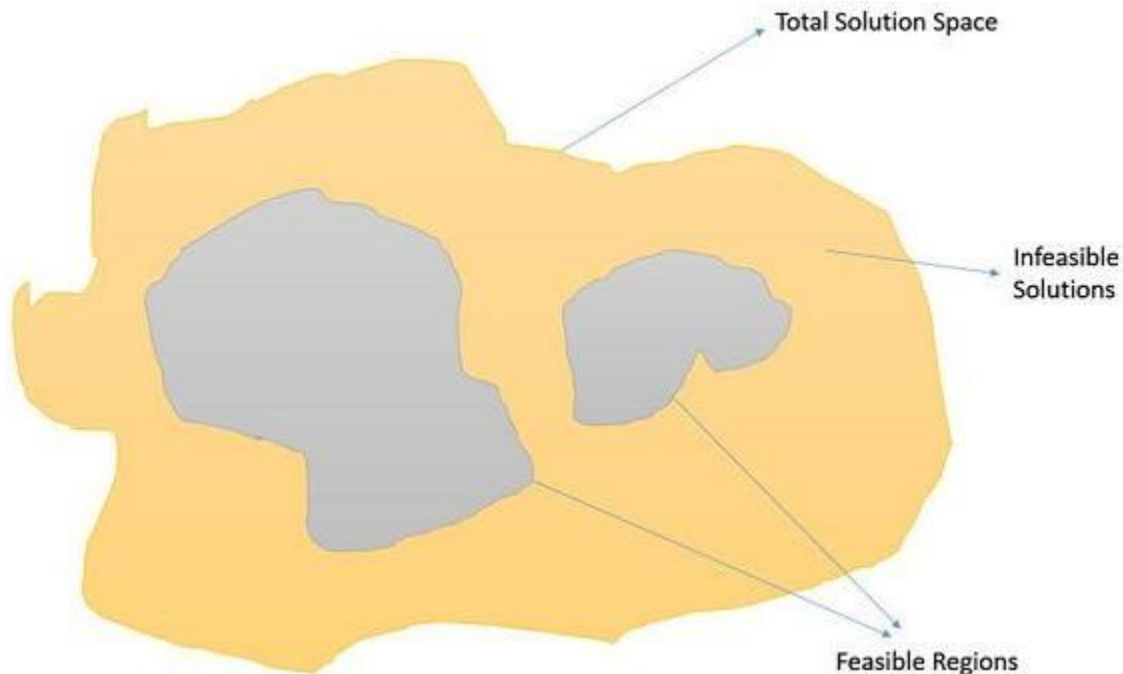
Variation of parameters and techniques

In genetic algorithms, there is no "one size fits all" or a magic formula which works for all problems. Even after the initial GA is ready, it takes a lot of time and effort to play around with the parameters like population size, mutation and crossover probability etc. to find the ones which suit the particular problem.

In this section, we introduce some advanced topics in Genetic Algorithms. A reader looking for just an introduction to GAs may choose to skip this section.

Constrained Optimization Problems

Constrained Optimization Problems are those optimization problems in which we have to maximize or minimize a given objective function value that is subject to certain constraints. Therefore, not all results in the solution space are feasible, and the solution space contains feasible regions as shown in the following image.



In such a scenario, crossover and mutation operators might give us solutions which are infeasible. Therefore, additional mechanisms have to be employed in the GA when dealing with constrained Optimization Problems.

Some of the most common methods are –

- Using **penalty functions** which reduces the fitness of infeasible solutions, preferably so that the fitness is reduced in proportion with the number of constraints violated or the distance from the feasible region.
- Using **repair functions** which take an infeasible solution and modify it so that the violated constraints get satisfied.
- **Not allowing infeasible solutions** to enter into the population at all.
- Use a **special representation or decoder functions** that ensures feasibility of the solutions.

Basic Theoretical Background

In this section, we will discuss about the Schema and NFL theorem along with the building block hypothesis.

Schema Theorem

Researchers have been trying to figure out the mathematics behind the working of genetic algorithms, and Holland's Schema Theorem is a step in that direction. Over the years various improvements and suggestions have been done to the Schema Theorem to make it more general.

In this section, we don't delve into the mathematics of the Schema Theorem, rather we try to develop a basic understanding of what the Schema Theorem is. The basic terminology to know are as follows –

- A **Schema** is a "template". Formally, it is a string over the alphabet =

$\{0,1,*\}$, where * is don't care and can take any value.

Therefore, $*10*1$ could mean 01001, 01011, 11001, or 11011

Geometrically, a schema is a hyper-plane in the solution search space.

- **Order** of a schema is the number of specified fixed positions in a gene.

Schema	Order
***	0
101	3
*11	2
1**	1

- **Defining length** is the distance between the two furthest fixed symbols in the gene.

Schema	Defining Length
****	0
11	1
1*0*	2
1111	3

The schema theorem states that this schema with above average fitness, short defining length and lower order is more likely to survive crossover and mutation.

Building Block Hypothesis

Building Blocks are low order, low defining length schemata with the above given average fitness. The building block hypothesis says that such building blocks serve as a foundation for the GAs success and adaptation in GAs as it progresses by successively identifying and recombining such “building blocks”.

No Free Lunch (NFL) Theorem

Wolpert and Macready in 1997 published a paper titled "No Free Lunch Theorems for Optimization." It essentially states that if we average over the space of all possible problems, then all non-revisiting black box algorithms will exhibit the same performance.

It means that the more we understand a problem, our GA becomes more problem specific and gives better performance, but it makes up for that by performing poorly for other problems.

GA Based Machine Learning

Genetic Algorithms also find application in Machine Learning. **Classifier systems** are a form of **genetics-based machine learning** (GBML) system that are frequently used in the field of machine learning. GBML methods are a niche approach to machine learning.

There are two categories of GBML systems –

- **The Pittsburgh Approach** – In this approach, one chromosome encoded one solution, and so fitness is assigned to solutions.
- **The Michigan Approach** – one solution is typically represented by many chromosomes and so fitness is assigned to partial solutions.

It should be kept in mind that the standard issue like crossover, mutation, Lamarckian or Darwinian, etc. are also present in the GBML systems.

Genetic Algorithms are primarily used in optimization problems of various kinds, but they are frequently used in other application areas as well.

In this section, we list some of the areas in which Genetic Algorithms are frequently used. These are —

- **Optimization** – Genetic Algorithms are most commonly used in optimization problems wherein we have to maximize or minimize a given objective function value under a given set of constraints. The approach to solve Optimization problems has been highlighted throughout the tutorial.
- **Economics** – GAs are also used to characterize various economic models like the cobweb model, game theory equilibrium resolution, asset pricing, etc.
- **Neural Networks** – GAs are also used to train neural networks, particularly recurrent neural networks.
- **Parallelization** – GAs also have very good parallel capabilities, and prove to be very effective means in solving certain problems, and also provide a good area for research.
- **Image Processing** – GAs are used for various digital image processing (DIP) tasks as well like dense pixel matching.
- **Vehicle routing problems** – With multiple soft time windows, multiple depots and a heterogeneous fleet.
- **Scheduling applications** – GAs are used to solve various scheduling problems as well, particularly the time tabling problem.
- **Machine Learning** – as already discussed, genetics based machine learning (GBML) is a niche area in machine learning.
- **Robot Trajectory Generation** – GAs have been used to plan the path which a robot arm takes by moving from one point to another.
- **Parametric Design of Aircraft** – GAs have been used to design aircrafts by varying the parameters and evolving better solutions.
- **DNA Analysis** – GAs have been used to determine the structure of DNA using spectrometric data about the sample.
- **Multimodal Optimization** – GAs are obviously very good approaches for multimodal optimization in which we have to find multiple optimum solutions.
- **Traveling salesman problem and its applications** – GAs have been used to solve the TSP, which is a well-known combinatorial problem using novel crossover and packing strategies.

References :

1. Jiawei, H., & Micheline, K. (2006). Data mining: concepts and techniques. Morgan kaufmann.
2. Jordan, M. I., & Mitchell, T. M. (2015). Machine learning: Trends, perspectives, and prospects. Science, 349(6245), 255-260.
3. <https://www.tutorialspoint.com/>
4. <https://www.geeksforgeeks.org/>

Unit-IV

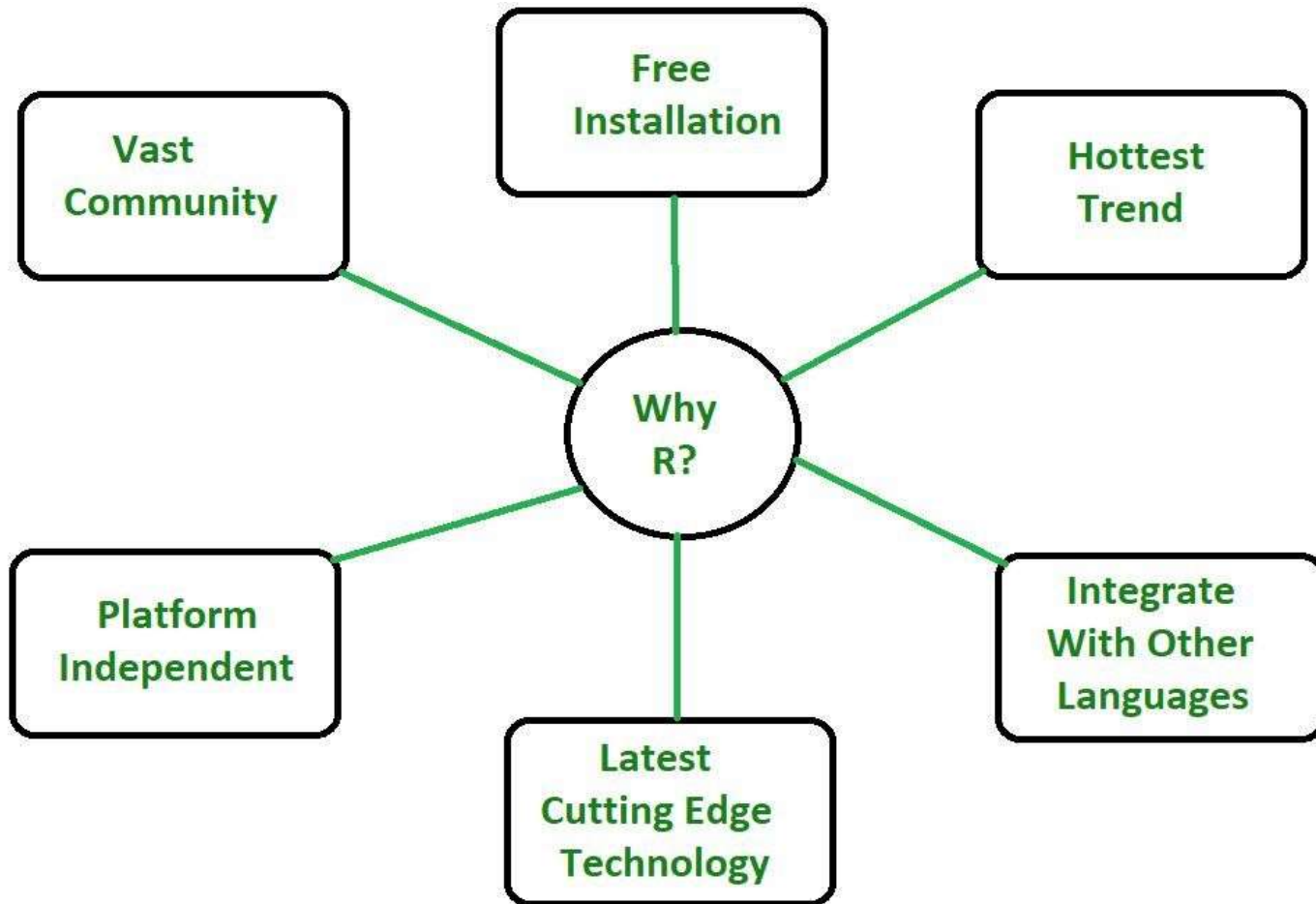
Content

- Introduction to R
- The S Philosophy
- Basic Features of R
- Free Software Design of the R
- System Limitations of R
- R Resources
- Installation and getting started with the R interface
- Data Frames
- The dplyr Package
- Lexical Scoping
- Dynamic Scoping

Introduction to R

- R programming is a leading tool for machine learning, statistics, and data analysis.
- It's a platform-independent language. This means it can be applied to all operating system.
- R programming language is not only a statistic package but also allows us to integrate with other languages (C, C++). Thus, you can easily interact with many data sources and statistical packages.
- The R programming language has a vast community of users and it's growing day by day.
- R is currently one of the most requested programming languages in the Data Science job market that makes it the hottest trend nowadays.

Why R Programming Language?



The S Philosophy

- S is a language that was developed by John Chambers and others at the old Bell Telephone Laboratories, originally part of AT&T Corp.
- S was initiated in 1976 as an internal statistical analysis environment—originally implemented as Fortran libraries.
- Early versions of the language did not even contain functions for statistical modelling.
- In particular, it's important to realize that the S language had its roots in data analysis, and did not come from a traditional programming language background.
- Its inventors were focused on figuring out how to make data analysis easier, first for themselves, and then eventually for others.
- The R language came to use quite a bit after S had been developed. One key limitation of the S language was that it was only available in a commercial package, S-PLUS.

Basic Features of R

- Statistical Features of R
 - Basic Statistics
 - Static graphics
 - Probability distributions
 - Data analysis
- Programming Features of R
 - R Packages
 - Distributed Computing

Advantages of R

- Open Source
- Platform Independent
- Machine Learning Operations
- Exemplary support for data wrangling
- Quality plotting and graphing
- The array of packages
- Statistics
- Continuously Growing

Applications of R

- We use R for Data Science. It gives us a broad variety of libraries related to statistics. It also provides the environment for statistical computing and design.
- R is used by many quantitative analysts as its programming tool. Thus, it helps in data importing and cleaning.
- R is the most prevalent language. So many data analysts and research programmers use it. Hence, it is used as a fundamental tool for finance.
- Tech giants like Google, Facebook, bing, Twitter, Accenture, Wipro and many more using R nowadays.

Free Software Design of the R

- R is available as Free Software under the terms of the Free Software Foundation's GNU General Public License in source code form.
- It compiles and runs on a wide variety of UNIX platforms and similar systems (including FreeBSD and Linux), Windows and MacOS.

System Limitations of R

- Data Handling
- Basic Security
- Complicated Language
- Weak Origin
- Lesser Speed

R Resources

- Useful Video Courses
 - R Programming Online Training
 - Data Visualization Using R Programming
 - Data Analytics Using R Programming
 - R Programming Language
- Useful Books
 - Kun Ren, “Learning R Programming”, Packt Publishing.
 - Jaynal Abedin , “Modern R Programming Cookbook”, Packt Publishing.
 - R Programming for Data Science, by Roger D. Peng.
 - Using R for Introductory Statistics, by John Verzani, Chapman & Hall/CRC, 2004, ISBN 1584884509.
 - Advanced R, by Hadley Wickham, ISBN 9781466586963.

Installation and getting started with the R interface

- Install R- R is available to download from the official R website. Look for this section of the web page:

The Comprehensive R Archive Network

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

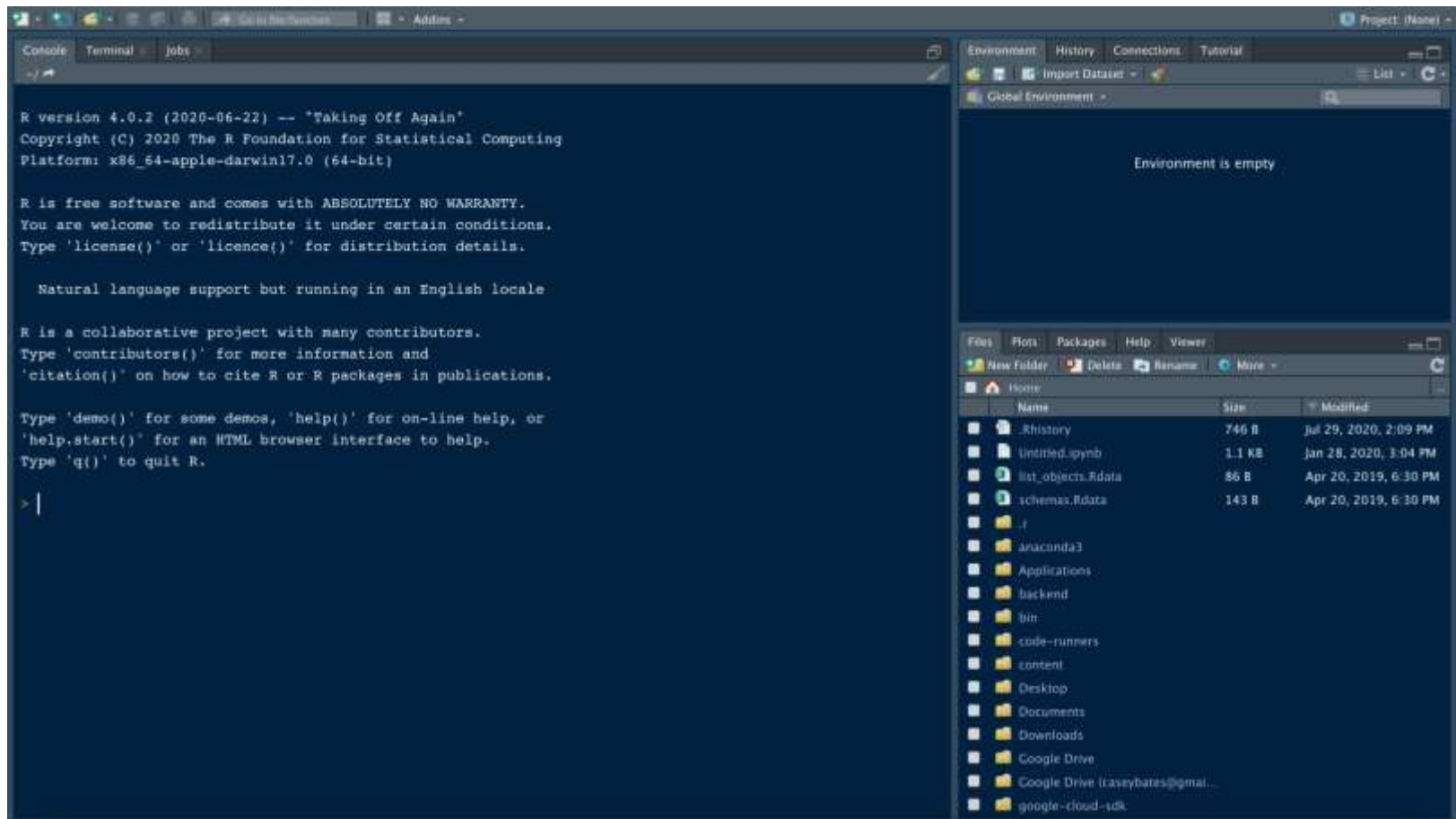
- [Download R for Linux](#)
- [Download R for \(Mac\) OS X](#)
- [Download R for Windows](#)

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

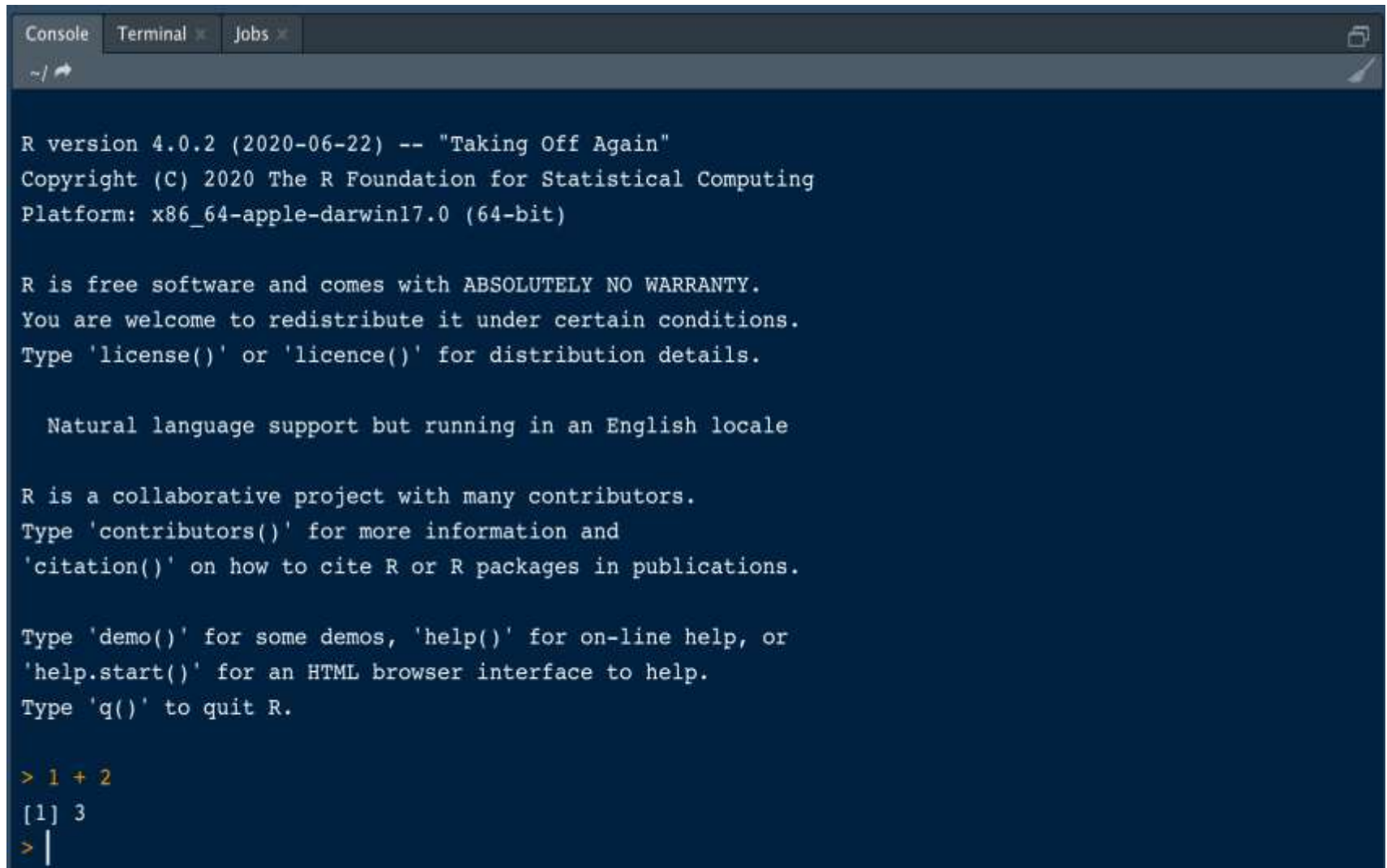
- Install Rstudio- Now that R is installed, we can install RStudio.

	RStudio Desktop Open Source License Free Learn more	RStudio Desktop Commercial License \$995 /year Learn more	RStudio Server Open Source License Free Learn more	RStudio Server Pro Commercial License \$4,975 /year (5 Named Users) Evaluation Learn more
Integrated Tools for R	✓	✓	✓	✓
Priority Support		✓		✓
Access via Web Browser			✓	✓
Enterprise Security				✓
Project Sharing				✓
Manage Multiple R Sessions & Versions				✓
Admin Dashboard				✓
Load Balancing				✓
Auditing and Monitoring				✓
Data Connectivity				✓
Launcher				✓
Tutorial API				✓
License	AGPL	Commercial	AGPL	Commercial

- First Look at Rstudio- When we open RStudio for the first time, we'll probably see a layout like this:



- The Console-



```
Console Terminal x Jobs x
~/ ↩

R version 4.0.2 (2020-06-22) -- "Taking Off Again"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin17.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

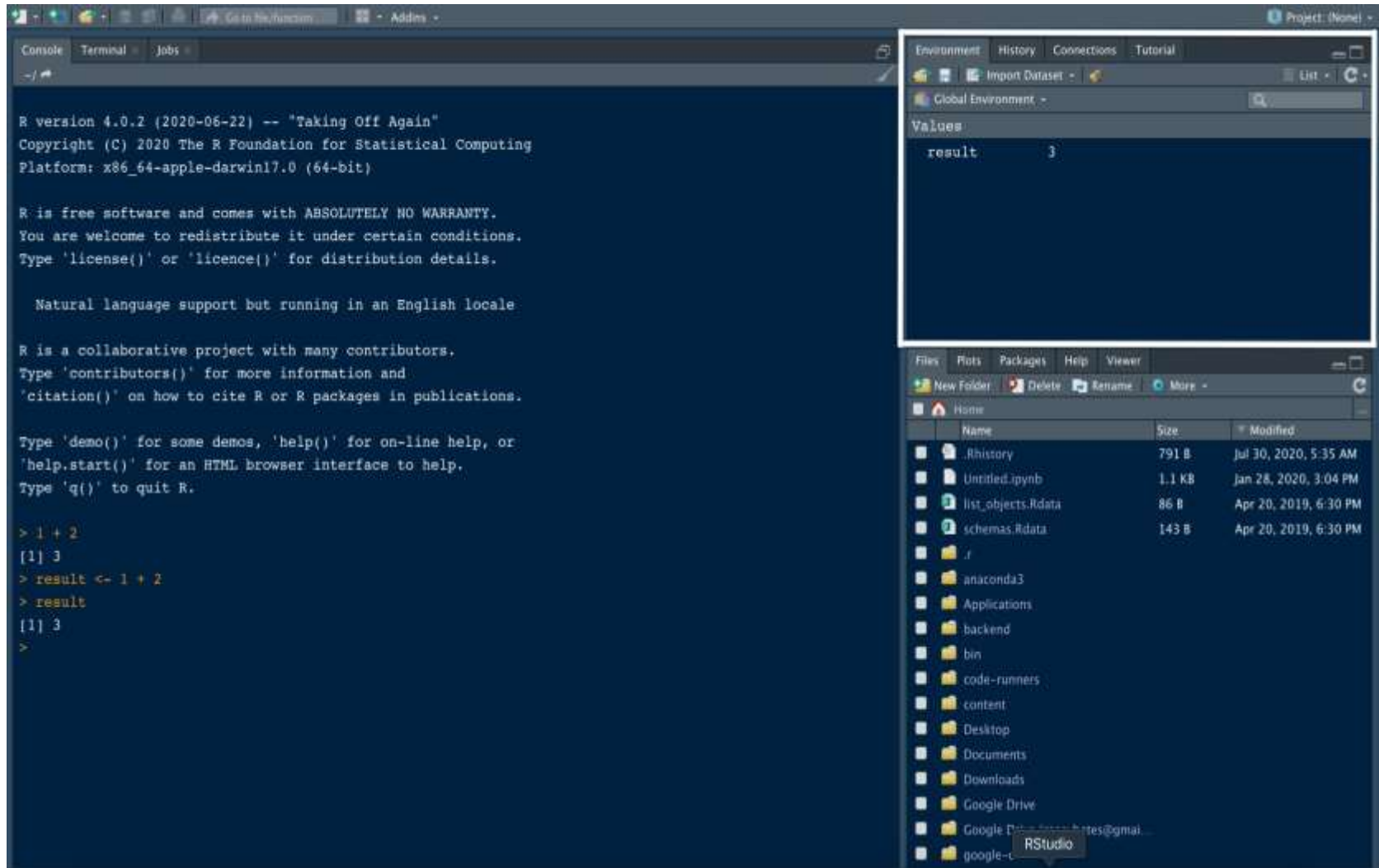
Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 1 + 2
[1] 3
> |
```

- The Global Environment-



Data Frames

- Data Frames are data displayed in a format as a table.
- Data Frames can have different types of data inside it. While the first column can be character, the second and third can be numeric or logical. However, each column should have the same type of data.
- Use the `data.frame()` function to create a data frame.

Example

```
# Create a data frame
Data_Frame <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)

# Print the data frame
Data_Frame
```

Summarize the Data: Use the `summary()` function to summarize the data from a Data Frame:

Example

```
Data_Frame <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)

Data_Frame

summary(Data_Frame)
```

Access Items: We can use single brackets `[ ]`, double brackets `[[ ]]` or `$` to access columns from a data frame:

Example

```
Data_Frame <- data.frame (  
  Training = c("Strength", "Stamina", "Other"),  
  Pulse = c(100, 150, 120),  
  Duration = c(60, 30, 45)  
)
```

```
Data_Frame[1]
```

```
Data_Frame[["Training"]]
```

```
Data_Frame$Training
```

Add Rows: Use the `rbind()` function to add new rows in a Data Frame:

Example

```
Data_Frame <- data.frame (  
  Training = c("Strength", "Stamina", "Other"),  
  Pulse = c(100, 150, 120),  
  Duration = c(60, 30, 45)  
)  
  
# Add a new row  
New_row_DF <- rbind(Data_Frame, c("Strength", 110, 110))  
  
# Print the new row  
New_row_DF
```

Add Columns: Use the `cbind()` function to add new columns in a Data Frame:

Example

```
Data_Frame <- data.frame (  
  Training = c("Strength", "Stamina", "Other"),  
  Pulse = c(100, 150, 120),  
  Duration = c(60, 30, 45)  
)  
  
# Add a new column  
New_col_DF <- cbind(Data_Frame, Steps = c(1000, 6000, 2000))  
  
# Print the new column  
New_col_DF
```

Remove Rows and Columns: Use the `c()` function to remove rows and columns in a Data Frame:

Example

```
Data_Frame <- data.frame (  
  Training = c("Strength", "Stamina", "Other"),  
  Pulse = c(100, 150, 120),  
  Duration = c(60, 30, 45)  
)  
  
# Remove the first row and column  
Data_Frame_New <- Data_Frame[-c(1), -c(1)]  
  
# Print the new data frame  
Data_Frame_New
```

Combining Data Frames: Use the `rbind()` function to combine two or more data frames in R vertically:

Example

```
Data_Frame1 <- data.frame (  
  Training = c("Strength", "Stamina", "Other"),  
  Pulse = c(100, 150, 120),  
  Duration = c(60, 30, 45)  
)  
  
Data_Frame2 <- data.frame (  
  Training = c("Stamina", "Stamina", "Strength"),  
  Pulse = c(140, 150, 160),  
  Duration = c(30, 30, 20)  
)  
  
New_Data_Frame <- rbind(Data_Frame1, Data_Frame2)  
New_Data_Frame
```

And use the `cbind()` function to combine two or more data frames in R horizontally:

Example

```
Data_Frame3 <- data.frame (  
  Training = c("Strength", "Stamina", "Other"),  
  Pulse = c(100, 150, 120),  
  Duration = c(60, 30, 45)  
)  
  
Data_Frame4 <- data.frame (  
  Steps = c(3000, 6000, 2000),  
  Calories = c(300, 400, 300)  
)  
  
New_Data_Frame1 <- cbind(Data_Frame3, Data_Frame4)  
New_Data_Frame1
```

The dplyr Package

- The dplyr package in R Programming Language is a structure of data manipulation that provides a uniform set of verbs, helping to resolve the most frequent data manipulation hurdles.
- The dplyr Package in R performs the steps given below quicker and in an easier fashion:
 - By limiting the choices the focus can now be more on data manipulation difficulties.
 - There are uncomplicated “verbs”, functions present for tackling every common data manipulation and the thoughts can be translated into code faster.
 - There are valuable back-ends and hence waiting time for the computer reduces.

Data Manipulation in R With dplyr Package

- There are different ways to perform data manipulation in R, such as using Base R functions like `subset()`, `with()`, `within()`, etc., Packages like `data.table`, `ggplot2`, `reshape2`, `readr`, etc., and different Machine Learning algorithms.
- The `dplyr` package consists of many functions specifically used for data manipulation. These functions process data faster than Base R functions and are known the best for data exploration and transformation, as well.

Important Functions in the dplyr Package

- **select()** :- To select columns (variables)
- **filter()** :-To filter (subset) rows.
- **mutate()** :-To create new variables
- **summarise()** :- To summarize (or aggregate) data
- **group_by()** :- To group data
- **arrange()** :- To sort data
- **join()** :- To join data frames.

Installing the dplyr Package

- To install the dplyr package, run the following command:

```
install.packages("dplyr")
```

- For example, we are going to use the iris dataset from the datasets package in R programming that can be loaded as follows:

```
#To load dplyr package  
library("dplyr")  
#To load datasets package  
library("datasets")  
#To load iris dataset  
data(iris)  
summary(iris)
```

Select()

It is used to select data by its column name. We can select any number of columns in a number of ways.

For example:

```
#To select the following columns
selected <- select(iris, Sepal.Length, Sepal.Width, Petal.Length)
head(selected)

#To select all columns from Sepal.Length to Petal.Length
selected1 <- select(iris, Sepal.Length:Petal.Length)
#To print first four rows
head(selected1, 4)

#To select columns with numeric indexes
selected1 <- select(iris, c(3:5))
head(selected1)
```

```
#We use(-)to hide a particular column
selected <- select(iris, -Sepal.Length, -Sepal.Width)
head(selected)
```

Filter()

It is used to find rows with matching criteria. It also works like the select() function, i.e., we pass a data frame along with a condition separated by a comma. For example:

```
#To select the first 3 rows with Species as setosa  
filtered <- filter(iris, Species == "setosa" )  
head(filtered,3)
```

```
#To select the last 5 rows with Species as versicolor and Sepal width more than 3  
filtered1 <- filter(iris, Species == "versicolor", Sepal.Width > 3)  
tail(filtered1)
```

Mutate()

It creates new columns and preserves the existing columns in a dataset.
For example:

```
#To create a column "Greater.Half" which stores TRUE if given condition  
is TRUE  
col1 <- mutate(iris, Greater.Half = Sepal.Width > 0.5 * Sepal.Length)  
tail(col1)
```

```
#To check how many flowers satisfy this condition  
table(col1$Greater.Half)
```

Arrange()

It is used to sort rows by variables in both an ascending and descending order. For example:

```
#To arrange Sepal Width in ascending order  
arranged <- arrange(col1, Sepal.Width)  
head(arranged)  
#To arrange Sepal Width in descending order  
arranged <- arrange(col1, desc(Sepal.Width))  
head(arranged)
```

Summarise()

It is used to find insights(mean, median, mode, etc.) from a dataset. It reduces multiple values down to a single value. For example:

```
summarised <- summarise(arranged, Mean.Width = mean(Sepal.Width))  
head(summarised)
```

Grouping()

It is done to group observations within a dataset by one or more variables. Most data operations are performed on groups defined by variables. For example:

```
#To find mean sepal width by Species, we use grouping as follows  
gp <- group_by(iris,Species)  
mn <- summarise(gp,Mean.Sepal = mean(Sepal.Width))  
head(mn)
```

Scoping Rules

- Related to the scoping rules is how R uses the search list to bind a value to a symbol.
- Consider the following function:

```
f <- function(x, y) {  
  x^2 + y / z  
}
```

This function has 2 formal arguments `x` and `y`. In the body of the function there is another symbol, `z`. In this case `z` is called a **free variable**.

The scoping rules of a language determine how values are assigned to free variables. Free variables are not formal arguments and are not local variables (assigned inside the function body).

R uses **lexical scoping** = **static scoping**. An alternative to lexical scoping is *dynamic scoping* which is implemented by some languages. Lexical scoping turns out to be particularly useful for simplifying statistical computations.

Lexical Scoping

- Scoping Rules of a language are responsible for determining how value will be associated with the free variable in a function in the R language.
- In Lexical Scoping the scope of the variable is determined by the textual structure of a program.
- Most programming languages we use today are lexically scoped.
- Even, a human can determine the scope of a variable just by properly reading the code.

Below is a code of lexical Scoping in R. In this example, first, a mapping for a is created. In the next line a function b is defined which returns the value of some a. On line 3, we define a function c, which creates a new mapping for a and then calls b. Note that the assignment on line 4 does not update the definition on line 1 and in the Output value, 1 is returned.

```
# R program to depict scoping

# Assign a value to a
a <- 1

# Defining function b and c
b <- function() a
c <- function(){
  a <- 2
  b()
}

# Call to function c
c()
```

Output: 1

Dynamic Scoping

- In Dynamic scoping, the variable takes the value of the most latest value assigned to that variable.
- Here, Considering the same above example.
- The output of this code should be 2 according to the rules of Dynamic as the latest value assigned to the variable a is 2.

Difference Between Lexical and Dynamic Scoping

S. No.	Lexical	Dynamic
1	In this variable refers to top level environment	In this variable is associated to most recent environment
2	It is easy to find the scope by reading the code	In this programmer has to anticipate all possible contexts
3	It is dependent on how code is written	It is dependent on how code is executed
4	Structure of program defines which variable is referred to.	Runtime state of program stack determines the variable.
5	It is property of program text and unrelated to real time stack.	It is dependent on real time stack rather than program text
6	It provides less flexibility	It provides more flexibility
7	Accessing to nonlocal variables in lexical scoping is fast.	Accessing to nonlocal variables in dynamic scoping takes more time.
8	Local variables can be protected to be accessed by local variables.	There is no way to protect local variables to be accessed by subprograms.

References

- R Programming for Data Science, by Roger D. Peng.
- R for Introductory Statistics, by John Verzani, Chapman & Hall/CRC, 2004, ISBN 1584884509.
- Advanced R, by Hadley Wickham, ISBN 9781466586963.
- <https://nptel.ac.in/courses/110/107/110107092/>
- <https://www.youtube.com/watch?v=J326LIUrZM8>
- <https://www.geeksforgeeks.org/>
- <https://www.w3schools.com/>

Thank-you

UNIT-V

Introduction to Data Warehouse:

A data warehouse is a subject-oriented, integrated, time-variant and non-volatile collection of data in support of management's decision making process. **Subject-Oriented:** A data warehouse can be used to analyze a particular subject area. For example, "sales" can be a particular subject.

Integrated: A data warehouse integrates data from multiple data sources. For example, source A and source B may have different ways of identifying a product, but in a data warehouse, there will be only a single way of identifying a product.

Time-Variant: Historical data is kept in a data warehouse. For example, one can retrieve data from 3 months, 6 months, 12 months, or even older data from a data warehouse. This contrasts with a transactions system, where often only the most recent data is kept. For example, a transaction system may hold the most recent address of a customer, where a data warehouse can hold all addresses associated with a customer.

Non-volatile: Once data is in the data warehouse, it will not change. So, historical data in a data warehouse should never be altered.

Data Warehouse Design Process:

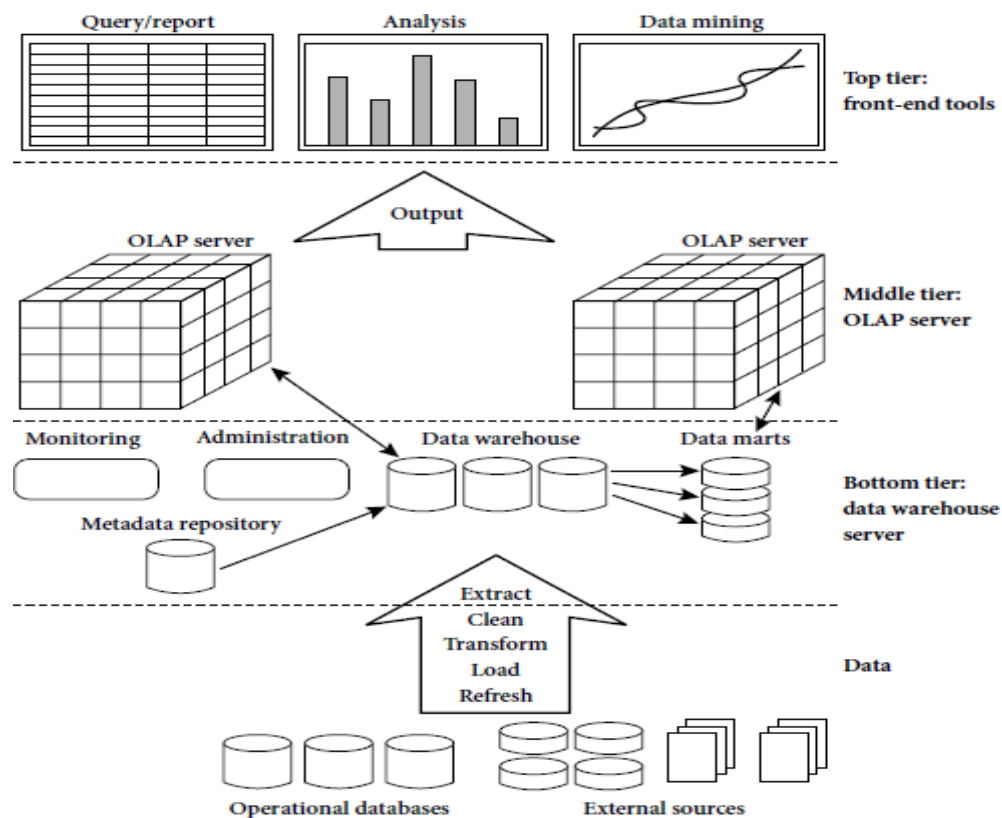
A data warehouse can be built using a top-down approach, a bottom-up approach, or a combination of both.

- The top-down approach starts with the overall design and planning. It is useful in cases where the technology is mature and well known, and where the business problems that must be solved are clear and well understood.
- The bottom-up approach starts with experiments and prototypes. This is useful in the early stage of business modeling and technology development. It allows an organization to move forward at considerably less expense and to evaluate the benefits of the technology before making significant commitments.
- In the combined approach, an organization can exploit the planned and strategic nature of the top-down approach while retaining the rapid implementation and opportunistic application of the bottom-up approach.

The warehouse design process consists of the following steps:

- Choose a business process to model, for example, orders, invoices, shipments, inventory, account administration, sales, or the general ledger. If the business process is organizational and involves multiple complex object collections, a data warehouse model should be followed. However, if the process is departmental and focuses on the analysis of one kind of business process, a data mart model should be chosen.
- Choose the grain of the business process. The grain is the fundamental, atomic level of data to be represented in the fact table for this process, for example, individual transactions, individual daily snapshots, and so on.
- Choose the dimensions that will apply to each fact table record. Typical dimensions are time, item, customer, supplier, warehouse, transaction type, and status.
- Choose the measures that will populate each fact table record. Typical measures are numeric additive quantities like dollars sold and units sold.

A Three Tier Data Warehouse Architecture:



Tier-1:

The bottom tier is a warehouse database server that is almost always a relational database system. Back-end tools and utilities are used to feed data into the bottom tier from operational databases or other external sources (such as customer profile information provided by external consultants). These tools and utilities perform data extraction, cleaning, and transformation (e.g., to merge similar data from different sources into a unified format), as well as load and refresh functions to update the data warehouse. The data are extracted using application program interfaces known as gateways. A gateway is supported by the underlying DBMS and allows client programs to generate SQL code to be executed at a server. Examples of gateways include ODBC (Open Database Connection) and OLEDB (Open Linking and Embedding for Databases) by Microsoft and JDBC (Java Database Connection). This tier also contains a metadata repository, which stores information about the data warehouse and its contents.

Tier-2:

The middle tier is an OLAP server that is typically implemented using either a relational OLAP (ROLAP) model or a multidimensional OLAP.

- OLAP model is an extended relational DBMS that maps operations on multidimensional data to standard relational operations.
- A multidimensional OLAP (MOLAP) model, that is, a special-purpose server that directly implements multidimensional data and operations.

Tier-3:

The top tier is a front-end client layer, which contains query and reporting tools, analysis tools, and/or data mining tools (e.g., trend analysis, prediction, and so on).

Data Warehouse Models:

There are three data warehouse models.

1. Enterprise warehouse:

- An enterprise warehouse collects all of the information about subjects spanning the entire organization.
- It provides corporate-wide data integration, usually from one or more operational systems or external information providers, and is cross-functional in scope.
- It typically contains detailed data as well as summarized data, and can range in size from

A few gigabytes to hundreds of gigabytes, terabytes, or beyond.

- An enterprise data warehouse may be implemented on traditional mainframes, computer super servers, or parallel architecture platforms. It requires extensive business modeling and may take years to design and build.

2. Data mart:

- A data mart contains a subset of corporate-wide data that is of value to a specific group of users. The scope is confined to specific selected subjects. For example, a marketing data mart may confine its subjects to customer, item, and sales. The data contained in data marts tend to be summarized.
- Data marts are usually implemented on low-cost departmental servers that are UNIX/LINUX- or Windows-based. The implementation cycle of a data mart is more likely to be measured in weeks rather than months or years. However, it may involve complex integration in the long run if its design and planning were not enterprise-wide.
- Depending on the source of data, data marts can be categorized as independent or more dependent. Independent data marts are sourced from data captured from one or more operational systems or external information providers, or from data generated locally within a particular department or geographic area. Dependent data marts are sourced directly from enterprise data warehouses.

3. Virtual warehouse:

- A virtual warehouse is a set of views over operational databases. For efficient query processing, only some of the possible summary views may be materialized.
- A virtual warehouse is easy to build but requires excess capacity on operational database servers.

Meta Data Repository:

Metadata are data about data. When used in a data warehouse, metadata are the data that define warehouse objects. Metadata are created for the data names and definitions of the given warehouse. Additional metadata are created and captured for time stamping any extracted data, the source of the extracted data, and missing fields that have been added by data cleaning or integration processes.

A meta data repository should contain the following:

- A description of the structure of the data warehouse, which includes the warehouse schema, view, dimensions, hierarchies, and derived data definitions, as well as data mart locations and contents.
 - Operational metadata, which include data lineage (history of migrated data and the sequence of transformations applied to it), currency of data (active, archived, or purged), and monitoring information (warehouse usage statistics, error reports, and audit trails).
 - The algorithms used for summarization, which include measure and dimension definition algorithms, data on granularity, partitions, subject areas, aggregation, summarization, and predefined queries and reports.
 - The mapping from the operational environment to the data warehouse, which includes source data bases and their contents, gate way descriptions, data partitions, data extraction, cleaning, transformation rules and defaults, data refresh and purging rules, and security (user authorization and access control).
 - Data related to system performance, which include indices and profiles that improve data access and retrieval performance, in addition to rules for the timing and scheduling of refresh, update, and replication cycles.
 - Business metadata, which include business terms and definitions, data ownership information, and charging policies.
-

Schema Design:

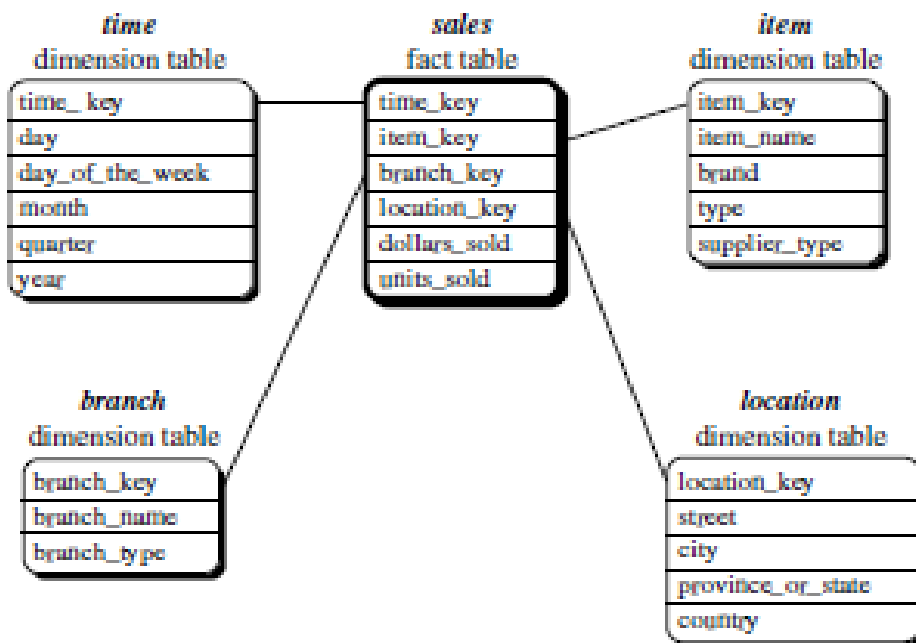
Stars, Snowflakes, and Fact Constellations: Schemas for Multidimensional Databases The entity-relationship data model is commonly used in the design of relational databases, where a database schema consists of a set of entities and the relationships between them. Such a data model is appropriate for on-line transaction processing. A data warehouse, however, requires a concise, subject-oriented schema that facilitates on-line data analysis. The most popular data model for a data warehouse is a multidimensional model. Such a model can exist in the form of a star schema, a snowflake schema, or a fact constellation schema. Let's look at each of the schema types. **Star schema:** The most common modeling paradigm is the star schema, in which the data warehouse contains (1) a large central table (fact table) containing the bulk of the data, with no redundancy, and (2) a set of smaller attendant tables (dimension tables), one for each dimension. The schema graph resembles a star burst, with the dimension tables displayed in a radial pattern around the central fact table.

Star schema:

A star schema for All Electronics sales is shown in Figure. Sales are considered along four dimensions, namely, time, item, branch, and location. The schema contains a central fact table for sales that contains keys to each of the four dimensions, along with two measures: dollars sold and units sold. To minimize the size of the fact table, dimension identifiers (such as time key and item key) are system-generated identifiers. Notice that in the star schema, each dimension is represented by only one table, and each table contains a set of attributes. For example, the location dimension table contains the attribute set

{locationkey, street, city, province or state, country}. This constraint may introduce some redundancy.

For example, "Vancouver" and "Victoria" are both cities in the Canadian province of British Columbia. Entries for such cities in the location dimension table will create redundancy among the attributes province or state and country, that is, (... , Vancouver, British Columbia, Canada) and (... , Victoria, British Columbia, Canada). Moreover, the attributes within a dimension table may form either a hierarchy (total order) or a lattice (partial order).

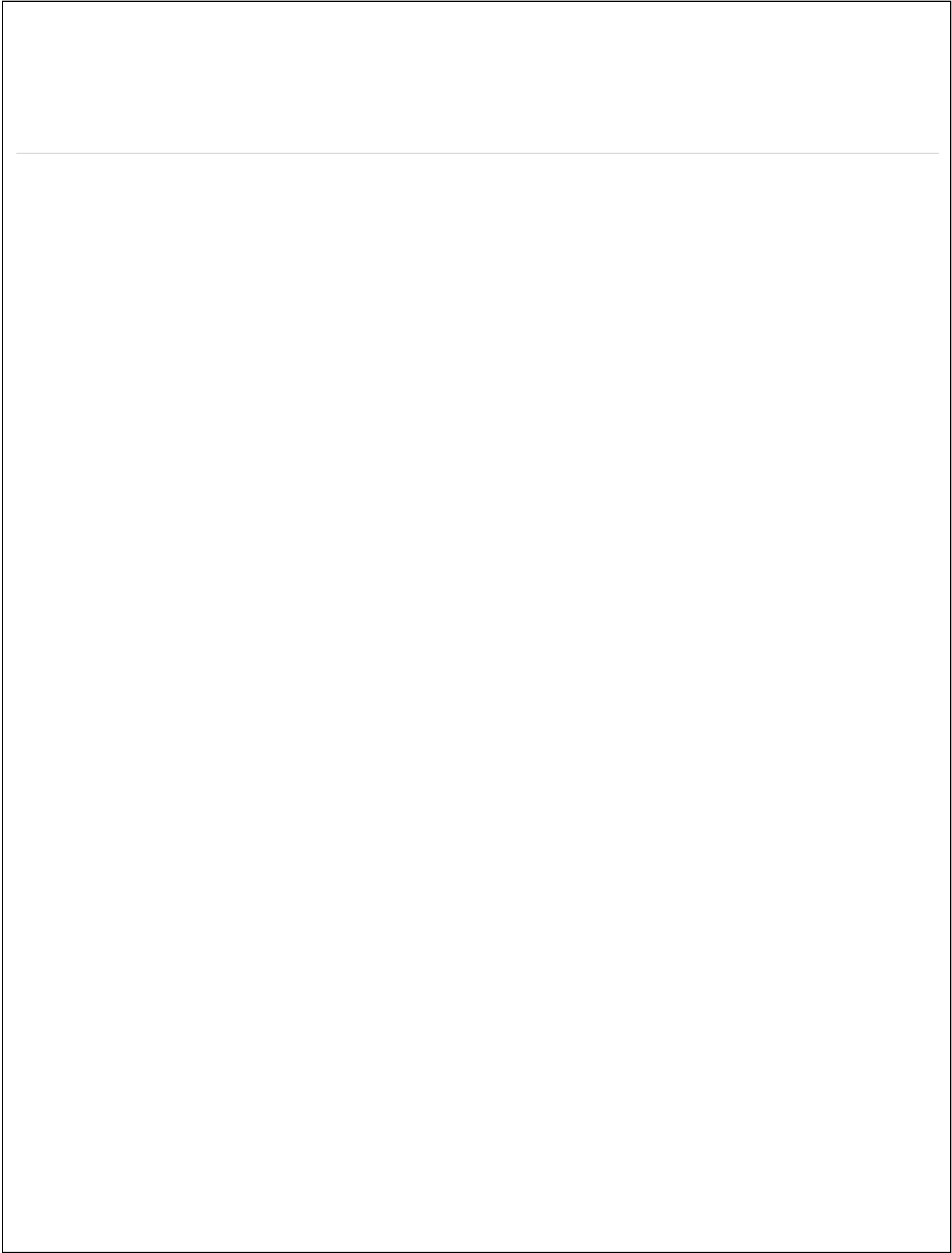


Star schema of a data warehouse for sales.

Snowflake schema.:

A snowflake schema for AllElectronics sales is given in Figure Here, the sales fact table is identical to that of the star schema in Figure . The main difference between the two schemas is in the definition of dimension tables.

The single dimension table for item in the star schema is normalized in the snowflake schema, resulting in new item and supplier tables. For example, the itemdimension table now contains the attributes itemkey, itemname, brand,type,andsupplierkey,wheresupplierkeyislinkedtothesupplierdimensiontable,containingssupplierkey and supplier type information. Similarly, the single dimension table for location in the star schema can be normalizedintotwonewtables:locationandcity.Thecitykeyinthenewlocationtablelinkstothe citydimension. Notice that further normalization can be performed on province or state and country in the snowflake schema



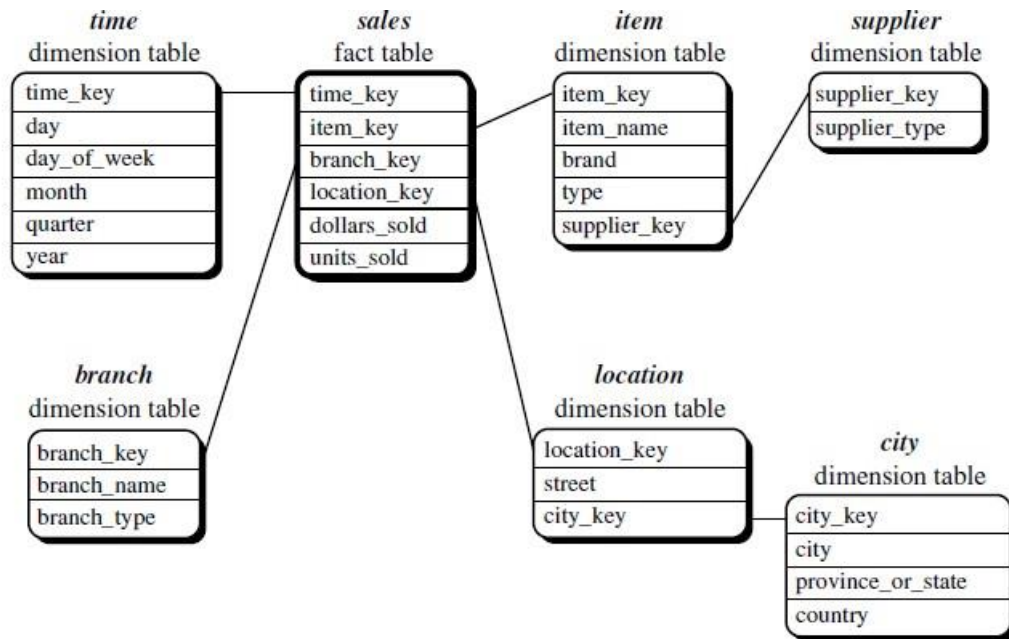


Figure 10.1: Snowflake schema of a data warehouse for sales.

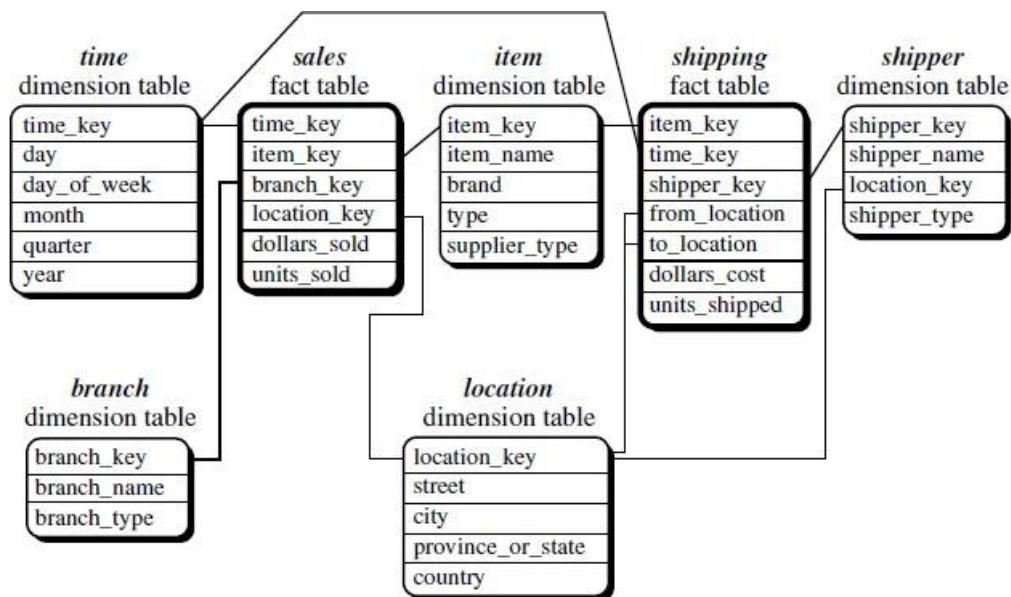
Fact constellation.

A fact constellation schema is shown in Figure. This schema specifies two fact tables, sales and shipping. The sales table definition is identical to that of the star schema. The shipping table has five dimensions, or keys: item key, time key, shipper key, from location, and to location, and two measures: dollars cost and units shipped.

A fact constellation schema allows dimension table to be shared between fact tables. For example, the dimension tables for time, item, and location are shared between both the sales and shipping fact tables.

In data warehousing, there is a distinction between a data warehouse and a datamart.

A data warehouse collects information about subjects that span the entire organization, such as customers, items, sales, assets, and personnel, and thus its scope is enterprise-wide. For data warehouses, the fact constellation schema is commonly used, since it can model multiple, interrelated subjects. A data mart, on the other hand, is a department subset of the data warehouse that focuses on selected subjects, and thus its scope is department wide. For data marts, the star or snowflake schema are commonly used, since both are geared toward modeling single subjects, although the star schema is more popular and efficient.



Fact constellation schema of a data warehouse for sales and shipping.

Measures: Their Categorization and Computation:

A measure value is computed for a given point by aggregating the data corresponding to the respective dimension-value pairs defining the given point.

Measures can be organized into three categories(i.e., distributive,algebraic,holistic), based on the kind of aggregate functions used.

Distributive: An aggregate function is distributive if it can be computed in a distributed manner as follows. Suppose the data are partitioned into sets. We apply the function to each partition, resulting in aggregate values. If the result derived by applying the function to then aggregatevaluesisthesameasthat derived by applying the function to the entire data set (without partitioning), the function can be computed in

A distributed manner. Forexample,count()canbe computed for a data cube byfirst partitioning the cube into a set of subcubes, computing count() for each subcube, and then summing up the counts obtained for each

subcube. Hence, count() is a distributive aggregate function. For the same reason, sum(), min(), and max() are distributive aggregate functions. A measure is distributive if it is obtained by applying a distributive aggregate function. Distributive measures can be computed efficiently because they can be computed in a distributive manner.

OLAP(Online analytical Processing):

- OLAP is an approach to answering multi-dimensional analytical (MDA) queries swiftly.
- OLAP is part of the broader category of business intelligence, which also encompasses relational database, report writing and data mining.
- OLAP tools enable users to analyze multidimensional data interactively from multiple perspectives.

OLAP consists of three basic analytical operations:

- Consolidation (Roll-Up)
 - Drill-Down
 - Slicing And Dicing
- Consolidation involves the aggregation of data that can be accumulated and computed in one or more dimensions. For example, all sales offices are rolled up to the sales Department to sales division to anticipate sales trends.
 - The drill-down is a technique that allows users to navigate through the details. For instance, users can view the sales by individual products that make up a region's sales.
 - Slicing and dicing is a feature where by users can take out (slicing) a specific set of data of the OLAP cube and view (dicing) the slices from different viewpoints.

Types of OLAP:

1. Relational OLAP (ROLAP):

• ROLAP works directly with relational databases. The base data and the dimension tables are stored as relational tables and new tables are created to hold the aggregated information. It depends on a specialized schema design.

- This methodology relies on manipulating the data stored in the relational database to
-

give the appearance of traditional OLAP's slicing and dicing functionality. In essence, each action of slicing and dicing is equivalent to adding a "WHERE" clause in the SQL statement. ROLAP tools do not use pre-calculated data cubes but instead pose the query to the standard relational database and it strables in order to bring back the data required to answer the question.

- ROLAP tools feature the ability to ask any question because the methodology does not limit to the contents of a cube. ROLAP also has the ability to drill down to the lowest level of detail in the database.

2. Multidimensional OLAP (MOLAP):

- MOLAP is the 'classic' form of OLAP and is sometimes referred to as just OLAP.
- MOLAP stores this data in an optimized multi-dimensional array storage, rather than in a relational database. Therefore it requires the pre-computation and storage of information in the cube - the operation known as processing.
- MOLAP tools generally utilize a pre-calculated data set referred to as a data cube. The data cube contains all the possible answers to a given range of questions.
- MOLAP tools have a very fast response time and the ability to quickly write back data into the data set.

3. Hybrid OLAP (HOLAP):

- There is no clear agreement across the industry as to what constitutes Hybrid OLAP, except that a database will divide data between relational and specialized storage.
 - For example, for some vendors, a HOLAP database will use relational tables to hold the larger quantities of detailed data, and use specialized storage for at least some aspects of the smaller quantities of more-aggregate or less-detailed data.
 - HOLAP addresses the shortcomings of MOLAP and ROLAP by combining the capabilities of both approaches.
 - HOLAP tools can utilize both pre-calculated cubes and relational data sources.
-

