Data Mining

Introduction

- Rapid advances in data collection and storage technology have enabled or- ganizations to accumulate vast amounts of data.
- Data mining is a technology that blends traditional data analysis methods with sophisticated algorithms for processing large volumes of data.
- Data mining is the process of automatically discovering useful information in large data repositories.
- Looking up individual records using a database management system or finding particular Web pages via a query to an Internet search engine are tasks related to the area of **information retrieval.**

.

Data Mining and Knowledge Discovery

- Data mining is an integral part of **knowledge discovery in databases** (KDD), which is the overall process of converting raw data into useful information.

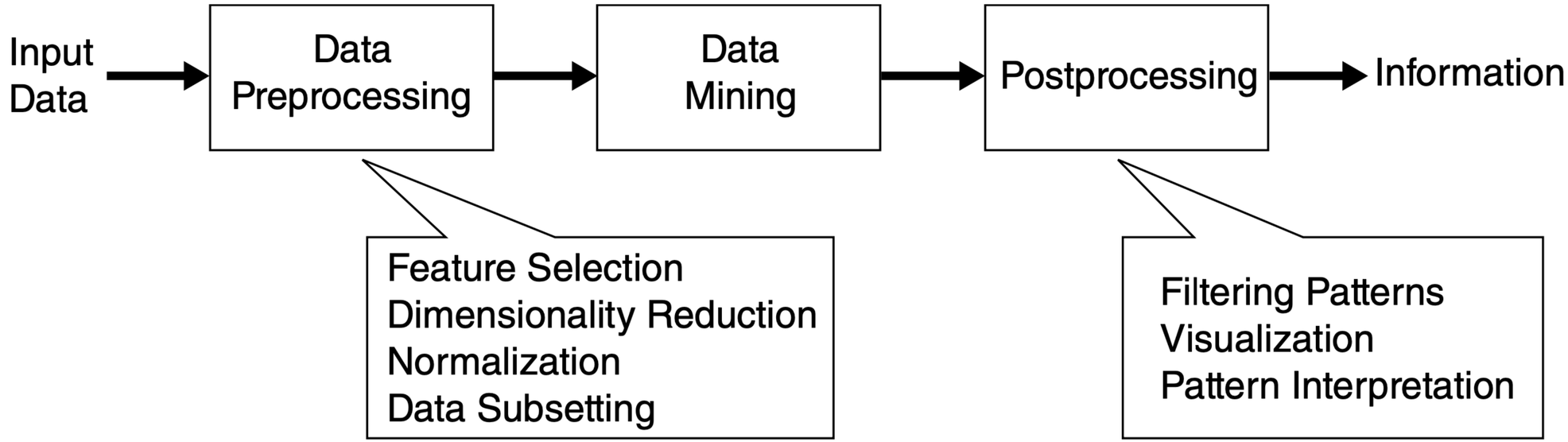


Figure 1.1. The process of knowledge discovery in databases (KDD).

- The **input data** can be stored in a variety of formats (flat files, spread- sheets, or relational tables) and may reside in a centralized data repository or be distributed across multiple sites.
- The purpose of **preprocessing** is to transform the raw input data into an appropriate format for subsequent analysis.
- The steps involved in data preprocessing include fusing data from multiple sources, cleaning data to remove noise and duplicate observations, and selecting records and features that are relevant to the data mining task at hand.
- **Postprocessing** step that ensures that only valid and useful results are incorporated into the decision support system.
- An example of postprocessing is visualization, which allows analysts to explore the data and the data mining results from a variety of viewpoints.
- Statistical measures or hypoth- esis testing methods can also be applied during postprocessing to eliminate spurious data mining results.

Challenges

1. **Scalability** Because of advances in data generation and collection, data sets with sizes of gigabytes, terabytes, or even petabytes are becoming common. If data mining algorithms are to handle these massive data sets, then they must be scalable.
2. **High Dimensionality** It is now common to encounter data sets with hun- dreds or thousands of attributes instead of the handful common a few decades ago.
3. **Heterogeneous and Complex Data** Traditional data analysis methods often deal with data sets containing attributes of the same type, either contin- uous or categorical. Examples of non-traditional types of data include collections of Web pages containing semi-structured text and hyperlinks; DNA data with sequential and three-dimensional structure; and climate data that consists of time series measurements (temperature, pressure, etc.) at various locations on the Earth's surface.

4. Data Ownership and Distribution Sometimes, the data needed for an analysis is not stored in one location or owned by one organization. Instead, the data is geographically distributed among resources belonging to multiple entities.

- This requires the development of distributed data mining techniques.
- Need to reduce the amount of communication needed to perform the distributed computation
- Need to effectively consolidate the data mining results obtained from multiple sources
- Need to address data security issues.

5. Non-traditional Analysis The traditional statistical approach is based on a hypothesize-and-test paradigm.

- In other words, a hypothesis is proposed, an experiment is designed to gather the data, and then the data is analyzed with respect to the hypothesis.
- Unfortunately, this process is extremely labor- intensive.
- Current data analysis tasks often require the generation and evaluation of thousands of hypotheses, and consequently, the development of some data mining techniques has been motivated by the desire to automate the process of hypothesis generation and evaluation.

Origins of Data Mining

- Data mining draws upon ideas, such as
- Sampling, estimation, and hypothesis testing from statistics
- Search algorithms, modeling techniques, and learning theories from artificial intelligence, pattern recognition, and machine learning.
- Database systems are needed to provide support for efficient storage, indexing, and query processing.
- Techniques from high performance (parallel) computing are often important in addressing the massive size of some data sets.
- Distributed techniques can also help address the issue of size and are essential when the data cannot be gathered in one location.

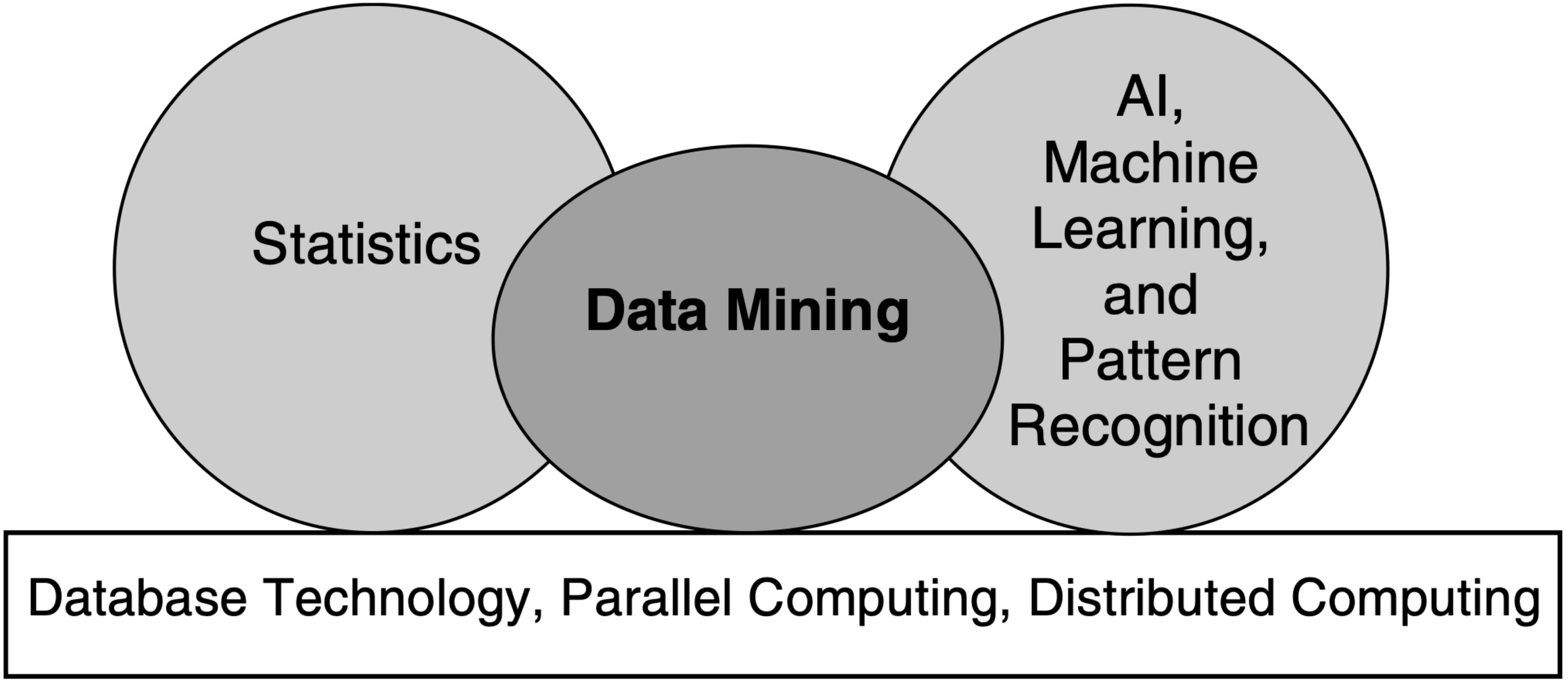


Figure 1.2. Data mining as a confluence of many disciplines.

Data Mining Tasks

Data mining tasks are generally divided into two major categories:

- **Predictive tasks.** The objective of these tasks is to predict the value of a par- ticular attribute based on the values of other attributes.
- The attribute to be predicted is commonly known as the target or **dependent** vari- able, while the attributes used for making the prediction are known as the explanatory or **independent** variables.
- **Descriptive tasks.** Here, the objective is to derive patterns (correlations, trends, clusters, trajectories, and anomalies) that summarize the underlying relationships in data.
- Descriptive data mining tasks are often exploratory in nature and frequently require postprocessing techniques to validate and explain the results.

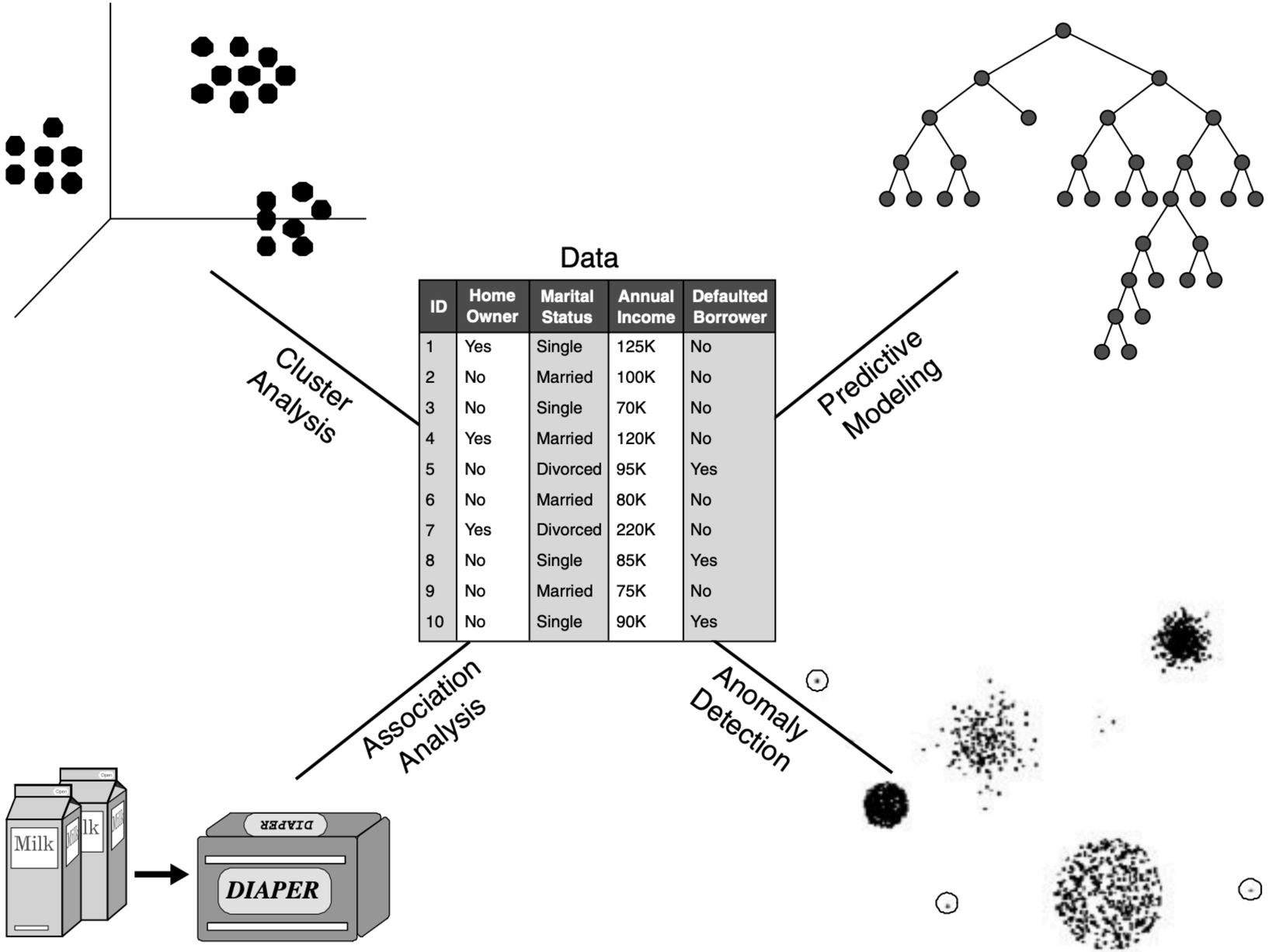


Figure 1.3. Four of the core data mining tasks.

Predictive modeling

Building a model for the target variable as a function of the explanatory variables.

- There are two types of predictive modeling tasks: **classification**, which is used for discrete target variables, and **regression**, which is used for continuous target variables.
- For example, predicting whether a Web user will make a purchase at an online bookstore is a classification task because the target variable is binary-valued.
- On the other hand, forecasting the future price of a stock is a regression task because price is a continuous-valued attribute.
- The goal of both tasks is to learn a model that minimizes the error between the predicted and true values of the target variable.

- Example - Consider the task of predicting a species of flower based on the characteristics of the flower.
- Iris Dataset - contains four attributes: sepal width, sepal length, petal length, and petal width. Based on these, species of the flower can be classified into - Setosa, Versicolour, or Virginica.
- Figure shows a plot of petal width versus petal length for the 150 flowers in the Iris data set. Petal width is broken into the categories low, medium, and high, which correspond to the intervals [0, 0.75), [0.75, 1.75), [1.75, ∞), respectively. Also, petal length is broken into categories low, medium, and high, which correspond to the intervals [0, 2.5), [2.5, 5), [5, ∞), respectively.
- Based on these categories of petal width and length, the following rules can be derived:
  1) Petal width low and petal length low implies Setosa.
  2) Petal width medium and petal length medium implies Versicolour.
  3) Petal width high and petal length high implies Virginica.
- While these rules do not classify all the flowers, they do a good (but not perfect) job of classifying most of the flowers.

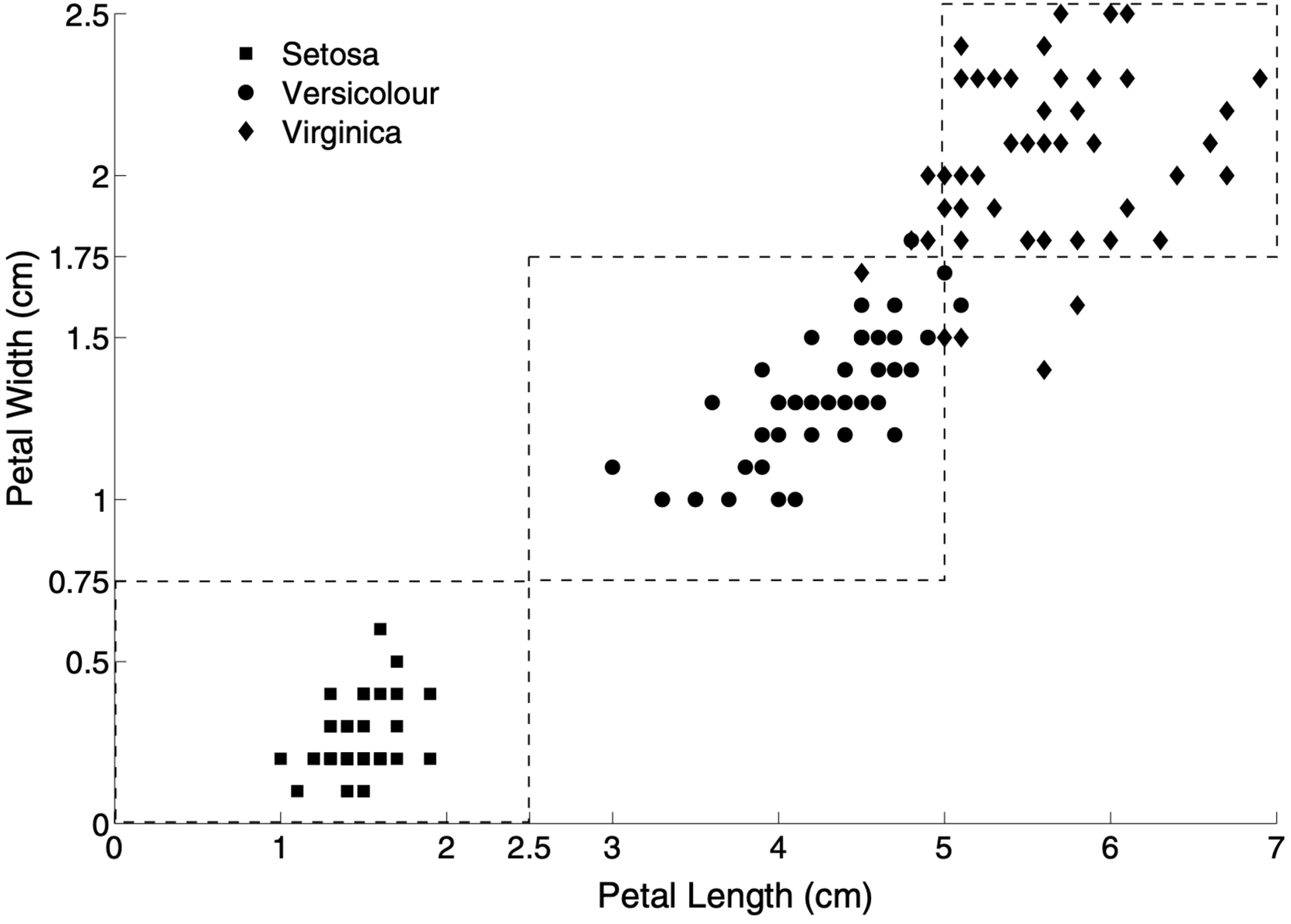


Figure 1.4. Petal width versus petal length for 150 Iris flowers.

Association analysis

Used to discover patterns that describe strongly as- sociated features in the data.

- The discovered patterns are typically represented in the form of implication rules or feature subsets.
- Useful applications of association analysis include finding groups of genes that have related functionality, identi- fying Web pages that are accessed together, or understanding the relationships between different elements of Earth's climate system.
- Example - The transactions shown in Table 1.1 illustrate point-of-sale data collected at the checkout counters of a grocery store. Association analysis can be applied to find items that are fre- quently bought together by customers. For example, we may discover the rule {Diapers} $\longrightarrow$ {Milk}, which suggests that customers who buy diapers also tend to buy milk. This type of rule can be used to identify potential cross-selling opportunities among related items.

Table 1.1. Market basket data.

Transaction ID	Items
1	{Bread, Butter, Diapers, Milk}
2	{Coffee, Sugar, Cookies, Salmon}
3	{Bread, Butter, Coffee, Diapers, Milk, Eggs}
4	{Bread, Butter, Salmon, Chicken}
5	{Eggs, Bread, Butter}
6	{Salmon, Diapers, Milk}
7	{Bread, Tea, Sugar, Eggs}
8	{Coffee, Sugar, Chicken, Eggs}
9	{Bread, Diapers, Milk, Salt}
10	{Tea, Eggs, Cookies, Diapers, Milk}

Cluster analysis

- Seeks to find groups of closely related observations so that observations that belong to the same cluster are more similar to each other than observations that belong to other clusters.
- Clustering has been used to group sets of related customers, find areas of the ocean that have a significant impact on the Earth's climate, and compress data.
- Example - The collection of news articles shown in Table 1.2 can be grouped based on their respective topics. Each article is represented as a set of word-frequency pairs (w, c), where w is a word and c is the number of times the word appears in the article. There are two natural clusters in the data set. The first cluster consists of the first four articles, which correspond to news about the economy, while the second cluster contains the last four articles, which correspond to news about health care. A good clustering algorithm should be able to identify these two clusters based on the similarity between words that appear in the articles.

Table 1.2. Collection of news articles.

Article	Words
1	dollar: 1, industry: 4, country: 2, loan: 3, deal: 2, government: 2
2	machinery: 2, labor: 3, market: 4, industry: 2, work: 3, country: 1
3	job: 5, inflation: 3, rise: 2, jobless: 2, market: 3, country: 2, index: 3
4	domestic: 3, forecast: 2, gain: 1, market: 2, sale: 3, price: 2
5	patient: 4, symptom: 2, drug: 3, health: 2, clinic: 2, doctor: 2
6	pharmaceutical: 2, company: 3, drug: 2, vaccine: 1, flu: 3
7	death: 2, cancer: 4, drug: 3, public: 4, health: 3, director: 2
8	medical: 2, cost: 3, increase: 2, patient: 2, health: 3, care: 1

Anomaly detection

- Task of identifying observations whose characteristics are significantly different from the rest of the data.
- Such observations are known as **anomalies** or **outliers**.
- The goal of an anomaly detection algorithm is to discover the real anomalies and avoid falsely labeling normal objects as anomalous.
- In other words, a good anomaly detector must have a high detection rate and a low false alarm rate.
- Applications of anomaly detection include the detection of fraud, network intrusions, unusual patterns of disease, and ecosystem disturbances.

- Example - Credit Card Fraud Detection). A credit card company records the transactions made by every credit card holder, along with personal information such as credit limit, age, annual income, and address.
- Since the number of fraudulent cases is relatively small compared to the number of legitimate transactions, anomaly detection techniques can be applied to build a profile of legitimate transactions for the users.
- When a new transaction arrives, it is compared against the profile of the user. If the characteristics of the transaction are very different from the previously created profile, then the transaction is flagged as potentially fraudulent.

Data

Data-Related Issues

- **The Type of Data** The attributes used to describe data objects can be of different types—quantitative or qualitative. The type of data determines which tools and techniques can be used to analyze the data.
- **The Quality of the Data** Data quality issues that often need to be addressed include the presence of noise and outliers; missing, inconsistent, or duplicate data; and data that is biased.
- **Preprocessing Steps to Make the Data More Suitable for Data Mining** Often, the raw data must be processed in order to make it suitable for analysis. For example, a continuous attribute, e.g., length, may need to be transformed into an attribute with discrete categories, e.g., short, medium, or long, in order to apply a particular technique (Feature Transformation). As another example, the number of attributes in a data set is often reduced because many techniques are more effective when the data has a relatively small number of attributes (Feature Selection).
- **Analyzing Data in Terms of Its Relationships** One approach to data analysis is to find relationships among the data objects and then perform the remaining analysis using these relationships rather than the data objects themselves. For instance, we can compute the similarity or distance between pairs of objects and then perform the analysis—clustering, classification, or anomaly detection—based on these similarities or distances.

Types of Data

- A data set can often be viewed as a collection of **data objects**.
- Other names for a data object are record, point, vector, pattern, event, case, sample, observation, or entity.
- In turn, data objects are described by a number of attributes that capture the basic characteristics of an object, such as the mass of a physical object or the time at which an event occurred.
- Other names for an attribute are variable, characteristic, field, feature, or dimension.

Table 2.1. A sample data set containing student information.

Student ID	Year	Grade Point Average (GPA)	...
	⋮		
1034262	Senior	3.24	...
1052663	Sophomore	3.51	...
1082246	Freshman	3.62	...
	⋮		

Attributes and Measurement

- An **attribute** is a property or characteristic of an object that may vary, either from one object to another or from one time to another.
- For example, eye color varies from person to person with a small number of possible values {brown, black, blue, green, hazel, etc.}, while the temperature of an object varies over time, and is a numerical attribute with a potentially unlimited number of values.
- A **measurement scale** is a rule (function) that associates a numerical or symbolic value with an attribute of an object.
- The process of **measurement** is the application of a measurement scale to associate a value with a particular attribute of a specific object.

The Type of an Attribute

- The values used to represent an attribute may have properties that are not properties of the attribute itself, and vice versa.
- Example - Two attributes that might be associated with an employee are ID and age (in years). Both of these attributes can be represented as integers. However, while it is reasonable to talk about the average age of an employee, it makes no sense to talk about the average employee ID. The only valid operation for employee IDs is to test whether they are equal. For the age attribute, the properties of the integers used to represent age are very much the properties of the attribute. Even so, the correspondence is not complete since, for example, ages have a maximum, while integers do not.

The Different Types of Attributes

- The following properties (operations) of numbers are typically used to describe attributes.
- 1. Distinctness = and/=
- 2. Order <, ≤, >, and ≥
- 3. Addition + and -
- 4. Multiplication * and /
- Given these properties, we can define four types of attributes: nominal, ordinal, interval, and ratio.

Table 2.2. Different attribute types.

Attribute Type		Description	Examples	Operations
Categorical (Qualitative)	Nominal	The values of a nominal attribute are just different names; i.e., nominal values provide only enough information to distinguish one object from another. $(=, \neq)$	zip codes, employee ID numbers, eye color, gender	mode, entropy, contingency correlation, χ^2 test
	Ordinal	The values of an ordinal attribute provide enough information to order objects. $(<, >)$	hardness of minerals, $\{good, better, best\}$, grades, street numbers	median, percentiles, rank correlation, run tests, sign tests
Numeric (Quantitative)	Interval	For interval attributes, the differences between values are meaningful, i.e., a unit of measurement exists. $(+, -)$	calendar dates, temperature in Celsius or Fahrenheit	mean, standard deviatio Pearson's correlation, t and F tests
	Ratio	For ratio variables, both differences and ratios are meaningful. $(*, /)$	temperature in Kelvin, monetary quantities, counts, age, mass, length, electrical current	geometric mean, harmonic mean, percent variation

Entropy

- I have a box full of an equal number of coffee pouches of two flavors: Caramel Latte and the regular, Cappuccino. You may choose either of the flavors but with eyes closed.
- This predicament where you would have to decide and this decision of yours that can lead to results with equal probability is nothing else but said to be the state of maximum uncertainty.
- In case, I had only caramel latte coffee pouches or cappuccino pouches then we know what the outcome would have been and hence the uncertainty (or surprise) will be zero.
- The probability of getting each outcome of a caramel latte pouch or cappuccino pouch is
- P(Coffee pouch == Caramel Latte) = 0.50
- P(Coffee pouch == Cappuccino) = 1 – 0.50 = 0.50
- When we have only one result either caramel latte or cappuccino pouch, then in the absence of uncertainty, the probability of the event is:
- P(Coffee pouch == Caramel Latte) = 1
- P(Coffee pouch == Cappuccino) = 1 – 1 = 0
- There is a relationship between heterogeneity and uncertainty; the more heterogeneous the event the more uncertainty. On the other hand, the less heterogeneous, or so to say, the more homogeneous the event, the lesser is the uncertainty. The uncertainty is expressed as **Gini** or **Entropy**.

Contingency Correlation and chi-square test

- Contingency correlation or Crosstabulation shows whether being in one category of the independent variable makes a case more likely to be in a particular category of the dependent variable. This allows researchers to examine the association between two categorical variables.
- The Chi-Square statistic is most commonly used to evaluate Tests of Independence when using a crosstabulation (also known as a bivariate table). The Test of Independence assesses whether an association exists between the two variables by comparing the observed pattern of responses in the cells to the pattern that would be expected if the variables were truly independent of each other.
- Calculating the Chi-Square statistic and comparing it against a critical value from the Chi-Square distribution allows the researcher to assess whether the observed cell counts are significantly different from the expected cell counts.
- Example - diabetic people are heart patients or not

- Nominal and ordinal attributes are collectively referred to as **categorical** or **qualitative** attributes.
- As the name suggests, qualitative attributes, such as employee ID, lack most of the properties of numbers. Even if they are represented by numbers, i.e., integers, they should be treated more like symbols.
- The remaining two types of attributes, interval and ratio, are collectively referred to as **quantitative** or **numeric** attributes.
- Quantitative attributes are represented by numbers and have most of the properties of numbers. Quantitative attributes can be integer-valued or continuous.
- The types of attributes can also be described in terms of transformations that do not change the meaning of an attribute.
- Example - the meaning of a length attribute is unchanged if it is measured in meters instead of feet.

Table 2.3. Transformations that define attribute levels.

Attribute Type		Transformation	Comment
Categorical (Qualitative)	Nominal	Any one-to-one mapping, e.g., a permutation of values	If all employee ID numbers are reassigned, it will not make any difference.
	Ordinal	An order-preserving change of values, i.e., $new\_value = f(old\_value)$, where f is a monotonic function.	An attribute encompassing the notion of good, better, best can be represented equally well by the values $\{1, 2, 3\}$ or by $\{0.5, 1, 10\}$.
Numeric (Quantitative)	Interval	$new\_value = a * old\_value + b$, a and b constants.	The Fahrenheit and Celsius temperature scales differ in the location of their zero value and the size of a degree (unit).
	Ratio	$new\_value = a * old\_value$	Length can be measured in meters or feet.

Describing Attributes by the Number of Values

- **Discrete** A discrete attribute has a finite or countably infinite set of values. Such attributes can be categorical, such as zip codes or ID numbers, or numeric, such as counts. Discrete attributes are often represented using integer variables. **Binary** attributes are a special case of discrete attributes and assume only two values, e.g., true/false, yes/no, male/female, or 0/1. Binary attributes are often represented as Boolean variables, or as integer variables that only take the values 0 or 1.
- **Continuous** A continuous attribute is one whose values are real numbers. Examples include attributes such as temperature, height, or weight. Continuous attributes are typically represented as floating-point variables.

Asymmetric Attributes

- For asymmetric attributes, only presence—a non-zero attribute value—is regarded as important.
- Consider a data set where each object is a student and each attribute records whether or not a student took a particular course at a university.
- For a specific student, an attribute has a value of 1 if the student took the course associated with that attribute and a value of 0 otherwise.
- Because students take only a small fraction of all available courses, most of the values in such a data set would be 0. Therefore, it is more meaningful and more efficient to focus on the non-zero values.
- Binary attributes where only non-zero values are important are called **asymmetric binary** attributes.
- If the number of credits associated with each course is recorded, then the resulting data set will consist of **asymmetric discrete or continuous attributes**.

General Characteristics of Data Sets

Dimensionality, Sparsity, and Resolution

- **Dimensionality** The dimensionality of a data set is the number of attributes that the objects in the data set possess.
- Data with a small number of dimensions tends to be qualitatively different than moderate or high-dimensional data.
- Indeed, the difficulties associated with analyzing high-dimensional data are sometimes referred to as the **curse of dimensionality**.
- Because of this, an important motivation in preprocessing the data is **dimensionality reduction**.

- **Sparsity** For some data sets, such as those with asymmetric features, most attributes of an object have values of 0; in many cases, fewer than 1% of the entries are non-zero.
- In practical terms, sparsity is an advantage because usually only the non-zero values need to be stored and manipulated. This results in significant savings with respect to computation time and storage.
- **Resolution** It is frequently possible to obtain data at different levels of resolution, and often the properties of the data are different at different resolutions.
- For instance, the surface of the Earth seems very uneven at a resolution of a few meters, but is relatively smooth at a resolution of tens of kilometers.
- The patterns in the data also depend on the level of resolution. If the resolution is too fine, a pattern may not be visible or may be buried in noise; if the resolution is too coarse, the pattern may disappear.
- For example, variations in atmospheric pressure on a scale of hours reflect the movement of storms and other weather systems. On a scale of months, such phenomena are not detectable.

Types of Data Sets

Record data, Graph- based data, and Ordered data

Record Data

- Data set is a collection of records (data objects), each of which consists of a fixed set of data fields (attributes).
- For the most basic form of record data, there is no explicit relationship among records or data fields, and every record (object) has the same set of attributes.
- Record data is usually stored either in flat files or in relational databases.
- Different types of record data - Transaction or Market Basket Data, The Data Matrix, The Sparse Data Matrix

Tid	Refund	Marital Status	Taxable Income	Defaulted Borrower
1	Yes	Single	125K	**No**
2	No	Married	100K	**No**
3	No	Single	70K	**No**
4	Yes	Married	120K	**No**
5	No	Divorced	95K	**Yes**
6	No	Married	60K	**No**
7	Yes	Divorced	220K	**No**
8	No	Single	85K	**Yes**
9	No	Married	75K	**No**
10	No	Single	90K	**Yes**

(a) Record data.

TID	*ITEMS*
1	Bread, Soda, Milk
2	Beer, Bread
3	Beer, Soda, Diaper, Milk
4	Beer, Bread, Diaper, Milk
5	Soda, Diaper, Milk

(b) Transaction data.

Projection of x Load	Projection of y Load	Distance	Load	Thickness
10.23	5.27	15.22	27	1.2
12.65	6.25	16.22	22	1.1
13.54	7.23	17.34	23	1.2
14.27	8.43	18.45	25	0.9

(c) Data matrix.

	team	coach	play	ball	score	game	win	lost	timeout	season
Document 1	3	0	5	0	2	6	0	2	0	2
Document 2	0	7	0	2	1	0	0	3	0	0
Document 3	0	1	0	0	1	2	2	0	3	0

(d) Document-term matrix.

Figure 2.2. Different variations of record data.

Transaction or Market Basket Data

- Transaction data is a special type of record data, where each record (transaction) involves a set of items.
- Consider a grocery store. The set of products purchased by a customer during one shopping trip constitutes a transaction, while the individual products that were purchased are the items. This type of data is called market basket data because the items in each record are the products in a person's "market basket."
- Transaction data is a collection of sets of items, but it can be viewed as a set of records whose fields are asymmetric attributes. Most often, the attributes are binary, indicating whether or not an item was purchased, but more generally, the attributes can be discrete or continuous, such as the number of items purchased or the amount spent on those items.

The Data Matrix

- If the data objects in a collection of data all have the same fixed set of numeric attributes, then the data objects can be thought of as points (vectors) in a multidimensional space, where each dimension represents a distinct attribute describing the object.
- A set of such data objects can be interpreted as an m by n matrix, where there are m rows, one for each object, and n columns, one for each attribute.
- A data matrix or a pattern matrix is a variation of record data, but because it consists of numeric attributes, standard matrix operation can be applied to transform and manipulate the data.
- Therefore, the data matrix is the standard data format for most statistical data.

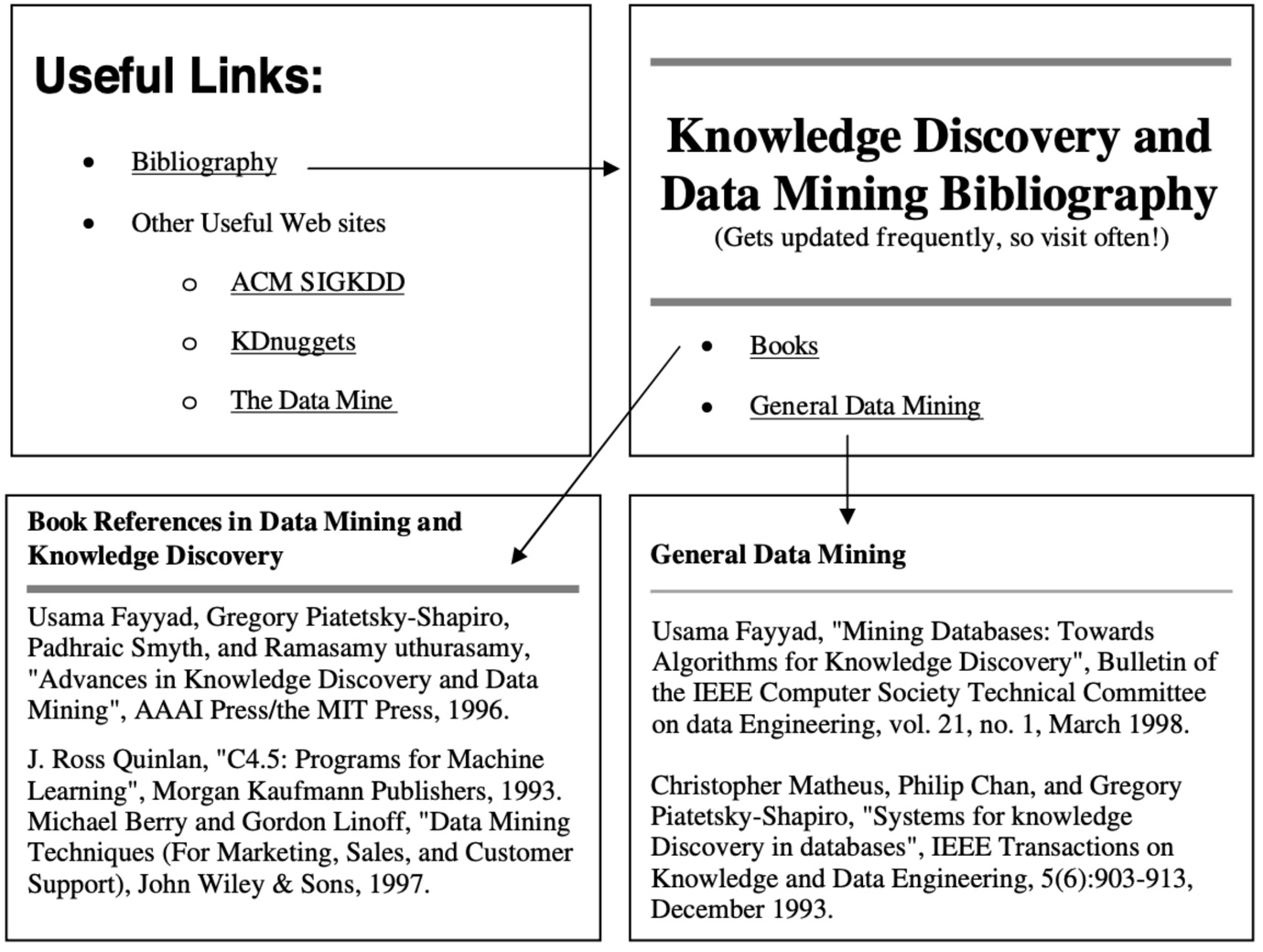
The Sparse Data Matrix

- A sparse data matrix is a special case of a data matrix in which the attributes are of the same type and are asymmetric; i.e., only non-zero values are important.
- Transaction data is an example of a sparse data matrix that has only 0–1 entries.
- Another common example is document data. In particular, if the order of the terms (words) in a document is ignored, then a document can be represented as a term vector, where each term is a component (attribute) of the vector and the value of each component is the number of times the corresponding term occurs in the document. This representation of a collection of documents is often called a document-term matrix.

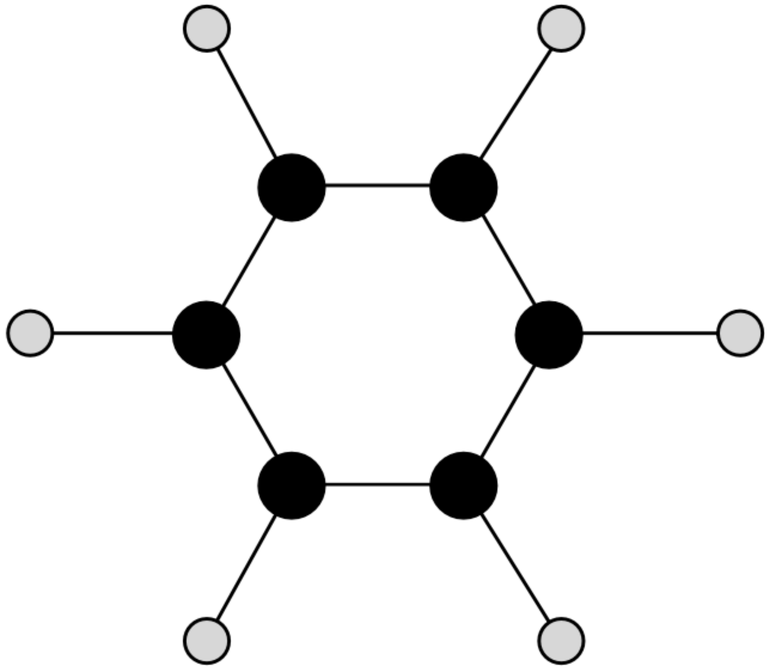
Graph-Based Data

- Two specific cases: (1) the graph captures relationships among data objects and (2) the data objects themselves are represented as graphs.
- **Data with Relationships among Objects** The data is often represented as a graph. In particular, the data objects are mapped to nodes of the graph, while the relationships among objects are captured by the links between objects and link properties, such as direction and weight.
- Consider Web pages on the World Wide Web, which contain both text and links to other pages. In order to process search queries, Web search engines collect and process Web pages to extract their contents. It is well known, however, that the links to and from each page provide a great deal of information about the relevance of a Web page to a query, and thus, must also be taken into consideration.

- **Data with Objects That Are Graphs** If objects have structure, that is, the objects contain subobjects that have relationships, then such objects are frequently represented as graphs.
- For example, the structure of chemical compounds can be represented by a graph, where the nodes are atoms and the links between nodes are chemical bonds.
- Figure 2.3(b) shows a ball-and-stick diagram of the chemical compound benzene, which contains atoms of carbon (black) and hydrogen (gray).



(a) Linked Web pages.



(b) Benzene molecule.

Figure 2.3. Different variations of graph data.

Ordered Data

The attributes have relationships that involve order in time or space.

- **Sequential Data** Sequential data, also referred to as temporal data, can be thought of as an extension of record data, where each record has a time associated with it.
- Consider a retail transaction data set that also stores the time at which the transaction took place. This time information makes it possible to find patterns such as “candy sales peak before Halloween.”
- A time can also be associated with each attribute. For example, each record could be the purchase history of a customer, with a listing of items purchased at different times. Using this information, it is possible to find patterns such as “people who buy DVD players tend to buy DVDs in the period immediately following the purchase.”

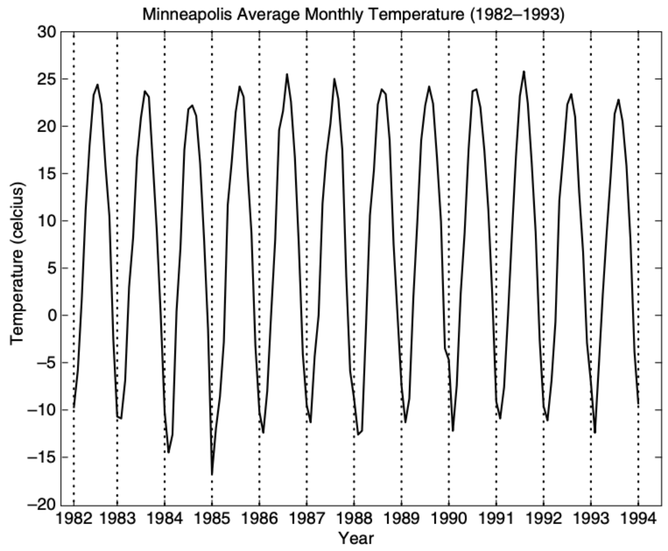
Time	Customer	Items Purchased
t1	C1	A, B
t2	C3	A, C
t2	C1	C, D
t3	C2	A, D
t4	C2	E
t5	C1	A, E

Customer	Time and Items Purchased
C1	(t1: A,B) (t2:C,D) (t5:A,E)
C2	(t3: A, D) (t4: E)
C3	(t2: A, C)

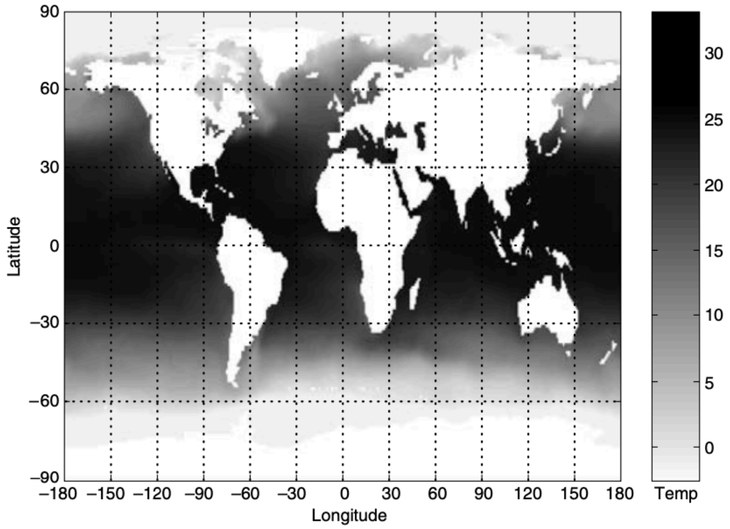
(a) Sequential transaction data.

```
GGTTCCGCCTTCAGCCCCGCGCC
CGCAGGGCCCGCCCCGCGCCGTC
GAGAAGGGCCCGCCTGGCGGGCG
GGGGGAGGCGGGGCCGCCCGAGC
CCAACCGAGTCCGACCAGGTGCC
CCCTCTGCTCGGCCTAGACCTGA
GCTCATTAGGCGGCAGCGGACAG
GCCAAGTAGAACACGCGAAGCGC
TGGGCTGCCTGCTGCGACCAGGG
```

(b) Genomic sequence data.



(c) Temperature time series.



(d) Spatial temperature data.

Figure 2.4. Different variations of ordered data.

- **Sequence Data** Sequence data consists of a data set that is a sequence of individual entities, such as a sequence of words or letters.
- It is quite similar to sequential data, except that there are no time stamps; instead, there are positions in an ordered sequence.
- For example, the genetic information of plants and animals can be represented in the form of sequences of nucleotides that are known as genes.
- Many of the problems associated with genetic sequence data involve predicting similarities in the structure and function of genes from similarities in nucleotide sequences.

- **Time Series Data** Time series data is a special type of sequential data in which each record is a time series, i.e., a series of measurements taken over time.
- For example, a financial data set might contain objects that are time series of the daily prices of various stocks.
- When working with temporal data, it is important to consider temporal autocorrelation; i.e., if two measurements are close in time, then the values of those measurements are often very similar.

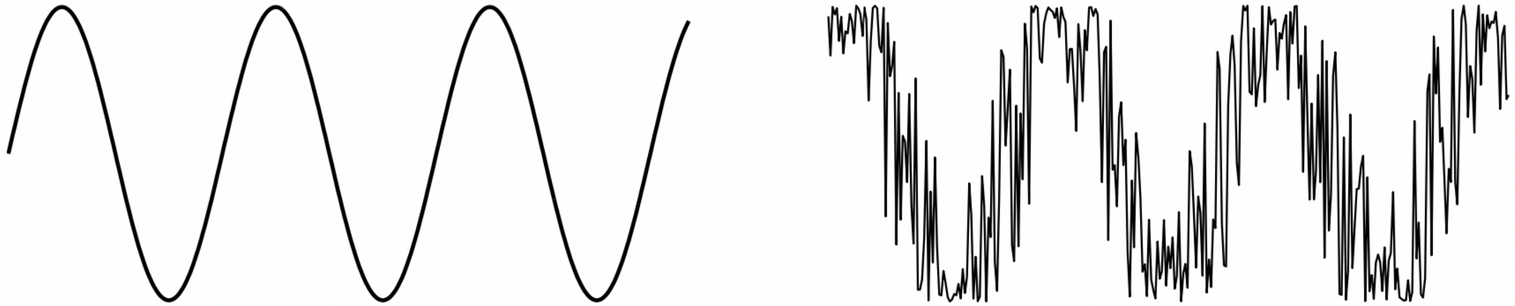
- **Spatial Data** Some objects have spatial attributes, such as positions or areas, as well as other types of attributes.
- An example of spatial data is weather data (precipitation, temperature, pressure) that is collected for a variety of geographical locations.
- An important aspect of spatial data is spatial auto-correlation; i.e., objects that are physically close tend to be similar in other ways as well. Thus, two points on the Earth that are close to each other usually have similar values for temperature and rainfall.
- For instance, Earth science data sets record the temperature or pressure measured at points (grid cells) on latitude–longitude spherical grids.

Data Quality

- The detection and correction of data quality problems - data cleaning
- The use of algorithms that can tolerate poor data quality.
- The term **measurement error** refers to any problem resulting from the measurement process. A common problem is that the value recorded differs from the true value to some extent.
- For continuous attributes, the numerical difference of the measured and true value is called the error.
- The term **data collection error** refers to errors such as omitting data objects or attribute values, or inappropriately including a data object. For example, a study of animals of a certain species might include animals of a related species that are similar in appearance to the species of interest.
- For example, keyboard errors are common when data is entered manually, and as a result, many data entry programs have techniques for detecting and, with human intervention, correcting such errors.

1. Noise and Artifacts

- Noise is the random component of a measurement error. It may involve the distortion of a value or the addition of spurious objects.
- Artifact is any error in the perception or representation of any information introduced by the involved equipment or technique(s). Example - a streak in the same place on a set of photographs.



(a) Time series.

(b) Time series with noise.

Figure 2.5. Noise in a time series context.

(a) Three groups of points.

(b) With noise points (+) added.

Figure 2.6. Noise in a spatial context.

2. Precision, Bias, Accuracy

- **Precision** - The closeness of repeated measurements (of the same quantity) to one another. Measured by the standard deviation of a set of values.
- **Bias** - A systematic variation of measurements from the quantity being measured. Measured by taking the difference between the mean of the set of values and the known value of the quantity being measured.
- Suppose that we have a standard laboratory weight with a mass of 1g and want to assess the precision and bias of our new laboratory scale. We weigh the mass five times, and obtain the following five values: {1.015,0.990,1.013,1.001,0.986}. The mean of these values is 1.001, and hence, the bias is 0.001. The precision, as measured by the standard deviation, is 0.013.
- **Accuracy** - The closeness of measurements to the true value of the quantity being measured.

3. Outliers

- Outliers are either (1) data objects that, in some sense, have characteristics that are different from most of the other data objects in the data set, or (2) values of an attribute that are unusual with respect to the typical values for that attribute. Alternatively, we can speak of anomalous objects or values.
- Outliers can be legitimate data objects or values. Thus, unlike noise, outliers may sometimes be of interest.
- In fraud and network intrusion detection, for example, the goal is to find unusual objects or events from among a large number of normal ones.

4. Missing Values

- It is not unusual for an object to be missing one or more attribute values.
- In some cases, the information was not collected; e.g., some people decline to give their age or weight.
- In other cases, some attributes are not applicable to all objects; e.g., often, forms have conditional parts that are filled out only when a person answers a previous question in a certain way, but for simplicity, all fields are stored.
- Techniques to handle - **Eliminate Data Objects or Attributes, Estimate Missing Values, Ignore the Missing Value during Analysis**

1. **Eliminate Data Objects or Attributes** A simple and effective strategy is to eliminate objects with missing values. However, even a partially specified data object contains some information, and if many objects have missing values, then a reliable analysis can be difficult or impossible. A related strategy is to eliminate attributes that have missing values. This should be done with caution, however, since the eliminated attributes may be the ones that are critical to the analysis.
2. **Estimate Missing Values** Sometimes missing data can be reliably estimated. Consider a data set that has many similar data points. In this situation, the attribute values of the points closest to the point with the missing value are often used to estimate the missing value. If the attribute is continuous, then the average attribute value of the nearest neighbors is used; if the attribute is categorical, then the most commonly occurring attribute value can be taken.
3. **Ignore the Missing Value during Analysis** Many data mining approaches can be modified to ignore missing values. For example, suppose that objects are being clustered and the similarity between pairs of data objects needs to be calculated. If one or both objects of a pair have missing values for some attributes, then the similarity can be calculated by using only the attributes that do not have missing values.

5. Inconsistent Values

- Data can contain inconsistent values.
- Consider an address field, where both a zip code and city are listed, but the specified zip code area is not contained in that city. It may be that the individual entering this information transposed two digits, or perhaps a digit was misread when the information was scanned from a handwritten form.
- Some types of inconsistencies are easy to detect. For instance, a person's height should not be negative.
- In other cases, it can be necessary to consult an external source of information. For example, when an insurance company processes claims for reimbursement, it checks the names and addresses on the reimbursement forms against a database of its customers.
- Once an inconsistency has been detected, it is sometimes possible to correct the data. The correction of an inconsistency requires additional or redundant information.

6. Duplicate Data

- A data set may include data objects that are duplicates, or almost duplicates, of one another.
- Many people receive duplicate mailings because they appear in a database multiple times under slightly different names.
- To detect and eliminate such duplicates, two main issues must be addressed.
- First, if there are two objects that actually represent a single object, then the values of corresponding attributes may differ, and these inconsistent values must be resolved.
- Second, care needs to be taken to avoid accidentally combining data objects that are similar, but not duplicates, such as two distinct people with identical names. The term **deduplication** is often used to refer to the process of dealing with these issues.

7. Issues Related to Applications

There are many issues that are specific to particular applications and fields.

- **Timeliness** Some data starts to age as soon as it has been collected. In particular, if the data provides a snapshot of some ongoing phenomenon or process, such as the purchasing behavior of customers or Web browsing patterns, then this snapshot represents reality for only a limited time. If the data is out of date, then so are the models and patterns that are based on it.
- **Relevance** The available data must contain the information necessary for the application. Consider the task of building a model that predicts the accident rate for drivers. If information about the age and gender of the driver is omitted, then it is likely that the model will have limited accuracy unless this information is indirectly available through other attributes.

7. Issues Related to Applications

- **Sampling bias**, which occurs when a sample does not contain different types of objects in proportion to their actual occurrence in the population. Because the results of a data analysis can reflect only the data that is present, sampling bias will typically result in an erroneous analysis.
- **Knowledge about the Data** Ideally, data sets are accompanied by documentation that describes different aspects of the data; the quality of this documentation can either aid or hinder the subsequent analysis. For example, if the documentation identifies several attributes as being strongly related, these attributes are likely to provide highly redundant information, and we may decide to keep just one. If the documentation is poor, however, and fails to tell us, for example, that the missing values for a particular field are indicated with a -9999, then our analysis of the data may be faulty. Other important characteristics are the precision of the data, the type of features (nominal, ordinal, interval, ratio), the scale of measurement (e.g., meters or feet for length), and the origin of the data.

Data Preprocessing

- Selecting data objects and attributes for the analysis or creating/changing the attributes.

1. Aggregation
2. Sampling
3. Dimensionality reduction
4. Feature subset selection
5. Feature creation
6. Discretization and binarization
7. Variable transformation

1. Aggregation

The combining of two or more objects into a single object.

- Consider a data set consisting of transactions (data objects) recording the daily sales of products in various store locations (Minneapolis, Chicago, Paris, ...) for different days over the course of a year.
- One way to aggregate transactions for this data set is to replace all the transactions of a single store with a single storewide transaction. This reduces the hundreds or thousands of transactions that occur daily at a specific store to a single daily transaction, and the number of data objects is reduced to the number of stores.
- Quantitative attributes, such as price, are typically aggregated by taking a sum or an average. A qualitative attribute, such as item, can either be omitted or summarized as the set of all the items that were sold at that location.

Table 2.4. Data set containing information about customer purchases.

Transaction ID	Item	Store Location	Date	Price	...
⋮	⋮	⋮	⋮	⋮	
101123	Watch	Chicago	09/06/04	$25.99	...
101123	Battery	Chicago	09/06/04	$5.99	...
101124	Shoes	Minneapolis	09/06/04	$75.00	...
⋮	⋮	⋮	⋮	⋮	

- Another possibility is reducing the possible values for date from 365 days to 12 months.

Why Aggregation?

- The smaller data sets resulting from data reduction require less memory and processing time, and hence, aggregation may permit the use of more expensive data mining algorithms.
- Aggregation can act as a change of scope or scale by providing a high-level view of the data instead of a low-level view. In the previous example, aggregating over store locations and months gives us a monthly, per store view of the data instead of a daily, per item view.
- Finally, the behaviour of groups of objects or attributes is often more stable than that of individual objects or attributes. This statement reflects the statistical fact that aggregate quantities, such as averages or totals, have less variability than the individual objects being aggregated.
- A disadvantage of aggregation is the potential loss of interesting details. In the store example aggregating over months loses information about which day of the week has the highest sales.

2. Sampling

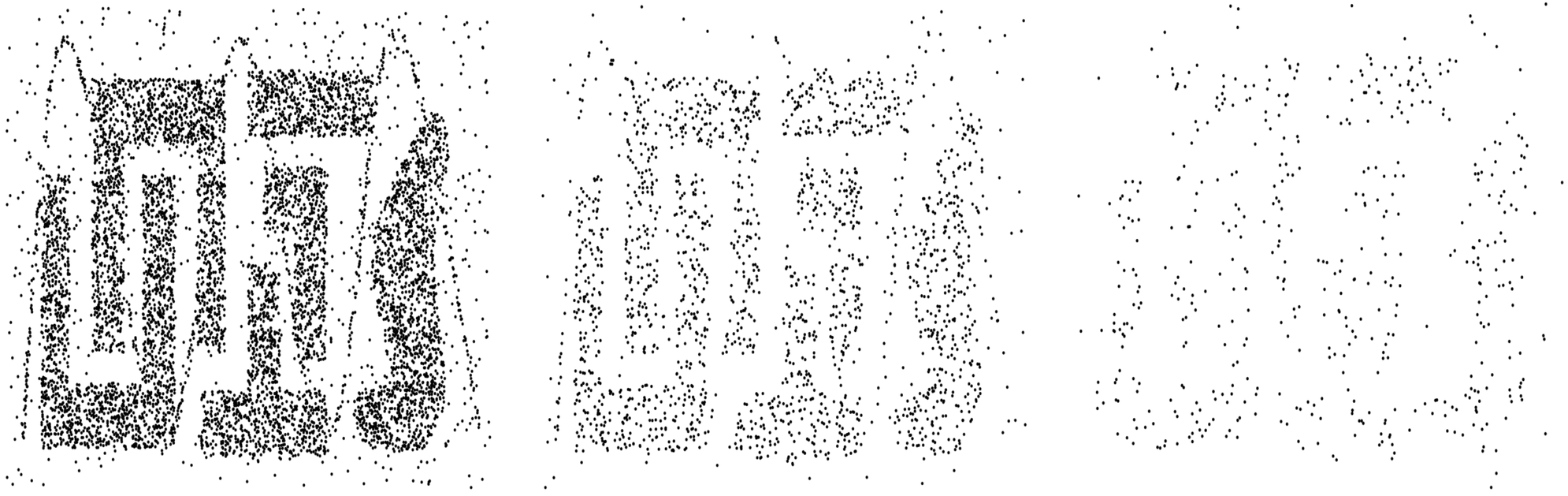
- Sampling is a commonly used approach for selecting a subset of the data objects to be analyzed.
- Statisticians use sampling because obtaining the entire set of data of interest is too expensive or time consuming, while data miners sample because it is too expensive or time consuming to process all the data.
- In some cases, using a sampling algorithm can reduce the data size to the point where a better, but more expensive algorithm can be used.
- Using a sample will work almost as well as using the entire data set if the sample is representative i.e., it has approximately the same property (of interest) as the original set of data. If the mean (average) of the data objects is the property of interest, then a sample is representative if it has a mean that is close to that of the original data.

Simple random sampling

There is an equal probability of selecting any particular item.

- There are two variations on random sampling (and other sampling techniques as well): (1) sampling without replacement—as each item is selected, it is removed from the set of all objects that together constitute the population, and (2) sampling with replacement—objects are not removed from the population as they are selected for the sample.
- In sampling with replacement, the same object can be picked more than once.
- When the population consists of different types of objects, with widely different numbers of objects, simple random sampling can fail to adequately represent those types of objects that are less frequent. For example, when building classification models for rare classes, it is critical that the rare classes be adequately represented in the sample.

- In **Stratified sampling**, which starts with prespecified groups of objects, equal numbers of objects are drawn from each group even though the groups are of different sizes.
- In another variation, the number of objects drawn from each group is proportional to the size of that group.
- Once a sampling technique has been selected, it is still necessary to choose the sample size.
- Larger sample sizes increase the probability that a sample will be representative, but they also eliminate much of the advantage of sampling.
- Conversely, with smaller sample sizes, patterns may be missed or erroneous patterns can be detected.



(a) 8000 points (b) 2000 points (c) 500 points

Figure 2.9. Example of the loss of structure with sampling.

- Figure 2.9(a) shows a data set that contains 8000 two-dimensional points, while Figures 2.9(b) and 2.9(c) show samples from this data set of size 2000 and 500, respectively. Although most of the structure of this data set is present in the sample of 2000 points, much of the structure is missing in the sample of 500 points.

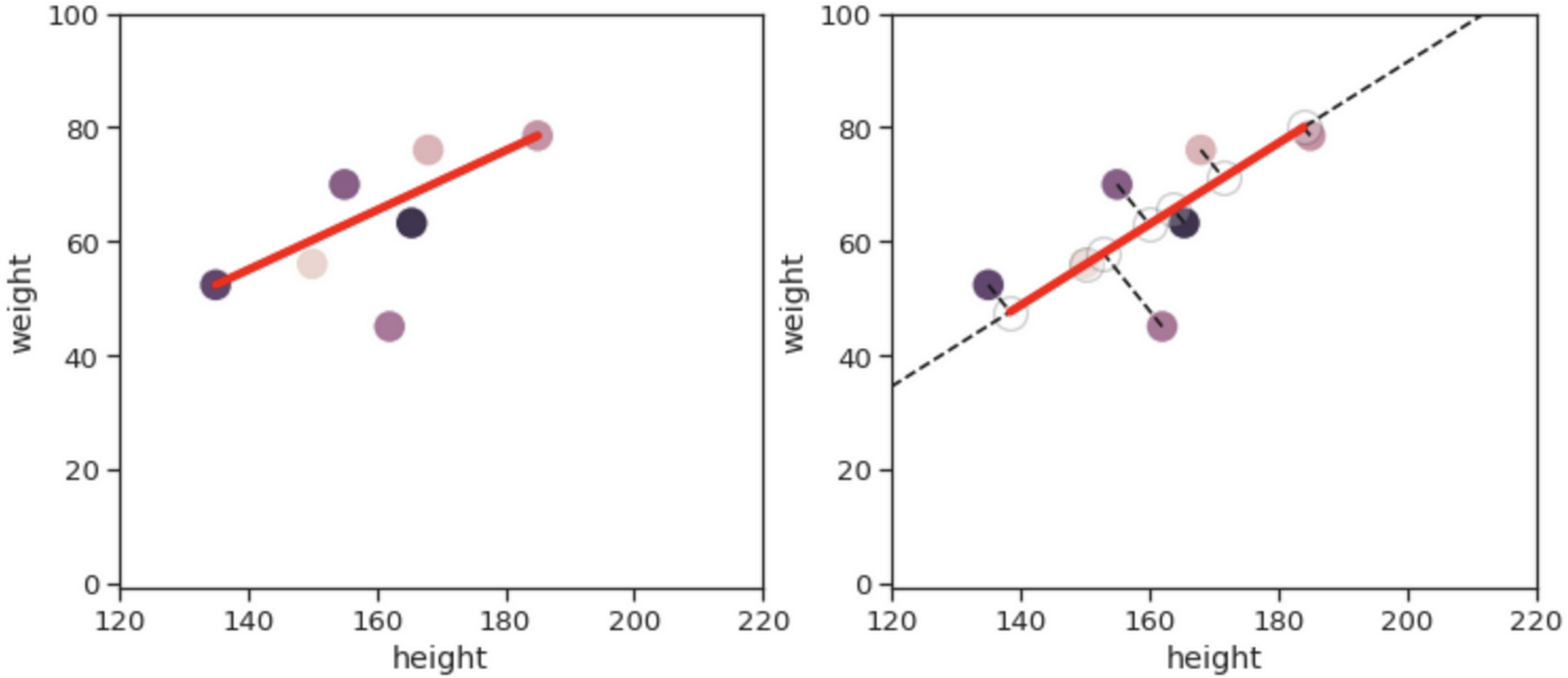
- **Adaptive or progressive sampling** start with a small sample, and then increase the sample size until a sample of sufficient size has been obtained.
- While this technique eliminates the need to determine the correct sample size initially, it requires that there be a way to evaluate the sample to judge if it is large enough.
- Suppose, for instance, that progressive sampling is used to learn a predictive model. Although the accuracy of predictive models increases as the sample size increases, at some point the increase in accuracy levels off. We want to stop increasing the sample size at this leveling-off point.
- By keeping track of the change in accuracy of the model as we take progressively larger samples, and by taking other samples close to the size of the current one, we can get an estimate as to how close we are to this leveling-off point, and thus, stop sampling.

.

3. Dimensionality Reduction

- Data sets can have a large number of features. Consider a set of documents, where each document is represented by a vector whose components are the frequencies with which each word occurs in the document.
- In such cases, there are typically thousands or tens of thousands of attributes (components), one for each word in the vocabulary.
- As another example, consider a set of time series consisting of the daily closing price of various stocks over a period of 30 years. In this case, the attributes, which are the prices on specific days, again number in the thousands.
- Many data mining algorithms work better if the dimensionality—the number of attributes in the data—is lower. This is partly because dimensionality reduction can eliminate irrelevant features and reduce noise.
- Another benefit is that a reduction of dimensionality can lead to a more understandable model because the model may involve fewer attributes.
- Also, dimensionality reduction may allow the data to be more easily visualized. Even if dimensionality reduction doesn't reduce the data to two or three dimensions, data is often visualized by looking at pairs or triplets of attributes, and the number of such combinations is greatly reduced.
- Finally, the amount of time and memory required by the data mining algorithm is reduced with a reduction in dimensionality.

- The term dimensionality reduction is often reserved for those techniques that reduce the dimensionality of a data set by creating new attributes that are a combination of the old attributes.
- The **curse of dimensionality** refers to the phenomenon that many types of data analysis become significantly harder as the dimensionality of the data increases.
- Specifically, as dimensionality increases, the data becomes increasingly **sparse** in the space that it occupies.
- For classification, this can mean that there are not enough data objects to allow the creation of a model that reliably assigns a class to all possible objects.
- For clustering, the definitions of density and the distance between points, which are critical for clustering, become less meaningful.
- As a result, many clustering and classification algorithms (and other data analysis algorithms) have trouble with high-dimensional data—reduced classification accuracy and poor quality clusters.



- **Principal Components Analysis (PCA)** is a linear algebra technique for continuous attributes that finds new attributes (principal components) that (1) are linear combinations of the original attributes, (2) are orthogonal (perpendicular) to each other, and (3) capture the maximum amount of variation in the data.
- **Singular Value Decomposition (SVD)** is a linear algebra technique that is related to PCA and is also commonly used for dimensionality reduction.

4. Feature Subset Selection

- Another way to reduce the dimensionality is to use only a subset of the features.
- Redundant features duplicate much or all of the information contained in one or more other attributes. For example, the purchase price of a product and the amount of sales tax paid contain much of the same information.
- Irrelevant features contain almost no useful information for the data mining task at hand. For instance, students' ID numbers are irrelevant to the task of predicting students' grade point averages.
- Redundant and irrelevant features can reduce classification accuracy and the quality of the clusters that are found.
- The ideal approach to feature selection is to try all possible subsets of features as input to the data mining algorithm of interest, and then take the subset that produces the best results. Unfortunately, since the number of subsets involving n attributes is 2 to the power n, such an approach is impractical.
- There are three standard approaches to feature selection: embedded, filter, and wrapper.

- **Embedded approaches** Feature selection occurs naturally as part of the data mining algorithm. Specifically, during the operation of the data mining algorithm, the algorithm itself decides which attributes to use and which to ignore. Algorithms for building decision tree classifiers, often operate in this manner.
- **Filter approaches** Features are selected before the data mining algorithm is run, using some approach that is independent of the data mining task. For example, we might select sets of attributes whose pairwise correlation is as low as possible.
- **Wrapper approaches** These methods use the target data mining algorithm as a black box to find the best subset of attributes, in a way similar to that of the ideal algorithm described above, but typically without enumerating all possible subsets.

.

Architecture for Feature Subset Selection

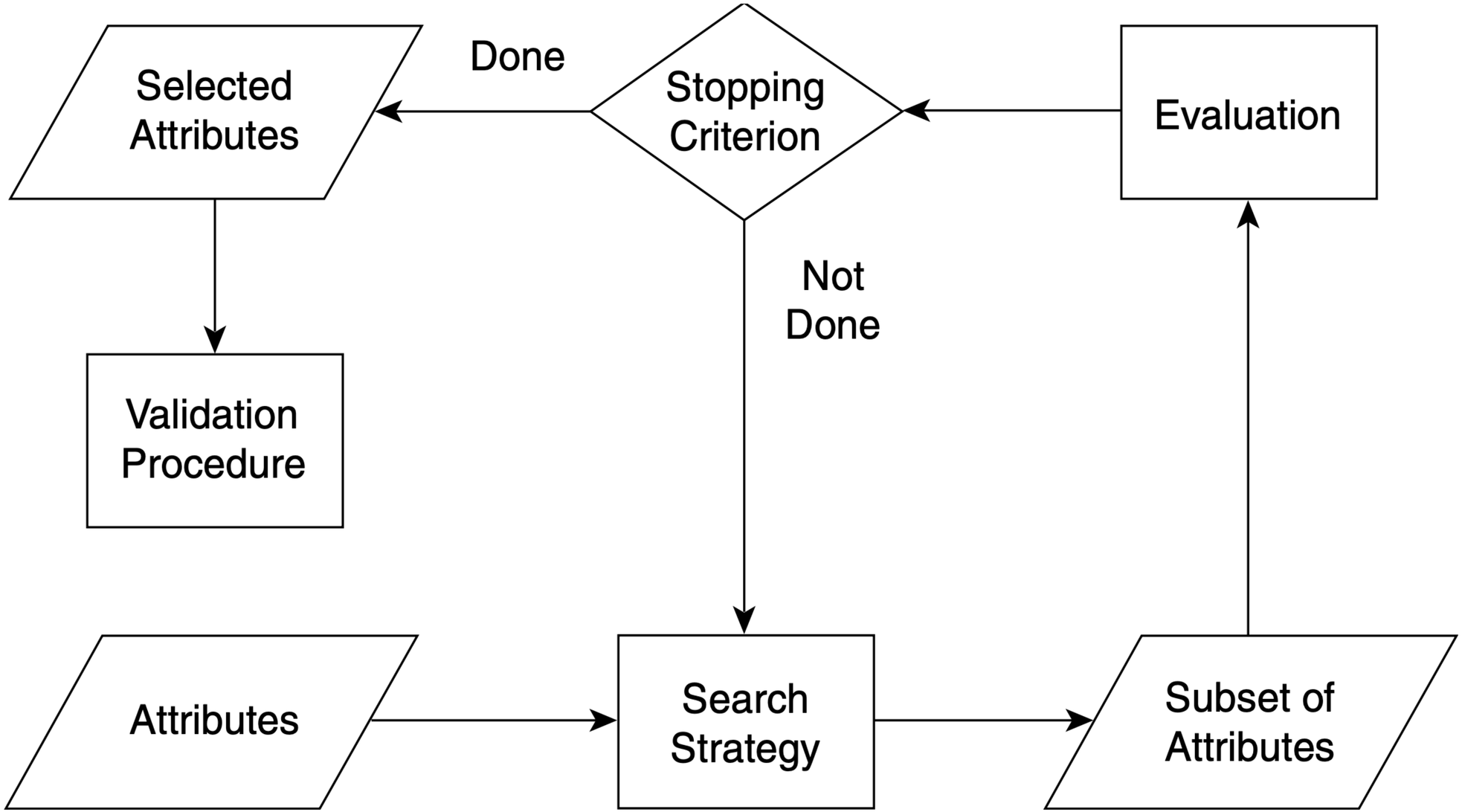


Figure 2.11. Flowchart of a feature subset selection process.

- The feature selection process is viewed as consisting of four parts: a measure for evaluating a subset, a search strategy that controls the generation of a new subset of features, a stopping criterion, and a valida- tion procedure.
- Feature subset selection is a **search** over all possible subsets of features. Many different types of search strategies can be used, but the search strategy should be computationally inexpensive and should find optimal or near optimal sets of features.
- An integral part of the search is an **evaluation step** to judge how the current subset of features compares to others that have been considered. This requires an evaluation measure that attempts to determine the goodness of a subset of attributes with respect to a particular data mining task, such as classification or clustering.
- Because the number of subsets can be enormous and it is impractical to examine them all, some sort of **stopping criterion** is necessary, which can be - the number of iterations, whether the value of the subset evaluation measure is optimal or exceeds a certain threshold, whether a subset of a certain size has been obtained etc.

- Finally, once a subset of features has been selected, the results of the target data mining algorithm on the selected subset should be **validated**. A straightforward evaluation approach is to run the algorithm with the full set of features and compare the full results to results obtained using the subset of features. Hopefully, the subset of features will produce results that are better than or almost as good as those produced when using all features.
- Another validation approach is to use a number of different feature selection algorithms to obtain subsets of features and then compare the results of running the data mining algorithm on each subset.
- **Feature weighting** is an alternative to keeping or eliminating features. More important features are assigned a higher weight, while less important features are given a lower weight.

5. Feature Creation

- It is frequently possible to create, from the original attributes, a new set of attributes that captures the important information in a data set much more effectively.
- Furthermore, the number of new attributes can be smaller than the number of original attributes, with all the benefits of dimensionality reduction.
- Three related methodologies for creating new attributes: feature extraction, mapping the data to a new space, and feature construction.

- **Feature Extraction** The creation of a new set of features from the original raw data is known as feature extraction.
- Consider a set of photographs, where each photograph is to be classified according to whether or not it contains a human face. The raw data is a set of pixels, and as such, is not suitable for many types of classification algorithms. However, if the data is processed to provide higher- level features, such as the presence or absence of certain types of edges and areas that are highly correlated with the presence of human faces, then a much broader set of classification techniques can be applied to this problem.
- **Mapping the Data to a New Space** A totally different view of the data can reveal important and interesting features. For example, in text classification, the input text can be mapped into a bag-of-words representation, where each word in the text is represented by a count of its frequency.

- **Feature Construction** Sometimes the features in the original data sets have the necessary information, but it is not in a form suitable for the data mining algorithm. In this situation, one or more new features constructed out of the original features can be more useful than the original features.
- Example 2.11 (Density). To illustrate this, consider a data set consisting of information about historical items that are made of a small number of materials (wood, clay, bronze, gold) and that we want to classify them with respect to the material of which they are made. In this case, a density feature constructed from the mass and volume features, i.e., density = mass/volume, would most directly yield an accurate classification.

6. Discretization and Binarization

- Transforming a continuous attribute into a categorical attribute (**discretization**)
- Transforming both continuous and discrete attributes into one or more binary attributes (**binarization**).
- If a categorical attribute has a large number of values (categories), or some values occur infrequently, then it may be beneficial for certain data mining tasks to reduce the number of categories by combining some of the values.

.

Binarization

- If there are m categorical values, then uniquely assign each original value to an integer in the interval [0, m – 1].
- Next, convert each of these m integers to a binary number.
- To illustrate, a categorical variable with 5 values {awful, poor, OK, good, great} would require three binary variables x1, x2, and x3.
- Such a transformation can cause complications, such as creating unintended relationships among the transformed attributes. For example, in Table 2.5, attributes x2 and x3 are correlated because information about the good value is encoded using both attributes.

Table 2.5. Conversion of a categorical attribute to three binary attributes.

Categorical Value	Integer Value	x_1	x_2	x_3
awful	0	0	0	0
poor	1	0	0	1
OK	2	0	1	0
good	3	0	1	1
great	4	1	0	0

Table 2.6. Conversion of a categorical attribute to five asymmetric binary attributes.

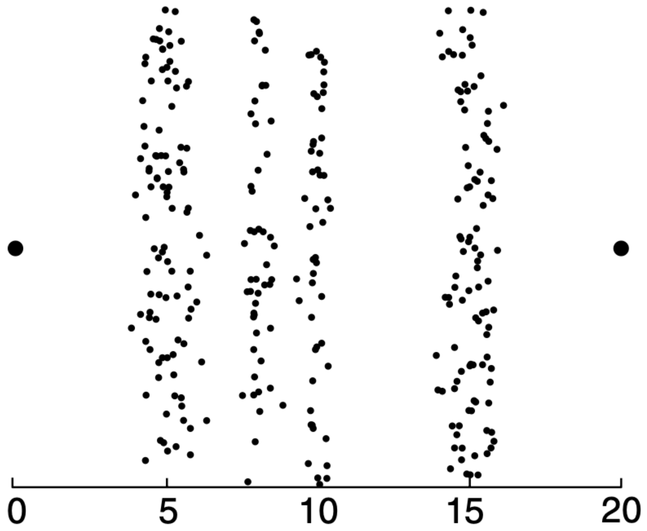
Categorical Value	Integer Value	x_1	x_2	x_3	x_4	x_5
awful	0	1	0	0	0	0
poor	1	0	1	0	0	0
OK	2	0	0	1	0	0
good	3	0	0	0	1	0
great	4	0	0	0	0	1

Discretization of Continuous Attributes

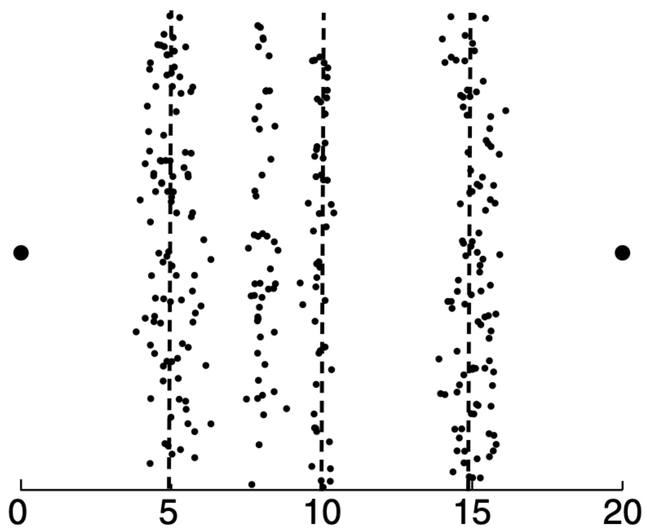
- Transformation of a continuous attribute to a categorical attribute involves two subtasks: deciding how many categories to have and determining how to map the values of the continuous attribute to these categories.
- In the first step, after the values of the continuous attribute are sorted, they are then divided into n intervals by specifying n – 1 split points.
- In the second, rather trivial step, all the values in one interval are mapped to the same categorical value.
- Therefore, the problem of discretization is one of deciding how many split points to choose and where to place them. The result can be represented either as a set of intervals $\{(x_0,x_1], (x_1,x_2], \ldots, (x_{n-1},x_n)\}$, where x_0 and x_n may be $+\infty$ or $-\infty$, respectively.

Unsupervised Discretization

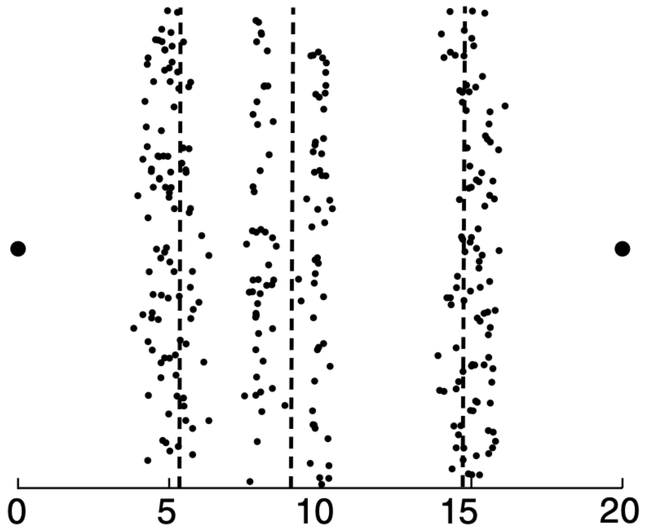
- The **equal width** approach divides the range of the attribute into a user-specified number of intervals each having the same width.
- Such an approach can be badly affected by outliers, and for that reason, an **equal frequency** (equal depth) approach, which tries to put the same number of objects into each interval, is often preferred.
- As another example of unsupervised discretization, a clustering method, such as **K-means**, can also be used.
- Finally, visually inspecting the data can sometimes be an effective approach.



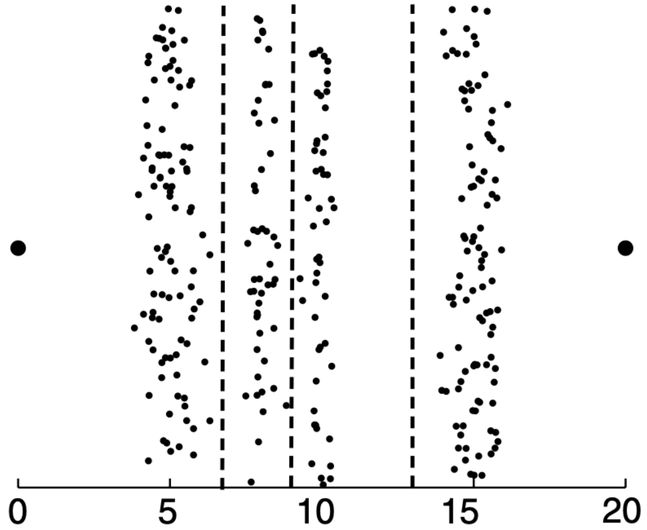
(a) Original data.



(b) Equal width discretization.



(c) Equal frequency discretization.



(d) K-means discretization.

Figure 2.13. Different discretization techniques.

Supervised Discretization

- Place the splits in a way that maximizes the purity of the intervals - minimum entropy.
- Let k be the number of different class labels, m_i be the number of values in the interval i of a partition, and m_{ij} be the number of values of class j in interval i. Then the entropy e_i of the ith interval is given by the equation $e_i = \sum_{j=1}^{k} p_{ij} \ log_2 \ p_{ij}$ where $p_{ij} = m_{ij}/m_i$ is the probability (fraction of values) of class j in the ith interval.
- The total entropy, e, of the partition is the weighted average of the individual interval entropies, i.e., $e = \sum_{i=1}^{n} w_i e_i$ where m is the number of values, $w_i = m_i/m$ is the fraction of values in the ith interval, and n is the number of intervals.
- Intuitively, the entropy of an interval is a measure of the purity of an interval. If an interval contains only values of one class (is perfectly pure), then the entropy is 0 and it contributes nothing to the overall entropy. If the classes of values in an interval occur equally often (the interval is as impure as possible), then the entropy is a maximum.

7. Variable Transformation

- A variable transformation refers to a transformation that is applied to all the values of a variable.
- In other words, for each object, the transformation is applied to the value of the variable for that object.
- For example, if only the magnitude of a variable is important, then the values of the variable can be transformed by taking the absolute value.
- **Simple functions** - For this type of variable transformation, a simple mathematical function is applied to each value individually. If x is a variable, then examples of such transformations include x^k, log x, e^x, $\sqrt{x}$, 1/x, sin x, or |x|.
- **Normalization or Standardization**

Normalization or Standardization

- If x’ is the mean (average) of the attribute values and s is their standard deviation, then the transformation x= (x – x’)/s creates a new variable that has a mean of 0 and a standard deviation of 1.
- If different variables are to be combined in some way, then such a transformation is often necessary to avoid having a variable with large values dominate the results of the calculation.
- To illustrate, consider comparing people based on two variables: age and income. For any two people, the difference in income will likely be much higher in absolute terms (hundreds or thousands of dollars) than the difference in age (less than 150). If the differ- ences in the range of values of age and income are not taken into account, then the comparison between people will be dominated by differences in income. In particular, if the similarity or dissimilarity of two people is calculated using the similarity or dissimilarity measures defined later in this chapter, then in many cases, such as that of Euclidean distance, the income values will dominate the calculation.

Dissimilarities between Data Objects

- The **Euclidean distance**, d, between two points, x and y, in one-, two-, three-, or higher- dimensional space, is given by the following familiar formula:

  $d(x, y) = \sqrt{\sum_{k=1}^{n} (x_k - y_k)^2}$ where n is the number of dimensions and x_k and y_k are, respectively, the kth attributes (components) of x and y.

- **Minkowski distance** $d(x, y) = \sum_{k=1}^{n} |x_k - y_k|^r \quad ^{1/r}$ where r is a parameter. The following are the three most common examples of Minkowski distances.

- r = 1. City block (Manhattan, taxicab, L_1 norm) distance. A common example is the Hamming distance, which is the number of bits that are different between two objects that have only binary attributes, i.e., between two binary vectors.
- r = 2. Euclidean distance (L_2 norm).
- r = ∞. Supremum (L_{max} or L_∞ norm) distance. This is the maximum difference between any attribute of the objects.
- $d(x, y) = lim_{r->\infty} \sum_{k=1}^{n} |x_k - y_k|^r$ $^{1/r}$

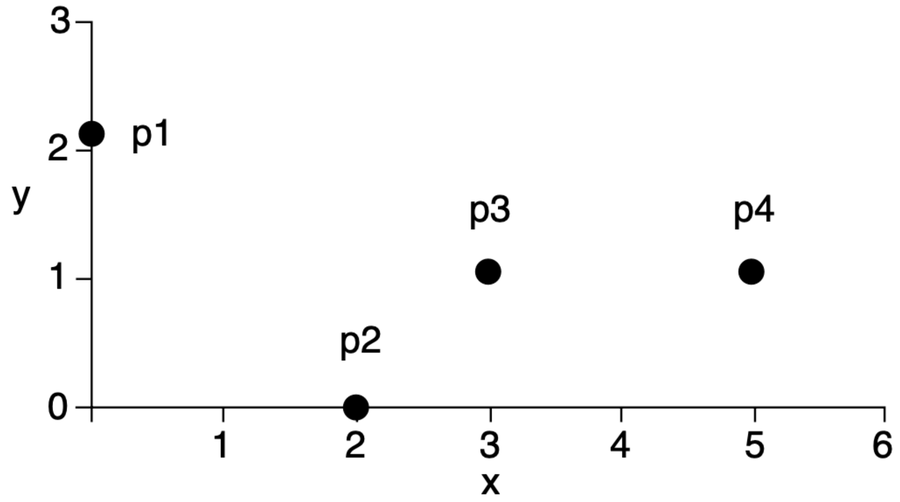


Figure 2.15. Four two-dimensional points.

Table 2.8. x and y coordinates of four points.

point	x coordinate	y coordinate
p1	0	2
p2	2	0
p3	3	1
p4	5	1

Table 2.9. Euclidean distance matrix for Table 2.8.

	p1	p2	p3	p4
p1	0.0	2.8	3.2	5.1
p2	2.8	0.0	1.4	3.2
p3	3.2	1.4	0.0	2.0
p4	5.1	3.2	2.0	0.0

Table 2.10. L_1 distance matrix for Table 2.8.

L_1	p1	p2	p3	p4
p1	0.0	4.0	4.0	6.0
p2	4.0	0.0	2.0	4.0
p3	4.0	2.0	0.0	2.0
p4	6.0	4.0	2.0	0.0

Table 2.11. L_∞ distance matrix for Table 2.8.

L_∞	p1	p2	p3	p4
p1	0.0	2.0	3.0	5.0
p2	2.0	0.0	1.0	3.0
p3	3.0	1.0	0.0	2.0
p4	5.0	3.0	2.0	0.0

- If d(x, y) is the distance between two points, x and y, then the following properties hold.

1. **Positivity**

(a) $d(x,x) \geq o$ for all x and y,

(b) $d(x,y) = o$ only if $x = y$.

2. **Symmetry**
$d(x,y) = d(y,x)$ for all x and y.

3. **Triangle Inequality**
$d(x,z) \leq d(x,y) + d(y,z)$ for all points x, y, and z.

Measures that satisfy all three properties are known as **metrics**.

- Example 2.14 (**Non-metric Dissimilarities: Set Differences**). Given two sets A and B, A – B is the set of elements of A that are not in B. For example, if A = {1,2,3,4} and B = {2,3,4}, then A–B = {1} and B – A = ∅, the empty set. We can define the distance d between two sets A and B as d(A, B) = size(A – B), where size is a function returning the number of elements in a set. This distance measure, which is an integer value greater than or equal to 0, does not satisfy the second part of the positivity property, the symmetry property, or the triangle inequality. However, these properties can be made to hold if the dissimilarity measure is modified as follows: d(A, B) = size(A – B) + size(B – A).

.

- Example 2.15 (**Non-metric Dissimilarities: Time**). Define a measure of the distance between times of the day as follows:

$$d(t_1, t_2) = \left\{ \begin{array}{ll} t_2 - t_1 & \text{if } t_1 \leq t_2 \\ 24 + (t_2 - t_1) & \text{if } t_1 \geq t_2 \end{array} \right\}.$$

- To illustrate, d(1PM, 2PM) = 1 hour, while d(2PM, 1PM) = 23 hours. Such a definition would make sense, for example, when answering the question: "If an event occurs at 1PM every day, and it is now 2PM, how long do I have to wait for that event to occur again?"

Similarities between Data Objects

- For similarities, the triangle inequality (or the analogous property) typically does not hold, but symmetry and positivity typically do.
- To be explicit, if s(x, y) is the similarity between points x and y, then the typical properties of similarities are the following:

1. s(x,y)=1 only if x=y. (0 ≤s ≤1)
2. s(x, y) = s(y, x) for all x and y. (Symmetry)

Example 2.16 (**A Non-symmetric Similarity Measure**). Consider an experiment in which people are asked to classify a small set of characters as they flash on a screen. The confusion matrix for this experiment records how often each character is classified as itself, and how often each is classified as another character. For instance, suppose that "l" appeared 200 times and was classified as a "l" 160 times, but as an "1" 40 times. Likewise, suppose that '1' appeared 200 times and was classified as an "1" 170 times, but as "l" only 30 times. If we take these counts as a measure of the similarity between two characters, then we have a similarity measure, but one that is not symmetric. In such situations, the similarity measure is often made symmetric by setting s'(x, y) = s'(y, x) = (s(x, y)+s(y, x))/2, where s' indicates the new similarity measure.

Classification

The task of assigning objects to one of several predefined categories

- The input data for a classification task is a collection of records. Each record, also known as an instance or example, is characterized by a tuple (x, y), where x is the attribute set and y is a special attribute, designated as the class label (also known as category or target attribute).
- The attribute set can also contain both discrete and continuous features.
- The class label, on the other hand, must be a discrete attribute.
- This is a key characteristic that distinguishes classification from **regression**, a predictive modeling task in which y is a continuous attribute.

Table 4.1. The vertebrate data set.

Name	Body Temperature	Skin Cover	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hibernates	Class Label
human	warm-blooded	hair	yes	no	no	yes	no	mammal
python	cold-blooded	scales	no	no	no	no	yes	reptile
salmon	cold-blooded	scales	no	yes	no	no	no	fish
whale	warm-blooded	hair	yes	yes	no	no	no	mammal
frog	cold-blooded	none	no	semi	no	yes	yes	amphibian
komodo dragon	cold-blooded	scales	no	no	no	yes	no	reptile
bat	warm-blooded	hair	yes	no	yes	yes	yes	mammal
pigeon	warm-blooded	feathers	no	no	yes	yes	no	bird
cat	warm-blooded	fur	yes	no	no	yes	no	mammal
leopard shark	cold-blooded	scales	yes	yes	no	no	no	fish
turtle	cold-blooded	scales	no	semi	no	yes	no	reptile
penguin	warm-blooded	feathers	no	semi	no	yes	no	bird
porcupine	warm-blooded	quills	yes	no	no	yes	yes	mammal
eel	cold-blooded	scales	no	yes	no	no	no	fish
salamander	cold-blooded	none	no	semi	no	yes	yes	amphibian

- Classification is the task of learning a target function f that maps each attribute set x to one of the predefined class labels y.
- The target function is also known informally as a **classification model**.
- **Descriptive Modeling** A classification model can serve as an explanatory tool to distinguish between objects of different classes - what features define a vertebrate as a mammal, reptile, bird, fish, or amphibian.
- **Predictive Modeling** A classification model can also be used to predict the class label of unknown records. A classification model can be treated as a black box that automatically assigns a class label when presented with the attribute set of an unknown record.
- Suppose we are given the following characteristics of a creature known as a gila monster:

Name	Body Temperature	Skin Cover	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hiber-nates	Class Label
gila monster	cold-blooded	scales	no	no	no	yes	yes	?

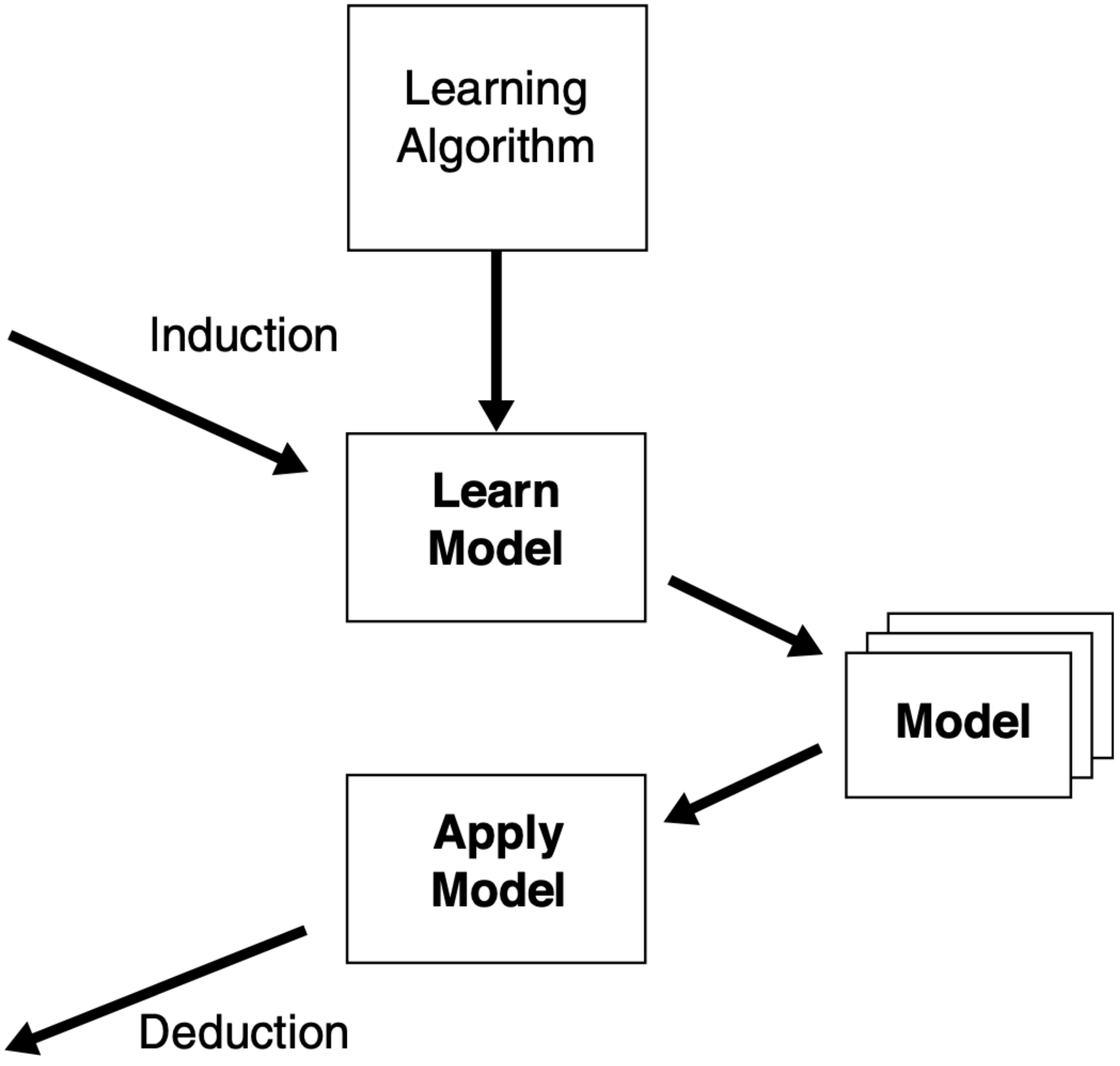
Limitations

- Classification techniques are most suited for predicting or describing data sets with binary or nominal categories.
- They are less effective for ordinal categories (e.g., to classify a person as a member of high-, medium-, or low- income group) because they do not consider the implicit order among the categories.
- Other forms of relationships, such as the subclass–superclass relationships among categories (e.g., humans and apes are primates, which in turn, is a subclass of mammals) are also ignored.

- A classification technique (or classifier) employs a learning algorithm to identify a model that best fits the relationship between the attribute set and class label of the input data.
- The model generated by a learning algorithm should both fit the input data well and correctly predict the class labels of records it has never seen before.
- Therefore, a key objective of the learning algorithm is to build models with good generalization capability; i.e., models that accurately predict the class labels of previously unknown records.
- First, a **training set** consisting of records whose class labels are known must be provided.
- The training set is used to build a classification model, which is subsequently applied to the test set, which consists of records with unknown class labels.

Training Set

Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes



Test Set

Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

Figure 4.3. General approach for building a classification model.

- Evaluation of the performance of a classification model is based on the counts of test records correctly and incorrectly predicted by the model. These counts are tabulated in a table known as a **confusion matrix**.
- Each entry f_{ij} in this table denotes the number of records from class i predicted to be of class j. For instance, f_{01} is the number of records from class 0 incorrectly predicted as class 1.
- Based on the entries in the confusion matrix, the total number of correct predictions made by the model is $(f_{11} + f_{00})$ and the total number of incorrect predictions is $(f_{10} + f_{01})$.

Performance Metric

- $Accuracy = \frac{No\ .\ of\ correct\ predictions}{Total\ no\ .\ of\ predictions} = \frac{f_{11} + f_{00}}{f_{11} + f_{00} + f_{10} + f_{01}}$
- $Error\ rate = \frac{No\ .\ of\ wrong\ predictions}{Total\ no\ .\ of\ predictions} = \frac{f_{10} + f_{01}}{f_{11} + f_{00} + f_{10} + f_{01}}$
- Most classification algorithms seek models that attain the highest accuracy, or equivalently, the lowest error rate when applied to the test set.

Decision Tree Induction

The tree has three types of nodes:

- A **root node** that has no incoming edges and zero or more outgoing edges.
- **Internal nodes**, each of which has exactly one incoming edge and two or more outgoing edges.
- **Leaf or terminal nodes**, each of which has exactly one incoming edge and no outgoing edges.

- In a decision tree, each leaf node is assigned a class label. The non- terminal nodes, which include the root and other internal nodes, contain attribute test conditions to separate records that have different characteristics.

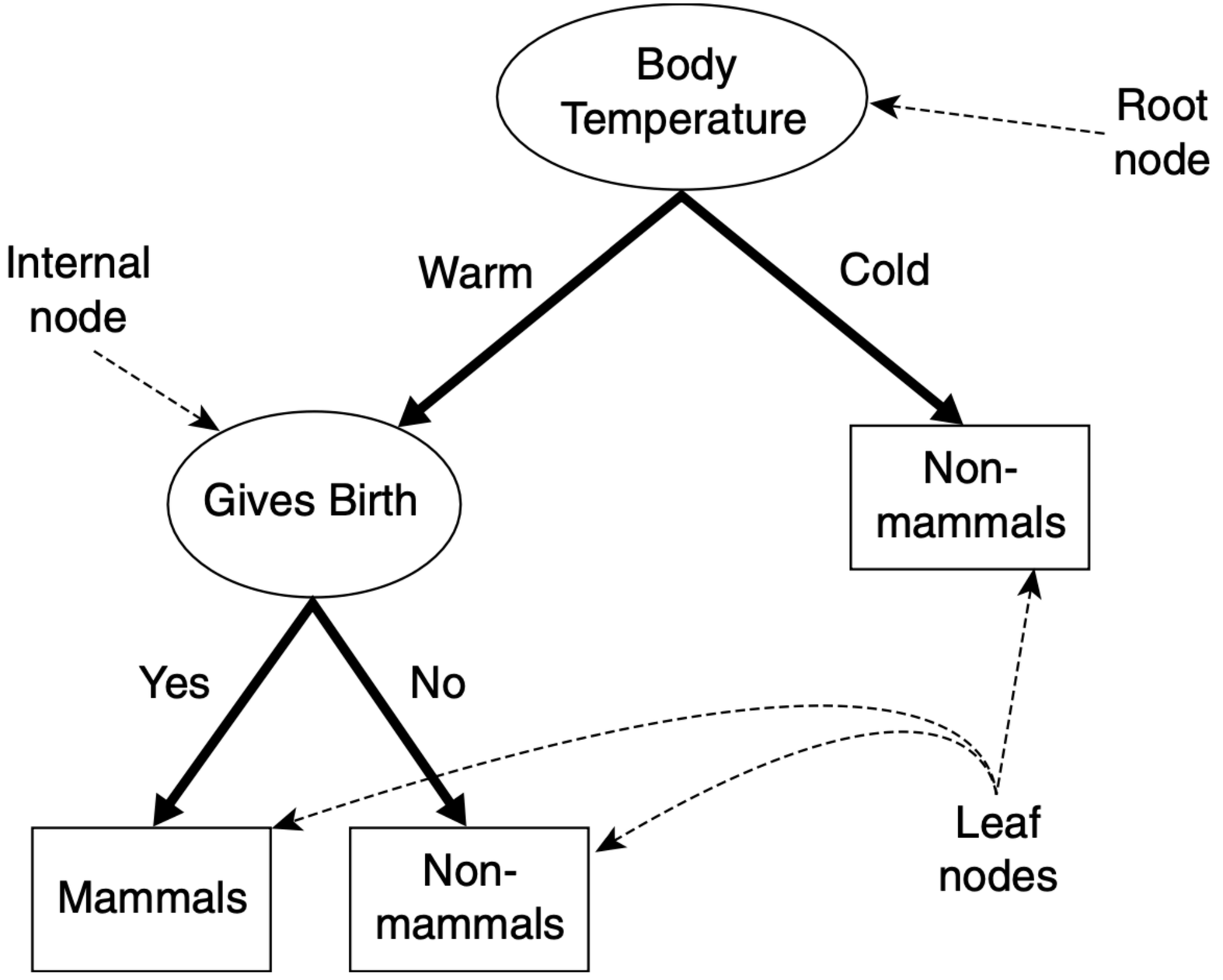


Figure 4.4. A decision tree for the mammal classification problem.

- For example, the root node shown in Figure 4.4 uses the attribute Body Temperature to separate warm-blooded from cold-blooded vertebrates. Since all cold-blooded vertebrates are non-mammals, a leaf node labeled Non-mammals is created as the right child of the root node. If the vertebrate is warm-blooded, a subsequent attribute, Gives Birth, is used to distinguish mammals from other warm-blooded creatures, which are mostly birds.
- Classifying a test record is straightforward once a decision tree has been constructed.
- Starting from the root node, we apply the test condition to the record and follow the appropriate branch based on the outcome of the test. This will lead us either to another internal node, for which a new test condition is applied, or to a leaf node.
- The class label associated with the leaf node is then assigned to the record. As an illustration, Figure 4.5 traces the path in the decision tree that is used to predict the class label of a flamingo. The path terminates at a leaf node labeled Non-mammals.

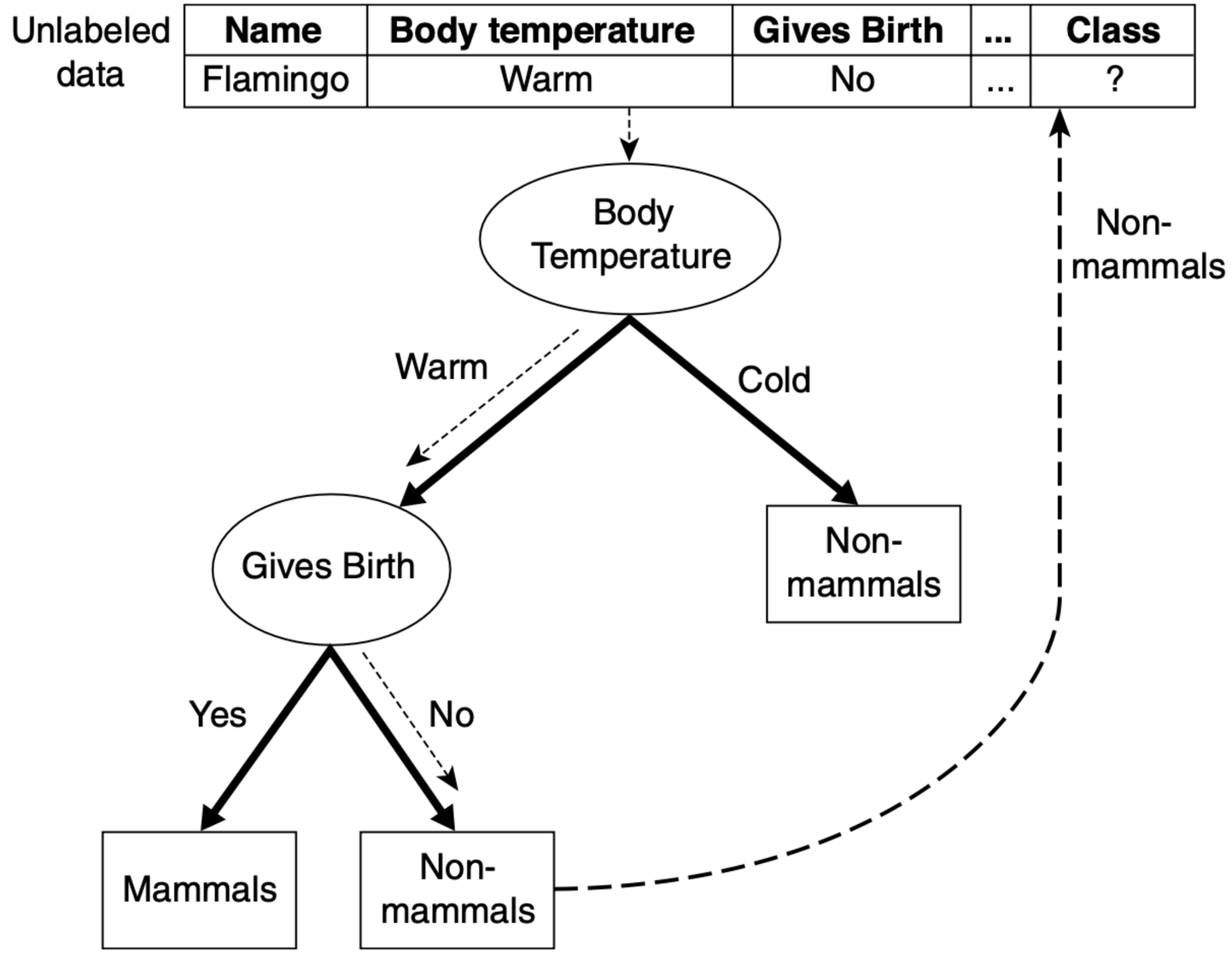


Figure 4.5. Classifying an unlabeled vertebrate. The dashed lines represent the outcomes of applying various attribute test conditions on the unlabeled vertebrate. The vertebrate is eventually assigned to the `Non-mammal` class.

Hunt's Algorithm

- In Hunt's algorithm, a decision tree is grown in a recursive fashion by parti- tioning the training records into successively purer subsets. Let D_t be the set of training records that are associated with node t and $y = \{y_1, y_2, \ldots, y_c\}$ be the class labels. The following is a recursive definition of Hunt's algorithm.
- Step 1: If all the records in D_t belong to the same class y_t , then t is a leaf node labeled as y_t.
- Step 2: If D_t contains records that belong to more than one class, an attribute test condition is selected to partition the records into smaller subsets. A child node is created for each outcome of the test condition and the records in D_t are distributed to the children based on the outcomes. The algorithm is then recursively applied to each child node.

	binary	categorical	continuous	class
Tid	Home Owner	Marital Status	Annual Income	Defaulted Borrower
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Figure 4.6. Training set for predicting borrowers who will default on loan payments.

To illustrate how the algorithm works, consider the problem of predicting whether a loan applicant will repay her loan obligations or become delinquent, subsequently defaulting on her loan.

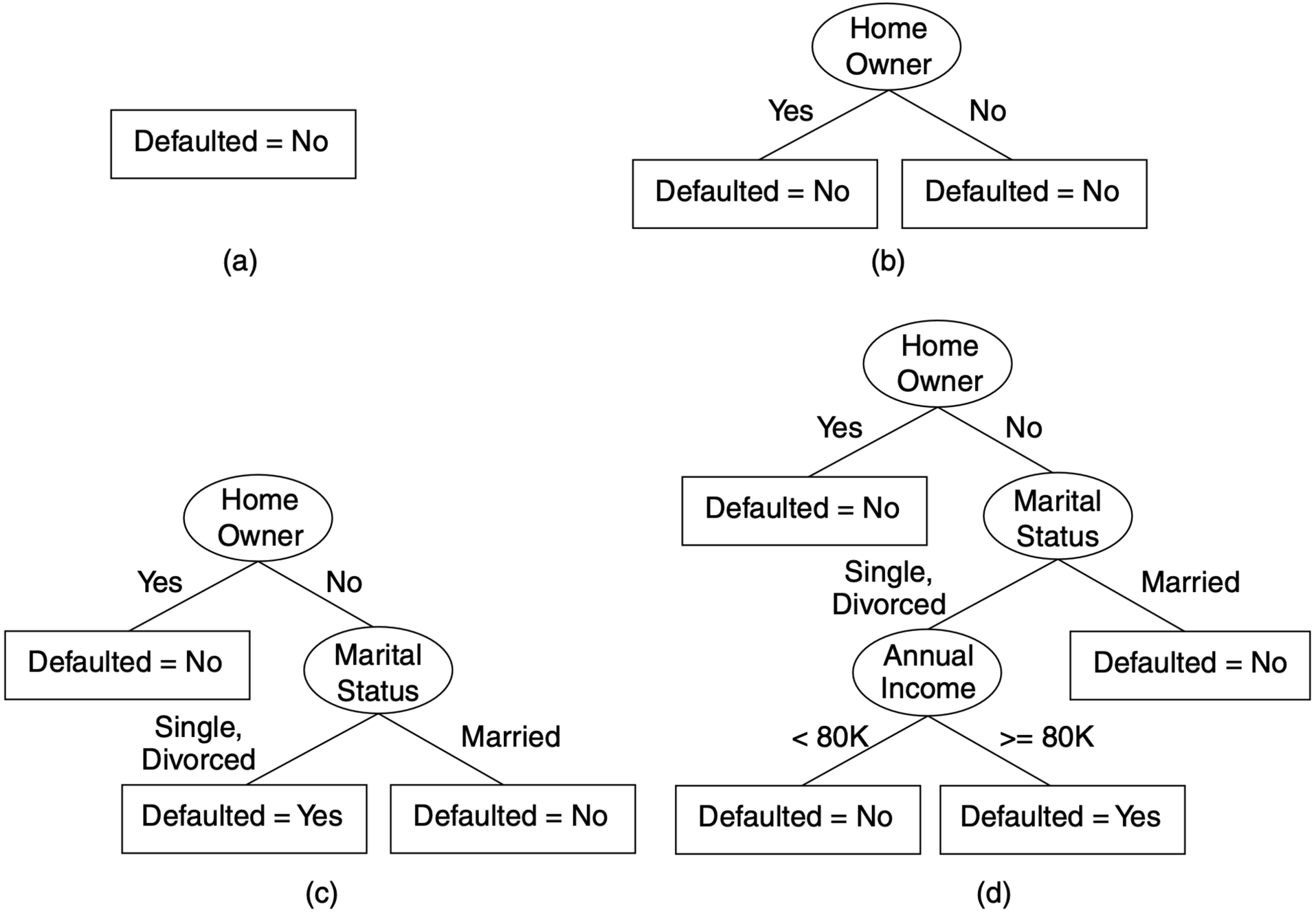


Figure 4.7. Hunt's algorithm for inducing decision trees.

- The initial tree for the classification problem contains a single node with class label Defaulted = No, which means that most of the borrowers successfully repaid their loans. The tree, however, needs to be refined since the root node contains records from both classes.
- The records are subsequently divided into smaller subsets based on the outcomes of the Home Owner test condition (assuming that this is the best criterion for splitting the data at this point).
- Hunt's algorithm is then applied recursively to each child of the root node. From the training set, notice that all borrowers who are home owners successfully repaid their loans. The left child of the root is therefore a leaf node labeled Defaulted = No.
- For the right child, we need to continue applying the recursive step of Hunt's algorithm until all the records belong to the same class.

.

Hunt's algorithm will work if every combination of attribute values is present in the training data and each combination has a unique class label. Additional conditions are needed to handle the following cases:

1. It is possible for some of the child nodes created in Step 2 to be empty; i.e., there are no records associated with these nodes. This can happen if none of the training records have the combination of attribute values associated with such nodes. In this case the node is declared a leaf node with the same class label as the majority class of training records associated with its parent node.

2. In Step 2, if all the records associated with D_t have identical attribute values (except for the class label), then it is not possible to split these records any further. In this case, the node is declared a leaf node with the same class label as the majority class of training records associated with this node.

- **How should the training records be split?** Each recursive step of the tree-growing process must select an attribute test condition to divide the records into smaller subsets. To implement this step, the algorithm must provide a method for specifying the test condition for different attribute types as well as an objective measure for evaluating the goodness of each test condition.
- **How should the splitting procedure stop?** A stopping condition is needed to terminate the tree-growing process. A possible strategy is to continue expanding a node until either all the records belong to the same class or all the records have identical attribute values. Although both conditions are sufficient to stop any decision tree induction algorithm, other criteria can be imposed to allow the tree-growing procedure to terminate earlier.

.

- **Binary Attributes** The test condition for a binary attribute generates two potential outcomes.
- **Nominal Attributes** Since a nominal attribute can have many values, its test condition can be expressed in two ways. For a **multiway split**, the number of outcomes depends on the number of distinct values for the corresponding attribute. For example, if an attribute such as marital status has three distinct values —single, married, or divorced—its test condition will produce a three-way split. On the other hand, some decision tree algorithms, such as CART, produce only binary splits by considering all $2^{k-1} - 1$ ways of creating a binary partition of k attribute values.
- **Ordinal Attributes** Ordinal attributes can also produce binary or multiway splits. Ordinal attribute values can be grouped as long as the grouping does not violate the order property of the attribute values. The grouping shown in Figure 4.10(c) violates this property because it combines the attribute values Small and Large into the same partition while Medium and Extra Large are combined into another partition.

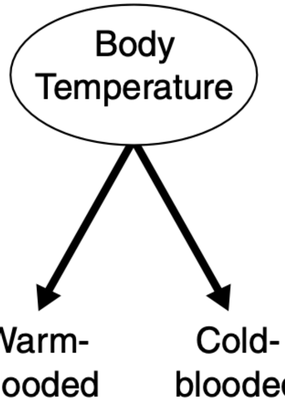
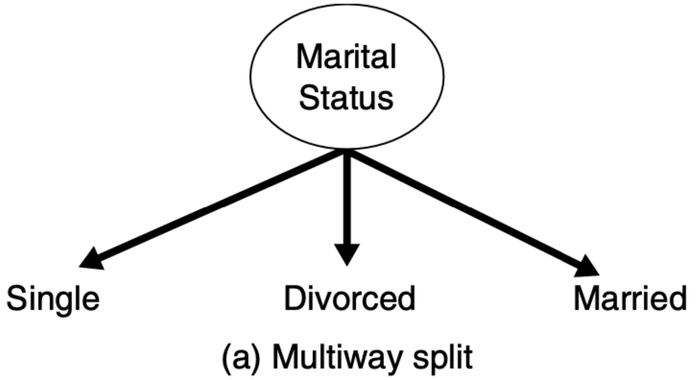
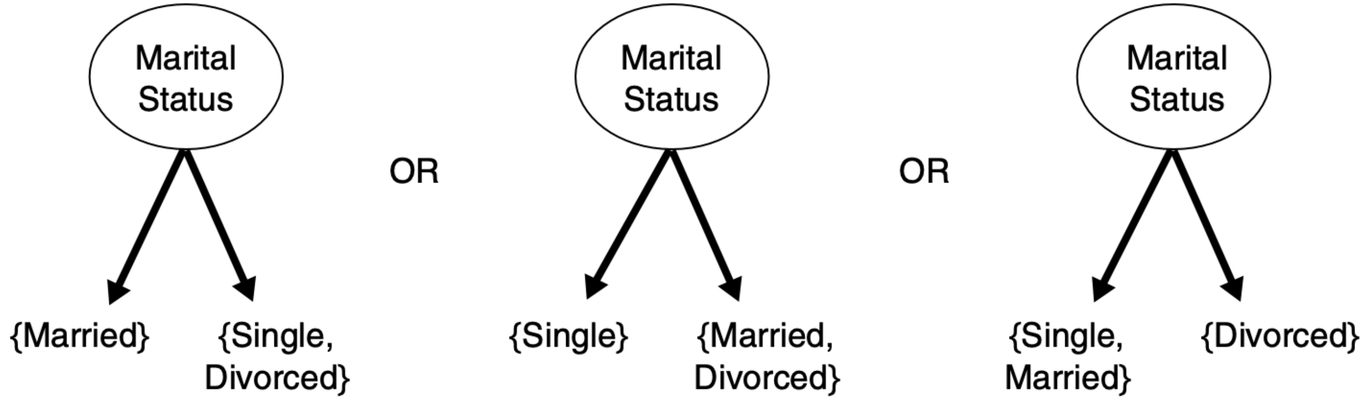


Figure 4.8. Test condition for binary attributes.



(a) Multiway split



(b) Binary split {by grouping attribute values}

Figure 4.9. Test conditions for nominal attributes.

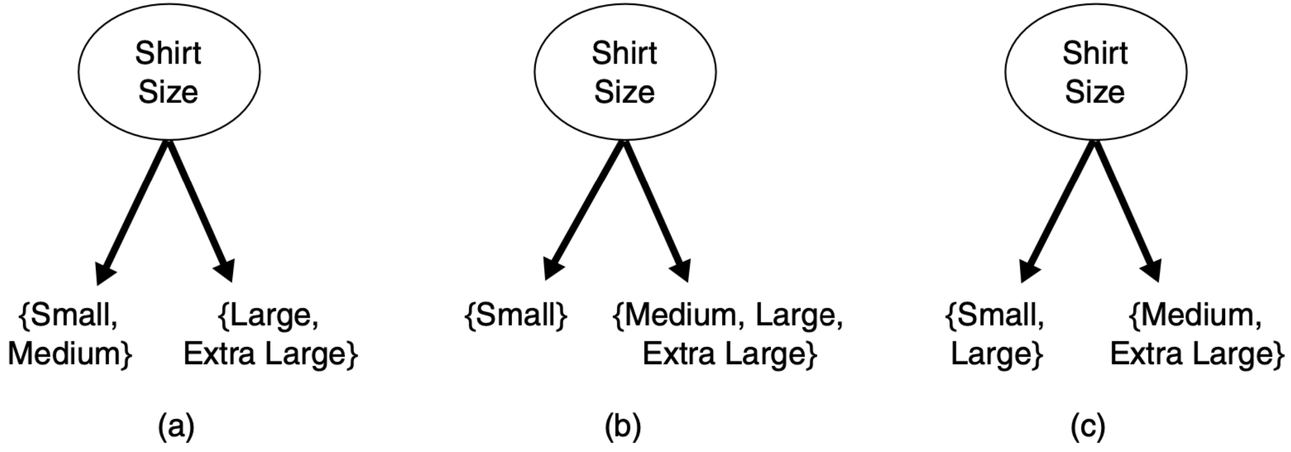


Figure 4.10. Different ways of grouping ordinal attribute values.

- **Continuous Attributes** For continuous attributes, the test condition can be expressed as a comparison test (A < v) or (A ≥ v) with binary outcomes, or a range query with outcomes of the form $v_i \leq A < v_{i+1}$, for i = 1,...,k.
- For the binary case, the decision tree algorithm must consider all possible split positions v, and it selects the one that produces the best partition.
- For the multiway split, the algorithm must consider all possible ranges of continuous values. One approach is to apply the discretization after which, a new ordinal value will be assigned to each discretized interval.

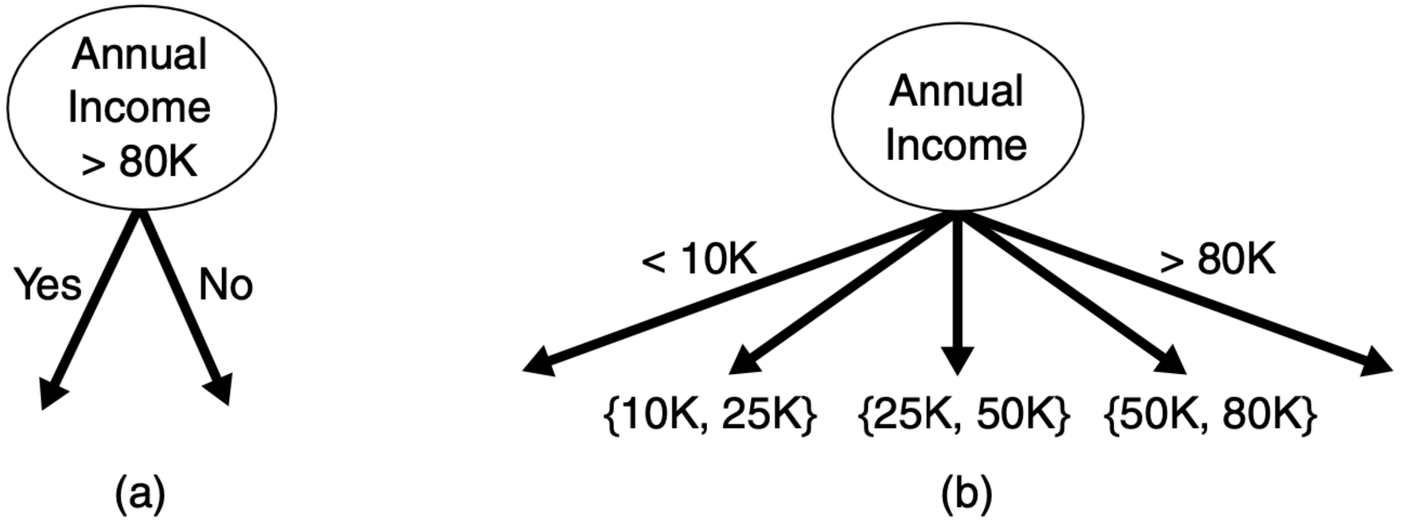


Figure 4.11. Test condition for continuous attributes.

Customer ID	Gender	Car Type	Shirt Size	Class
1	M	Family	Small	C0
2	M	Sports	Medium	C0
3	M	Sports	Medium	C0
4	M	Sports	Large	C0
5	M	Sports	Extra Large	C0
6	M	Sports	Extra Large	C0
7	F	Sports	Small	C0
8	F	Sports	Small	C0
9	F	Sports	Medium	C0
10	F	Luxury	Large	C0
11	M	Family	Large	C1
12	M	Family	Extra Large	C1
13	M	Family	Medium	C1
14	M	Luxury	Extra Large	C1
15	F	Luxury	Small	C1
16	F	Luxury	Small	C1
17	F	Luxury	Medium	C1
18	F	Luxury	Medium	C1
19	F	Luxury	Medium	C1
20	F	Luxury	Large	C1

Measures for selecting best split

- Let p_i denote the fraction of records belonging to class i at a given node t.
- In a two-class problem, the class distribution at any node can be written as (p_0, p_1), where $p_1 = 1 - p_0$.
- The class distribution before splitting is (0.5,0.5) because there are an equal number of records from each class. If we split the data using the Gender attribute, then the class distributions of the child nodes are (0.6, 0.4) and (0.4, 0.6), respectively.
- Although the classes are no longer evenly distributed, the child nodes still contain records from both classes.
- Splitting on the second attribute, Car Type, will result in purer partitions.

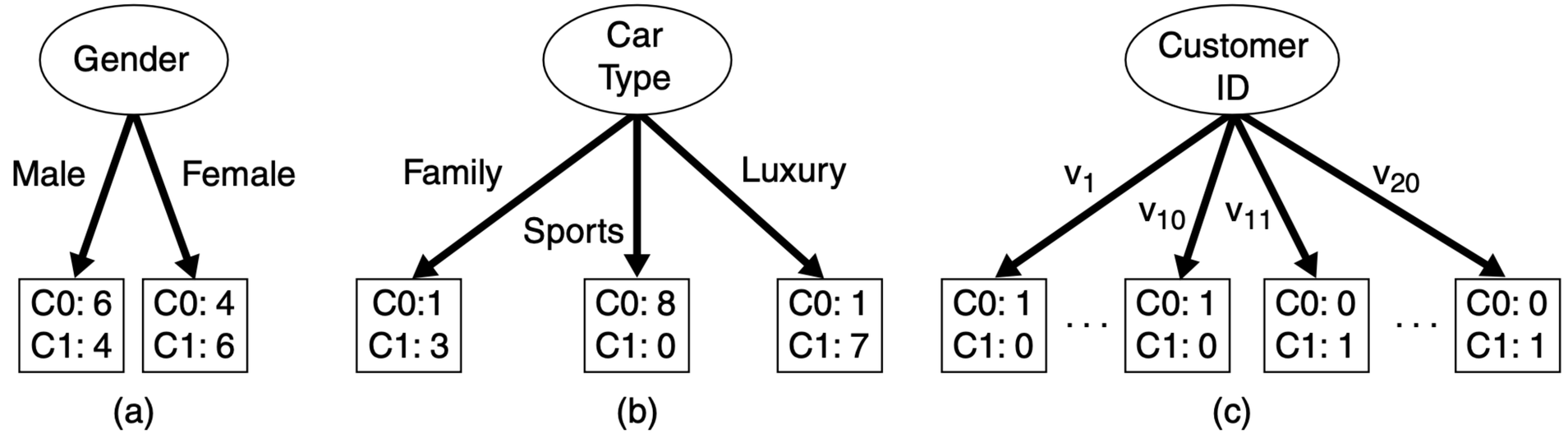


Figure 4.12. Multiway versus binary splits.

- The measures developed for selecting the best split are often based on the **degree of impurity** of the child nodes. The smaller the degree of impurity, the more skewed the class distribution. For example, a node with class distribution (0,1) has zero impurity, whereas a node with uniform class distribution (0.5, 0.5) has the highest impurity.

Impurity Measures

$$\begin{aligned}
\text{Entropy}(t) &= -\sum_{i=0}^{c-1} p(i|t) \log_2 p(i|t), \\
\text{Gini}(t) &= 1 - \sum_{i=0}^{c-1} [p(i|t)]^2, \\
\text{Classification error}(t) &= 1 - \max_i [p(i|t)],
\end{aligned}$$

where c is the number of classes.

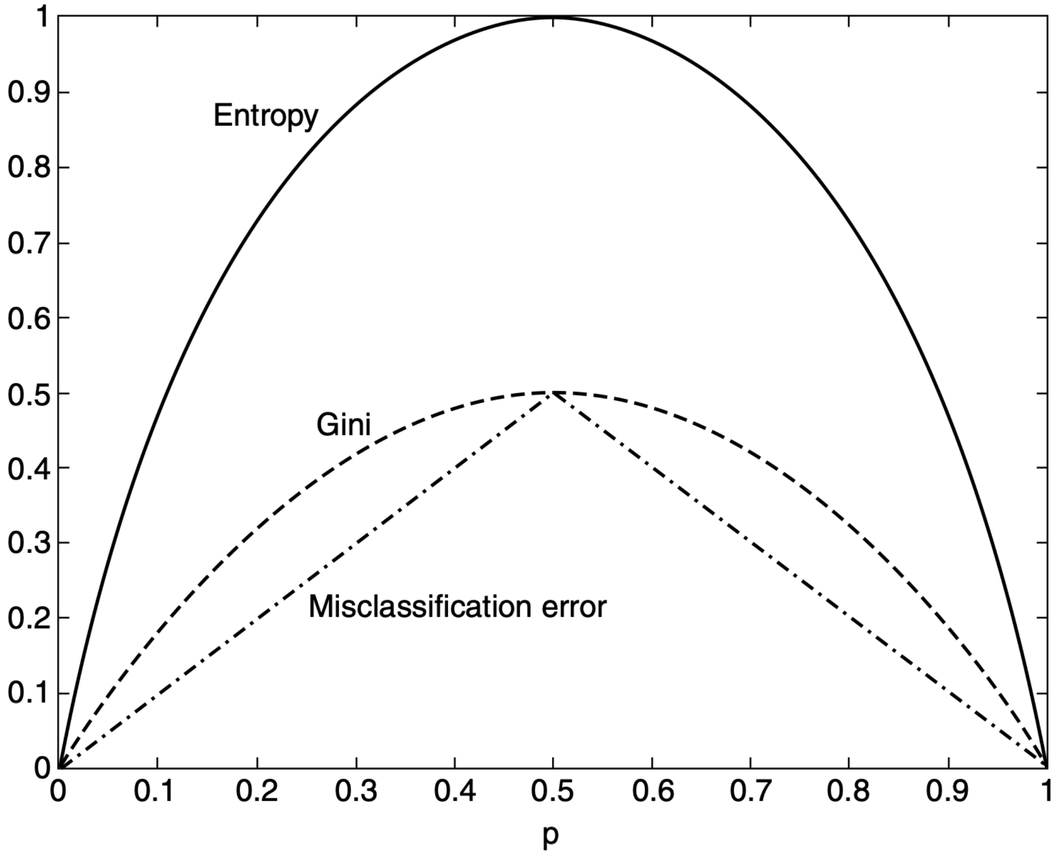


Figure 4.13. Comparison among the impurity measures for binary classification problems.

- Figure 4.13 compares the values of the impurity measures for binary classification problems. p refers to the fraction of records that belong to one of the two classes.
- All three measures attain their maximum value when the class distribution is uniform (i.e., when p = 0.5).
- The minimum values for the measures are attained when all the records belong to the same class (i.e., when p equals 0 or 1).

Node N_1	Count
Class=0	0
Class=1	6

$$\text{Gini} = 1 - (0/6)^2 - (6/6)^2 = 0$$
$$\text{Entropy} = -(0/6)\log_2(0/6) - (6/6)\log_2(6/6) = 0$$
$$\text{Error} = 1 - \max[0/6, 6/6] = 0$$

Node N_2	Count
Class=0	1
Class=1	5

$$\text{Gini} = 1 - (1/6)^2 - (5/6)^2 = 0.278$$
$$\text{Entropy} = -(1/6)\log_2(1/6) - (5/6)\log_2(5/6) = 0.650$$
$$\text{Error} = 1 - \max[1/6, 5/6] = 0.167$$

Node N_3	Count
Class=0	3
Class=1	3

$$\text{Gini} = 1 - (3/6)^2 - (3/6)^2 = 0.5$$
$$\text{Entropy} = -(3/6)\log_2(3/6) - (3/6)\log_2(3/6) = 1$$
$$\text{Error} = 1 - \max[3/6, 3/6] = 0.5$$

- Node N1 has the lowest impurity value, followed by N2 and N3.
- To determine how well a test condition performs, we need to compare the degree of impurity of the parent node (before splitting) with the degree of impurity of the child nodes (after splitting). The larger their difference, the better the test condition. The gain, Δ, is a criterion that can be used to determine the goodness of a split:
- $$\Delta = I(parent) - \sum_{j=1}^{k} \frac{N(v_j)}{N} I(v_j)$$
- where I(·) is the impurity measure of a given node, N is the total number of records at the parent node, k is the number of attribute values, and $N(v_j)$ is the number of records associated with the child node, v_j.
- Decision tree induction algorithms often choose a test condition that maximizes the gain Δ.
- When entropy is used as the impurity measure in Equation, the difference in entropy is known as the **information gain**, Δ_{info}

Splitting of Binary Attributes

- Suppose there are two ways to split the data into smaller subsets. Before splitting, the Gini index is 0.5 since there are an equal number of records from both classes.
- If attribute A is chosen to split the data, the Gini index for node N1 is 0.4898, and for node N2, it is 0.480.
- The weighted average of the Gini index for the descendent nodes is (7/12) × 0.4898 + (5/12) × 0.480 = 0.486.
- Similarly, we can show that the weighted average of the Gini index for attribute B is 0.375.
- Since the subsets for attribute B have a smaller Gini index, it is preferred over attribute A.

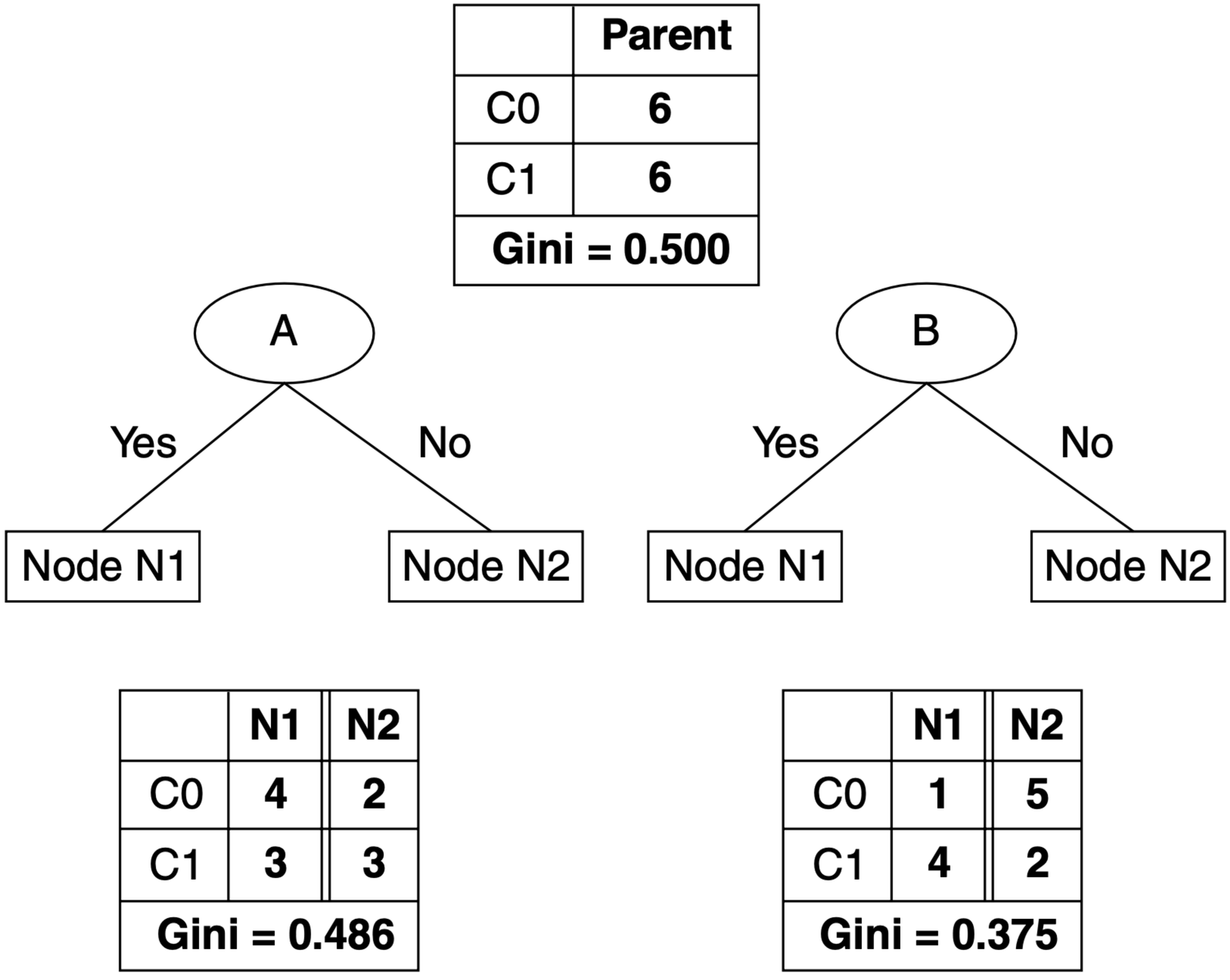
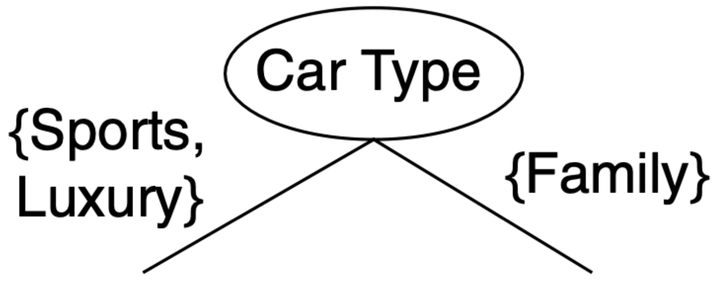
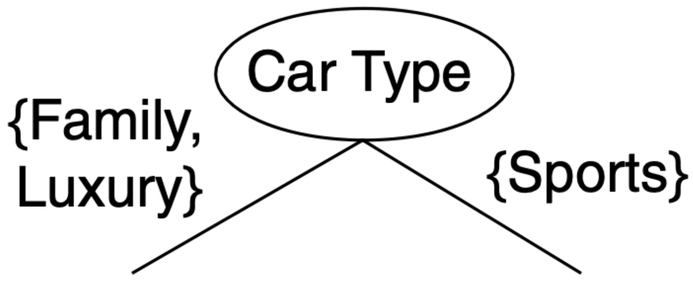


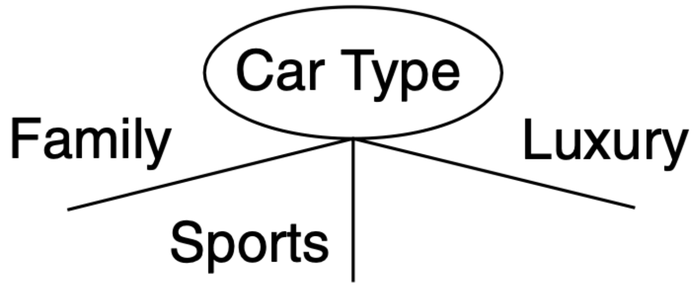
Figure 4.14. Splitting binary attributes.

Splitting of Nominal Attributes

- A nominal attribute can produce either binary or multi- way splits.
- The computation of the Gini index for a binary split is similar to that shown for determining binary attributes.
- For the first binary grouping of the Car Type attribute, the Gini index of {Sports, Luxury} is 0.4922 and the Gini index of {Family} is 0.3750.
- The weighted average Gini index for the grouping is equal to $16/20 \times 0.4922 + 4/20 \times 0.3750 = 0.468$.
- Similarly, for the second binary grouping of {Sports} and {Family, Luxury}, the weighted average Gini index is 0.167. The second grouping has a lower Gini index because its corresponding subsets are much purer.







	Car Type	
	{Sports, Luxury}	{Family}
C0	9	1
C1	7	3
Gini	0.468	

	Car Type	
	{Sports}	{Family, Luxury}
C0	8	2
C1	0	10
Gini	0.167	

(a) Binary split

	Car Type		
	Family	Sports	Luxury
C0	1	8	1
C1	3	0	7
Gini	0.163		

(b) Multiway split

Figure 4.15. Splitting nominal attributes.

- For the multiway split, the Gini index is computed for every attribute value. Since Gini({Family}) = 0.375, Gini({Sports}) = 0, and Gini({Luxury}) = 0.219, the overall Gini index for the multiway split is equal to 4/20×0.375+8/20×0+8/20×0.219 = 0.163.
- The multiway split has a smaller Gini index compared to both two-way splits. This result is not surprising because the two-way split actually merges some of the outcomes of a multiway split, and thus, results in less pure subsets.

Splitting of Continuous Attributes

Class	No	No	No	Yes	Yes	Yes	No	No	No	No
	Annual Income									
Sorted Values →	60	70	75	85	90	95	100	120	125	220
Split Positions →	55	65	72	80	87	92	97	110	122	172	230
	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >
Yes	0 3	0 3	0 3	0 3	1 2	2 1	3 0	3 0	3 0	3 0	3 0
No	0 7	1 6	2 5	3 4	3 4	3 4	3 4	4 3	5 2	6 1	7 0
Gini	0.420	0.400	0.375	0.343	0.417	0.400	*0.300*	0.343	0.375	0.400	0.420

Figure 4.16. Splitting continuous attributes.

Consider the example, in which the test condition Annual Income ≤ v is used to split the training records for the loan default classification problem.

- A brute-force method for finding v is to consider every value of the attribute in the N records as a candidate split position.
- For each candidate v, the data set is scanned once to count the number of records with annual income less than or greater than v.
- We then compute the Gini index for each candidate and choose the one that gives the lowest value.
- This approach is computationally expensive because it requires O(N) operations to compute the Gini index at each candidate split position. Since there are N candidates, the overall complexity of this task is O(N^2).
- To reduce the complexity, the training records are sorted based on their annual income, a computation that requires O(N log N) time. Candidate split positions are identified by taking the midpoints between two adjacent sorted values: 55, 65, 72, and so on.
- However, unlike the brute-force approach, we do not have to examine all N records when evaluating the Gini index of a candidate split position.

- For the first candidate, v = 55, none of the records has annual income less than $55K. As a result, the Gini index for the descendent node with Annual Income < $55K is zero.
- On the other hand, the number of records with annual income greater than or equal to $55K is 3 (for class Yes) and 7 (for class No), respectively. Thus, the Gini index for this node is 0.420. The overall Gini index for this candidate split position is equal to $0 \times 0 + 1 \times 0.420 = 0.420$.
- For the second candidate, v = 65, we can determine its class distribution by updating the distribution of the previous candidate. More specifically, the new distribution is obtained by examining the class label of the record with the lowest annual income (i.e., $60K). Since the class label for this record is No, the count for class No is increased from 0 to 1 (for Annual Income ≤ $65K) and is decreased from 7 to 6 (for Annual Income > $65K). The distribution for class Yes remains unchanged. The new weighted-average Gini index for this candidate split position is 0.400.

.

- Impurity measures such as entropy and Gini index tend to favor attributes that have a large number of distinct values.
- Comparing the first test condition, Gender, with the second, Car Type, it is easy to see that Car Type seems to provide a better way of splitting the data since it produces purer descendent nodes.
- However, if we compare both conditions with Customer ID, the latter appears to produce purer partitions. Yet Customer ID is not a predictive attribute because its value is unique for each record.
- Even in a less extreme situation, a test condition that results in a large number of outcomes may not be desirable because the number of records associated with each partition is too small to enable us to make any reliable predictions.

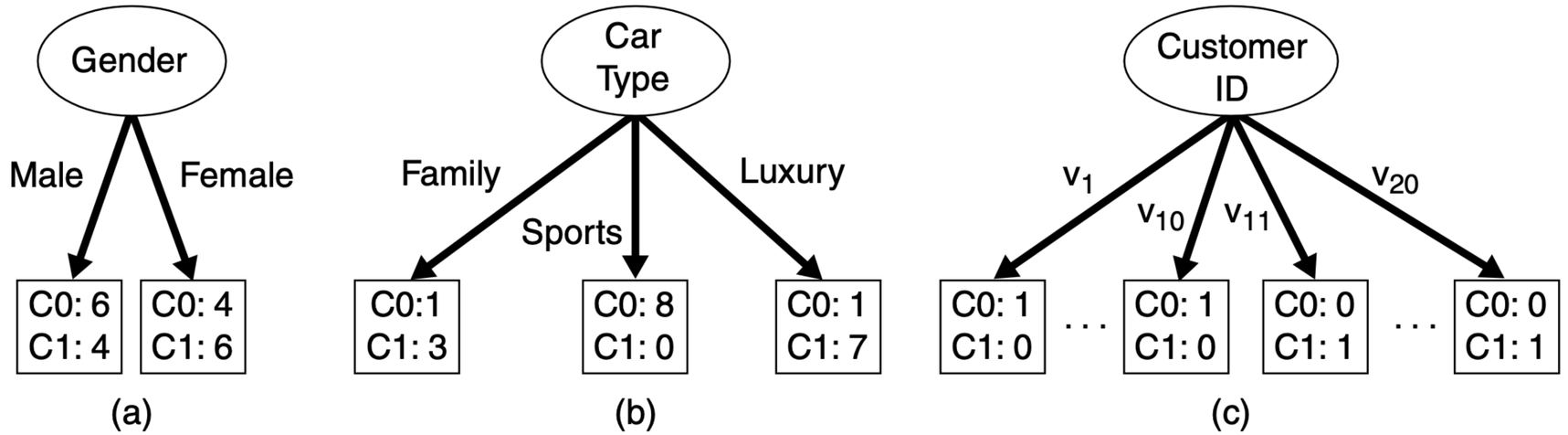


Figure 4.12. Multiway versus binary splits.

- There are two strategies for overcoming this problem. The first strategy is to restrict the test conditions to binary splits only. This strategy is employed by decision tree algorithms such as CART.
- Another strategy is to modify the splitting criterion to take into account the number of outcomes produced by the attribute test condition.

$$Gain\ ratio = \frac{\Delta_{info}}{Split\ info}$$

$$Split\ info = -\sum_{i=1}^{k} P(v_i)\ log_2 P(v_i)$$

where k is the total number of splits, v_i represents node i, $P(v_i)$ is the probability of a record belonging to node v_i

- The feature with the highest gain ratio is considered as the best feature for splitting.
- If each attribute value has the same number of records, then $\forall$i : $P(v_i) = 1/k$ and the split information would be equal to $log_2 k$.
- This example suggests that if an attribute produces a large number of splits, its split information will also be large, which in turn reduces its gain ratio.

	Temperature	Humidity	Wind	PlayTennis
2	Hot	High	Weak	No
3	Hot	High	Strong	No
4	Mild	High	Weak	Yes
5	Cool	Normal	Weak	Yes
6	Cool	Normal	Strong	No

- Let's consider a toy dataset with 3 attributes: "Temperature", "Humidity" and "Wind". The target feature is "Play Tennis" which can have two outcomes: "Yes" or "No".
- To calculate the Split Info for the attribute "Temperature", we first find the total number of instances in the dataset, which is 5. Then we find the number of instances for each outcome of the "Temperature" attribute, which is: Hot: 2 instances, Mild: 1 instance, Cool: 2 instances
- Split Info(Temperature) = — (2/5) * log2(2/5) — (1/5) * log2(1/5) — (2/5) * log2(2/5) which is approximately equal to 0.918

	Fruit	Color	Shape	Weight	Bitten
1	1	Red	Round	Heavy	No
2	2	Red	Round	Light	Yes
3	3	Yellow	Round	Heavy	Yes
4	4	Yellow	Round	Light	Yes
5	5	Yellow	Round	Light	No
6	6	Yellow	Square	Heavy	No
7	7	Orange	Round	Heavy	No
8	8	Orange	Square	Heavy	No
9	9	Orange	Square	Light	Yes
10	10	Orange	Square	Light	No

- Determine the best feature for splitting to build a decision tree using gain ratio.

1. Calculate the initial entropy of the target variable (Bitten) using the formula: Entropy = — p(Yes) log2 p(Yes) — p(No) log2 p(No)

p(Yes) = 3/10 = 0.3 (3 fruits are bitten),

p(No) = 7/10 = 0.7 (7 fruits are not bitten)

Entropy(Parent) = — (0.3) *log2 (0.3) — (0.7) *log2 (0.7) = 0.881

2. The information gain can be calculated by comparing the entropy of the parent node to the weighted average of the entropies of the child nodes.

Information Gain (feature) = Entropy(Parent) — Weighted Average of Entropies(Children)

Now, For each feature (Color, Shape, Weight), calculate the information gain.

- Colour has 3 possible values (Red, Yellow, Orange). Calculate the entropies of the three child nodes red, yellow and orange and take the weighted sum.
- Entropy(Red) = — p(Yes|Red) *log2 p(Yes|Red) — p(No|Red) *log2 p(No|Red)
- Entropy(Red) = — (1/2) *log2 (1/2) — (1/2) *log2 (1/2) = 1
- Entropy(Yellow) = — (2/4) *log2 (2/4) — (2/4)* log2 (2/4) = 1
- Entropy(Orange) = — (1/4) *log2 (1/4) — (3/4) *log2 (3/4) = 0.811
- Weighted Average of Entropies(Children)= (2/10) * (1) + (4/10) * (1) + (4/10) * (0.811) = 0.922
- Information Gain (Color) = 0.881–0.922 = -0.041

- Shape has 2 possible values (Round, Square).
- Entropy(Round) = — (3/6) *log2 (3/6) — (3/6) *log2 (3/6) = 1
- Entropy(Square) = — (3/4) *log2 (3/4) — (1/4) *log2 (1/4) = 0.22+0.35 = 0.57
- Weighted Average of Entropies(Children)= (6/10) * (1) + (4/10) * (0.57) = 0.6+0.228= 0.828
- Information Gain (Shape) = 0.881–0.828 = 0.053
- Weight has 2 possible values (Heavy, Light).
- Entropy(Heavy) = — (4/5) *log2 (4/5) — (1/5) *log2 (1/5) = 0.18 + 0.32 = 0.5
- Entropy(Light) = — (2/5) *log2 (2/5) — (3/5) *log2 (3/5) = 0.37+0.31 = 0.68
- Weighted Average of Entropies(Children)= (5/10) * (0.5) + (5/10) * (0.68) = 0.25+0.34= 0.59
- Information Gain (Weight) = 0.881–0.59= 0.291

3. For each feature, calculate the split info

Split Info (Color) = — (0.2) *log2 (0.2) — (0.4) *log2 (0.4) — (0.4) *log2 (0.4) = 1.371

Split Info (Shape) = — (0.6) *log2 (0.6) — (0.4) *log2 (0.4) = 0.31 + 0.37 = 0.68

Split Info (Weight) = — (0.5) *log2 (0.5) — (0.5) *log2 (0.5) = 1

4. Divide the information gain by the split info for each feature to calculate the gain ratio.

Gain Ratio (Color) = Information Gain (Color) / Split Info (Color) = -0.041 / 1.371 = -0.03

Gain Ratio (Shape) = Information Gain (Shape) / Split Info (Shape) = 0.053/ 0.068 = 0.779

Gain Ratio (Weight) = Information Gain (Weight) / Split Info (Weight) = 0.291 / 1 = 0.291

Shape has the highest gain ratio. Hence it is the best feature to split.

Algorithm

Algorithm 4.1 A skeleton decision tree induction algorithm.

`TreeGrowth` (E, F)

1: **if** `stopping_cond`$(E,F) = true$ **then**
2: $leaf =$ `createNode`().
3: $leaf.label =$ `Classify`(E).
4: return $leaf$.
5: **else**
6: $root =$ `createNode`().
7: $root.test\_cond =$ `find_best_split`(E, F).
8: let $V = \{v | v$ is a possible outcome of $root.test\_cond\ \}$.
9: **for** each $v \in V$ **do**
10: $E_v = \{e \mid root.test\_cond(e) = v$ and $e \in E\}$.
11: $child =$ `TreeGrowth`(E_v, F).
12: add $child$ as descendent of $root$ and label the edge $(root \rightarrow child)$ as v.
13: **end for**
14: **end if**
15: return $root$.

- The input to this algorithm consists of the training records E and the attribute set F.
- The algorithm works by recursively selecting the best attribute to split the data (Step 7) and expanding the leaf nodes of the tree (Steps 11 and 12) until the stopping criterion is met (Step 1).

1. The **createNode()** function extends the decision tree by creating a new node. A node in the decision tree has either a test condition, denoted as node.test cond, or a class label, denoted as node.label.
2. The **find_best split()** function determines which attribute should be selected as the test condition for splitting the training records. The choice of test condition depends on which impurity measure is used to determine the goodness of a split.
3. The **Classify()** function determines the class label to be assigned to a leaf node. For each leaf node t, let p(i|t) denote the fraction of training records from class i associated with the node t. In most cases, the leaf node is assigned to the class that has the majority number of training records:
   leaf.label = argmax p(i|t),
   where the argmax operator returns the argument i that maximizes the expression p(I|t).
4. The **stopping_cond()** function is used to terminate the tree-growing process by testing whether all the records have either the same class label or the same attribute values. Another way to terminate the recursive function is to test whether the number of records have fallen below some minimum threshold.

Evaluating the Performance of a Classifier

- The estimated error helps the learning algorithm to do model selection; i.e., to find a model of the right complexity that is not susceptible to overfitting.
- Once the model has been constructed, it can be applied to the test set to predict the class labels of previously unseen records.
- The accuracy or error rate computed from the test set can also be used to compare the relative performance of different classifiers on the same domain.

1. Holdout method
2. Random Subsampling
3. Cross validation
4. Bootstrap

1. Holdout method

- In the holdout method, the original data with labeled examples is partitioned into two disjoint sets, called the **training** and the **test** sets, respectively.
- A classification model is then induced from the training set and its performance is evaluated on the test set.
- The proportion of data reserved for training and for testing is typically at the discretion of the analysts (e.g., 50-50 or two- thirds for training and one-third for testing).
- The accuracy of the classifier can be estimated based on the accuracy of the induced model on the test set.
- The holdout method has several well-known limitations.
- First, fewer labeled examples are available for training because some of the records are with- held for testing. As a result, the induced model may not be as good as when all the labeled examples are used for training.
- Second, the model may be highly dependent on the composition of the training and test sets. The smaller the training set size, the larger the variance of the model. On the other hand, if the training set is too large, then the estimated accuracy computed from the smaller test set is less reliable.
- Finally, the training and test sets are no longer independent of each other. Because the training and test sets are subsets of the original data, a class that is overrepresented in one subset will be underrepresented in the other, and vice versa.

2. Random Subsampling

- The holdout method can be repeated several times to improve the estimation of a classifier's performance. This approach is known as random subsampling.
- Let acc_i be the model accuracy during the ith iteration. The overall accuracy is given by $acc_{sub} = \sum_{i=1}^{k} acc_i/k$.
- Random subsampling still encounters some of the problems associated with the holdout method because it does not utilize as much data as possible for training. It also has no control over the number of times each record is used for testing and training. Consequently, some records might be used for training more often than others.

3. Cross-Validation

- An alternative to random subsampling is cross-validation. In this approach, each record is used the same number of times for training and exactly once for testing.
- To illustrate this method, suppose we partition the data into two equal-sized subsets. First, we choose one of the subsets for training and the other for testing. We then swap the roles of the subsets so that the previous training set becomes the test set and vice versa. This approach is called a **two- fold cross-validation**.
- The total error is obtained by summing up the errors for both runs. In this example, each record is used exactly once for training and once for testing.
- The **k-fold cross-validation** method generalizes this approach by segmenting the data into k equal-sized partitions. During each run, one of the partitions is chosen for testing, while the rest of them are used for training.
- This procedure is repeated k times so that each partition is used for testing exactly once. Again, the total error is found by summing up the errors for all k runs.
- A special case of the k-fold cross-validation method sets k = N, the size of the data set. In this so-called **leave-one-out approach,** each test set contains only one record. This approach has the advantage of utilizing as much data as possible for training.
- The drawback of this approach is that it is computationally expensive to repeat the procedure N times. Furthermore, since each test set contains only one record, the variance of the estimated performance metric tends to be high.

4. Bootstrap

- The methods presented so far assume that the training records are sampled without replacement. As a result, there are no duplicate records in the training and test sets.
- In the bootstrap approach, the training records are sampled with replacement; i.e., a record already chosen for training is put back into the original pool of records so that it is equally likely to be redrawn.
- If the original data has N records, it can be shown that, on average, a bootstrap sample of size N contains about 63.2% of the records in the original data.
- Records that are not included in the bootstrap sample become part of the test set.
- The model induced from the training set is then applied to the test set to obtain an estimate of the accuracy of the bootstrap sample, ϵ_i.
- The sampling procedure is then repeated b times to generate b bootstrap samples.
- **.632 bootstrap** - computes the overall accuracy by combining the accuracies of each bootstrap sample (ϵ_i) with the accuracy computed from a training set that contains all the labeled examples in the original data (acc_s):

$$acc_{boot} = \frac{1}{b} \sum_{i=1}^{b} (0.632 \times \epsilon_i + 0.368 \times acc_s)$$

Rule-Based Classifier

- A rule-based classifier is a technique for classifying records using a collection of "if . . .then. . ." rules. The rules for the model are represented in a disjunctive normal form, R = $(r_1 \vee r_2 \vee \ldots r_k)$, where R is known as the rule set and $r_i's$ are the classification rules or disjuncts.

Table 5.1. Example of a rule set for the vertebrate classification problem.

r_1:	$(\text{Gives Birth} = \text{no}) \wedge (\text{Aerial Creature} = \text{yes}) \longrightarrow \text{Birds}$
r_2:	$(\text{Gives Birth} = \text{no}) \wedge (\text{Aquatic Creature} = \text{yes}) \longrightarrow \text{Fishes}$
r_3:	$(\text{Gives Birth} = \text{yes}) \wedge (\text{Body Temperature} = \text{warm-blooded}) \longrightarrow \text{Mammals}$
r_4:	$(\text{Gives Birth} = \text{no}) \wedge (\text{Aerial Creature} = \text{no}) \longrightarrow \text{Reptiles}$
r_5:	$(\text{Aquatic Creature} = \text{semi}) \longrightarrow \text{Amphibians}$

- Each classification rule can be expressed in the following way: $r_i : (condition_i) \rightarrow y_i$
- The left-hand side of the rule is called the **rule antecedent** or **precondition**. It contains a conjunction of attribute tests:

  $condition_i = (A_1\ op\ v_1) \wedge (A_2\ op\ v_2) \wedge \ldots (A_k\ op\ v_k)$ where (A_j, v_j) is an attribute-value pair and op is a logical operator chosen from the set {=,/=, <, >, ≤, ≥}.
- Each attribute test $(A_j\ op\ v_j)$ is known as a **conjunct**. The right-hand side of the rule is called the rule **consequent**, which contains the predicted class y_i.
- A rule r covers a record x if the precondition of r matches the attributes of x. r is also said to be fired or triggered whenever it covers a given record.

Name	Body Temperature	Skin Cover	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hiber-nates
hawk	warm-blooded	feather	no	no	yes	yes	no
grizzly bear	warm-blooded	fur	yes	no	no	yes	yes

- r_1 covers the first vertebrate because its precondition is satisfied by the hawk's attributes. The rule does not cover the second vertebrate because grizzly bears give birth to their young and cannot fly, thus violating the precondition of r_1.
- The quality of a classification rule can be evaluated using measures such as **coverage** and **accuracy**.
- Given a data set D and a classification rule r : A → y, the coverage of the rule is defined as the fraction of records in D that trigger the rule r.
- On the other hand, its **accuracy** or **confidence factor** is defined as the fraction of records triggered by r whose class labels are equal to y.
- $Coverage(r) = \frac{|A|}{|D|}$, $Accuracy(r) = \frac{|A \cap y|}{|A|}$ where |A| is the number of records that satisfy the rule antecedent, |A ∩ y| is the number of records that satisfy both the antecedent and consequent, and |D| is the total number of records.

Table 5.2. The vertebrate data set.

Name	Body Temperature	Skin Cover	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hiber-nates	Class Lab
human	warm-blooded	hair	yes	no	no	yes	no	Mammals
python	cold-blooded	scales	no	no	no	no	yes	Reptiles
salmon	cold-blooded	scales	no	yes	no	no	no	Fishes
whale	warm-blooded	hair	yes	yes	no	no	no	Mammals
frog	cold-blooded	none	no	semi	no	yes	yes	Amphibian
komodo dragon	cold-blooded	scales	no	no	no	yes	no	Reptiles
bat	warm-blooded	hair	yes	no	yes	yes	yes	Mammals
pigeon	warm-blooded	feathers	no	no	yes	yes	no	Birds
cat	warm-blooded	fur	yes	no	no	yes	no	Mammals
guppy	cold-blooded	scales	yes	yes	no	no	no	Fishes
alligator	cold-blooded	scales	no	semi	no	yes	no	Reptiles
penguin	warm-blooded	feathers	no	semi	no	yes	no	Birds
porcupine	warm-blooded	quills	yes	no	no	yes	yes	Mammals
eel	cold-blooded	scales	no	yes	no	no	no	Fishes
salamander	cold-blooded	none	no	semi	no	yes	yes	Amphibian

(Gives Birth = yes) $\wedge$ (Body Temperature = warm-blooded) $\rightarrow$ Mammals

This rule has a coverage of 33% since five of the fifteen records support the rule antecedent. The rule accuracy is 100% because all five vertebrates covered by the rule are mammals.

- A rule-based classifier classifies a test record based on the rule triggered by the record.
- The first vertebrate, which is a lemur, is warm-blooded and gives birth to its young. It triggers the rule r_3, and thus, is classified as a mammal.

Table 5.1. Example of a rule set for the vertebrate classification problem.

r_1: (Gives Birth = no) $\wedge$ (Aerial Creature = yes) $\longrightarrow$ Birds
r_2: (Gives Birth = no) $\wedge$ (Aquatic Creature = yes) $\longrightarrow$ Fishes
r_3: (Gives Birth = yes) $\wedge$ (Body Temperature = warm-blooded) $\longrightarrow$ Mammals
r_4: (Gives Birth = no) $\wedge$ (Aerial Creature = no) $\longrightarrow$ Reptiles
r_5: (Aquatic Creature = semi) $\longrightarrow$ Amphibians

Name	Body Temperature	Skin Cover	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hibernates
lemur	warm-blooded	fur	yes	no	no	yes	yes
turtle	cold-blooded	scales	no	semi	no	yes	no
dogfish shark	cold-blooded	scales	yes	yes	no	no	no

- The second vertebrate, which is a turtle, triggers the rules r_4 and r_5. Since the classes predicted by the rules are contradictory (reptiles versus amphibians), their conflicting classes must be resolved.
- None of the rules are applicable to a dogfish shark. In this case, we need to ensure that the classifier can still make a reliable prediction even though a test record is not covered by any rule.

Two important properties of the rule set gen- erated by a rule-based classifier

1. **Mutually Exclusive Rules** The rules in a rule set R are mutually exclusive if no two rules in R are triggered by the same record. This property ensures that every record is covered by at most one rule in R.
2. **Exhaustive Rules** A rule set R has exhaustive coverage if there is a rule for each combination of attribute values. This property ensures that every record is covered by at least one rule in R.

Table 5.3. Example of a mutually exclusive and exhaustive rule set.

r_1: (Body Temperature = cold-blooded) $\longrightarrow$ Non-mammals
r_2: (Body Temperature = warm-blooded) $\wedge$ (Gives Birth = yes) $\longrightarrow$ Mammals
r_3: (Body Temperature = warm-blooded) $\wedge$ (Gives Birth = no) $\longrightarrow$ Non-mammals

- Together, these properties ensure that every record is covered by exactly one rule.
- If the rule set is not exhaustive, then a default rule, $r_d : () \rightarrow y_d$, must be added to cover the remaining cases. A default rule has an empty antecedent and is triggered when all other rules have failed. y_d is known as the default class and is typically assigned to the majority class of training records not covered by the existing rules.
- If the rule set is not mutually exclusive, then a record can be covered by several rules, some of which may predict conflicting classes. There are two ways to overcome this problem - Ordered rules, Unordered rules.

- **Ordered Rules** In this approach, the rules in a rule set are ordered in decreasing order of their priority, which can be defined in many ways (e.g., based on accuracy, coverage, total description length, or the order in which the rules are generated).
- An ordered rule set is also known as a **decision list**.
- When a test record is presented, it is classified by the highest-ranked rule that covers the record.
- This avoids the problem of having conflicting classes predicted by multiple classification rules.

- **Unordered Rules** This approach allows a test record to trigger multiple classification rules and considers the consequent of each rule as a vote for a particular class.
- The votes are then tallied to determine the class label of the test record. The record is usually assigned to the class that receives the highest number of votes.
- In some cases, the vote may be weighted by the rule's accuracy.
- Unordered rules are less susceptible to errors caused by the wrong rule being selected to classify a test record (unlike classifiers based on ordered rules, which are sensitive to the choice of rule-ordering criteria).
- Model building is also less expensive because the rules do not have to be kept in sorted order.
- Nevertheless, classifying a test record can be quite an expensive task because the attributes of the test record must be compared against the precondition of every rule in the rule set.

Rule-Ordering Schemes

- Rule ordering can be implemented on a rule-by-rule basis or on a class-by-class basis.

Rule-Based Ordering Scheme This approach orders the individual rules by some rule quality measure. This ordering scheme ensures that every test record is classified by the “best” rule covering it.

- A potential drawback of this scheme is that lower-ranked rules are much harder to interpret because they assume the negation of the rules preceding them.
- For example, the fourth rule shown in Figure 5.1 for rule-based ordering, has the following interpretation: If the vertebrate does not have any feathers or cannot fly, and is cold-blooded and semi-aquatic, then it is an amphibian.
- The additional conditions are due to the fact that the vertebrate does not satisfy the first three rules.
- If the number of rules is large, interpreting the meaning of the rules residing near the bottom of the list can be a cumbersome task.

Rule-Based Ordering

(Skin Cover=feathers, Aerial Creature=yes)
==> Birds

(Body temperature=warm-blooded,
Gives Birth=yes) ==> Mammals

(Body temperature=warm-blooded,
Gives Birth=no) ==> Birds

(Aquatic Creature=semi)) ==> Amphibians

(Skin Cover=scales, Aquatic Creature=no)
==> Reptiles

(Skin Cover=scales, Aquatic Creature=yes)
==> Fishes

(Skin Cover=none) ==> Amphibians

Class-Based Ordering

(Skin Cover=feathers, Aerial Creature=yes)
==> Birds

(Body temperature=warm-blooded,
Gives Birth=no) ==> Birds

(Body temperature=warm-blooded,
Gives Birth=yes) ==> Mammals

(Aquatic Creature=semi)) ==> Amphibians

(Skin Cover=none) ==> Amphibians

(Skin Cover=scales, Aquatic Creature=no)
==> Reptiles

(Skin Cover=scales, Aquatic Creature=yes)
==> Fishes

Figure 5.1. Comparison between rule-based and class-based ordering schemes.

Class-Based Ordering Scheme In this approach, rules that belong to the same class appear together in the rule set R. The rules are then collectively sorted on the basis of their class information.

The relative ordering among the rules from the same class is not important; as long as one of the rules fires, the class will be assigned to the test record. This makes rule interpretation slightly easier.

However, it is possible for a high-quality rule to be overlooked in favor of an inferior rule that happens to predict the higher-ranked class.

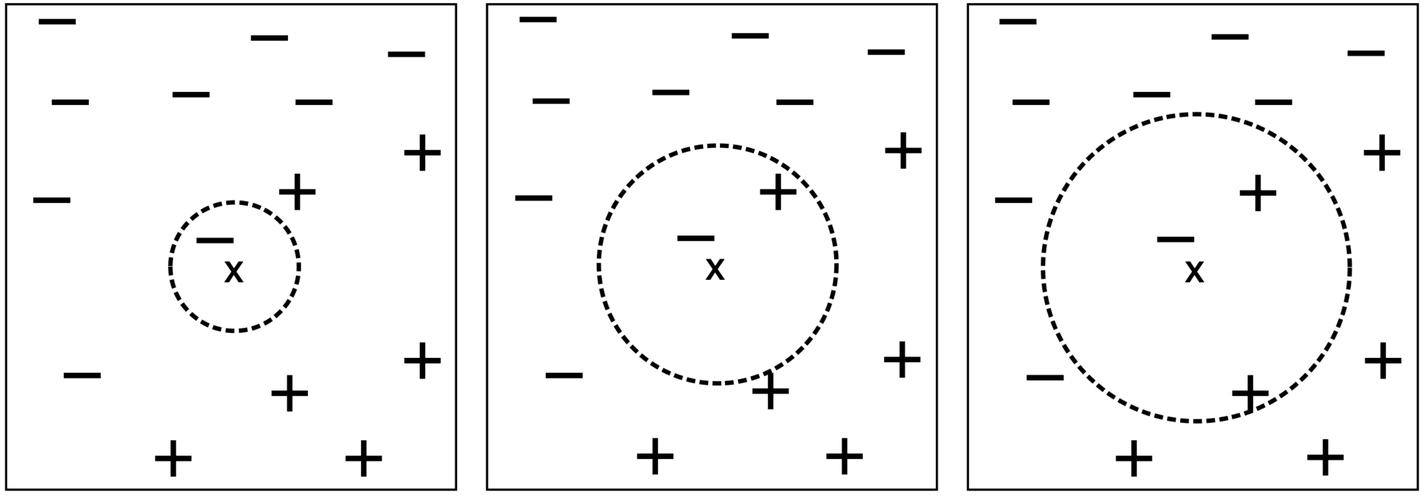
How to Build a Rule-Based Classifier

- To build a rule-based classifier, we need to extract a set of rules that identifies key relationships between the attributes of a data set and the class label.
- There are two broad classes of methods for extracting classification rules: (1) direct methods, which extract classification rules directly from data, and (2) indirect methods, which extract classification rules from other classification models, such as decision trees and neural networks.
- Direct methods partition the attribute space into smaller subspaces so that all the records that belong to a subspace can be classified using a single classification rule.
- Indirect methods use the classification rules to provide a succinct description of more complex classification models.

- Decision tree and rule-based classifiers are examples of **eager learners** because they are designed to learn a model that maps the input attributes to the class label as soon as the training data becomes available.
- An opposite strategy would be to delay the process of modeling the training data until it is needed to classify the test examples.
- Techniques that employ this strategy are known as **lazy learners**.
- An example of a lazy learner is the **Rote classifier**, which memorizes the entire training data and performs classification only if the attributes of a test instance match one of the training examples exactly.
- An obvious drawback of this approach is that some test records may not be classified because they do not match any training example.
- **Nearest Neighbors -** find all the training examples that are relatively similar to the attributes of the test example.

Nearest-Neighbor classifiers

- A nearest- neighbor classifier represents each example as a data point in a d-dimensional space, where d is the number of attributes.
- Given a test example, we compute its proximity to the rest of the data points in the training set, using one of the proximity measures. The k-nearest neighbors of a given example z refer to the k points that are closest to z.



(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

Figure 5.7. The 1-, 2-, and 3-nearest neighbors of an instance.

- The data point is classified based on the class labels of its neighbors.
- In the case where the neighbors have more than one label, the data point is assigned to the majority class of its nearest neighbors.
- In Figure 5.7(a), the 1-nearest neighbor of the data point is a negative example. Therefore the data point is assigned to the negative class.
- If the number of nearest neighbors is three, as shown in Figure 5.7(c), then the neighborhood contains two positive examples and one negative example. Using the majority voting scheme, the data point is assigned to the positive class.
- In the case where there is a tie between the classes (see Figure 5.7(b)), we may randomly choose one of them to classify the data point.

- If k is too small, then the nearest-neighbor classifier may be susceptible to overfitting because of noise in the training data.
- On the other hand, if k is too large, the nearest-neighbor classifier may misclassify the test instance because its list of nearest neighbors may include data points that are located far away from its neighborhood.

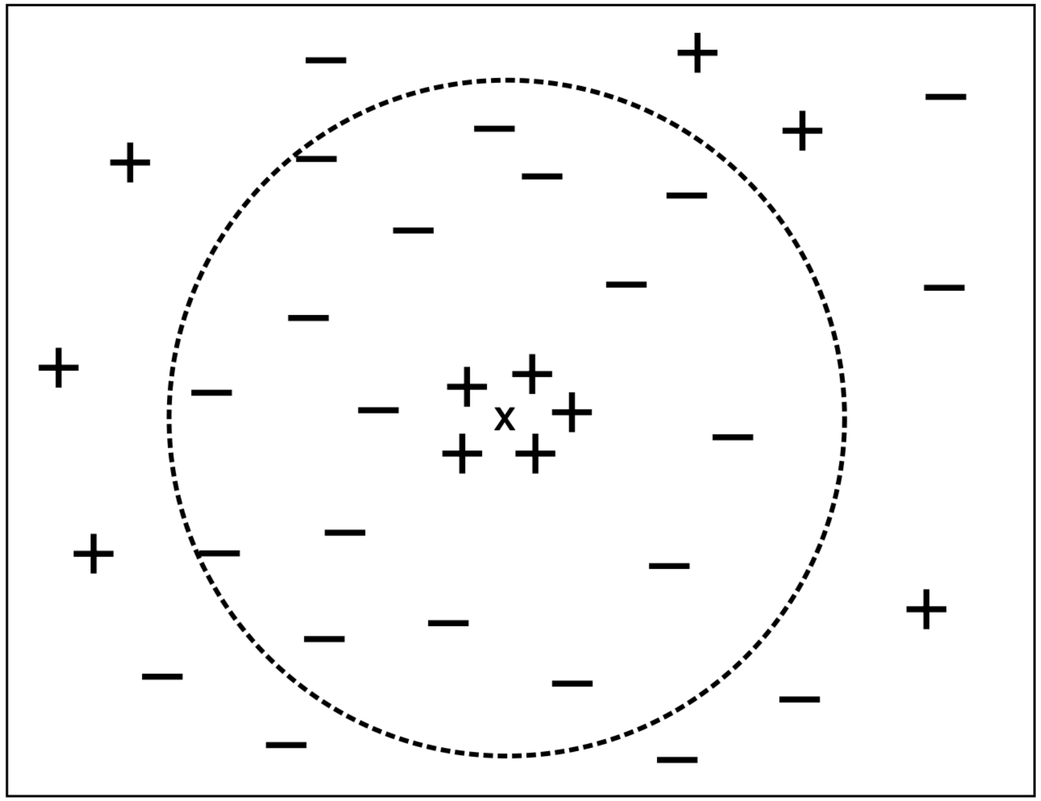


Figure 5.8. k-nearest neighbor classification with large k.

Algorithm 5.2 The k-nearest neighbor classification algorithm.

1: Let k be the number of nearest neighbors and D be the set of training examples.
2: **for** each test example $z = (\mathbf{x}', y')$ **do**
3: Compute $d(\mathbf{x}', \mathbf{x})$, the distance between z and every example, $(\mathbf{x}, y) \in D$.
4: Select $D_z \subseteq D$, the set of k closest training examples to z.
5: $y' = \underset{v}{\operatorname{argmax}} \sum_{(\mathbf{x}_i, y_i) \in D_z} I(v = y_i)$
6: **end for**

- The algorithm computes the distance (or similarity) between each test example $z = (x', y')$ and all the training examples (x,y) ∈ D to determine its nearest-neighbor list, D_z.
- Such computation can be costly if the number of training examples is large.
- Once the nearest-neighbor list is obtained, the test example is classified based on the majority class of its nearest neighbors - where v is a class label, y_i is the class label for one of the nearest neighbors, and I(·) is an indicator function that returns the value 1 if its argument is true and 0 otherwise.

- In the majority voting approach, every neighbor has the same impact on the classification. This makes the algorithm sensitive to the choice of k.
- One way to reduce the impact of k is to weight the influence of each nearest neighbor x_i according to its distance: $w_i = 1/d(x', x_i)^2$.
- As a result, training examples that are located far away from z have a weaker impact on the classification compared to those that are located close to z.
- Using the distance-weighted voting scheme, the class label can be determined as follows:

$$Distance - weighted\ Voting : y' = argmax_v \sum_{(x_i, y_i) \epsilon D_z} w_i \times I(v = y_i)$$

Example

BRIGHTNESS	SATURATION	CLASS
40	20	Red
50	50	Blue
60	90	Blue
10	25	Red
70	70	Blue
60	10	Red
25	80	Blue

BRIGHTNESS	SATURATION	CLASS
20	35	?

- Using Euclidean distance as the proximity measure, compute the distance of the new point with all the training data points.

$$d1 = \sqrt{(20-40)^2 + (35-20)^2} = \sqrt{400+225} = 25$$

$$d2 = \sqrt{(20-50)^2 + (35-50)^2} = \sqrt{900+225} = 33.54$$

$$d3 = \sqrt{(20-60)^2 + (35-90)^2} = \sqrt{1600+3025} = 68.01$$

$$d4 = \sqrt{(20-10)^2 + (35-25)^2} = \sqrt{100+100} = 14.14$$

$$d5 = \sqrt{(20-70)^2 + (35-70)^2} = \sqrt{2500+1225} = 61.03$$

$$d6 = \sqrt{(20-60)^2 + (35-10)^2} = \sqrt{1600+625} = 47.17$$

$$d7 = \sqrt{(20-25)^2 + (35-80)^2} = \sqrt{25+2025} = 45.28$$

BRIGHTNESS	SATURATION	CLASS
10	25	Red
40	20	Red
50	50	Blue

- For $k = 3$, choose the three closest points - d4, d1 and d2.
- The majority class within the 3 nearest neighbors to the new entry is **Red**. Therefore, we'll classify the new entry as **Red**.

Characteristics of Nearest-Neighbor Classifiers

- Nearest-neighbor classification is part of a more general technique known as **instance-based learning**, which uses specific training instances to make predictions without having to maintain an abstraction (or model) derived from data. Instance-based learning algorithms require a proximity measure to determine the similarity or distance between instances and a classification function that returns the predicted class of a test instance based on its proximity to other instances.
- Lazy learners such as nearest-neighbor classifiers do not require model building. However, classifying a test example can be quite expensive because we need to compute the proximity values individually between the test and training examples. In contrast, eager learners often spend the bulk of their computing resources for model building. Once a model has been built, classifying a test example is extremely fast.

Characteristics of Nearest-Neighbor Classifiers

- Nearest-neighbor classifiers make their predictions based on local information, whereas decision tree and rule-based classifiers attempt to find a global model that fits the entire input space. Because the classification decisions are made locally, nearest-neighbor classifiers (with small values of k) are quite susceptible to noise.
- Nearest-neighbor classifiers can produce arbitrarily shaped decision boundaries. Such boundaries provide a more flexible model representation compared to decision tree and rule-based classifiers that are often constrained to rectilinear decision boundaries.
- Nearest-neighbor classifiers can produce wrong predictions unless the appropriate proximity measure and data preprocessing steps are taken. For example, suppose we want to classify a group of people based on attributes such as height (measured in meters) and weight (measured in pounds). The height attribute has a low variability, ranging from 1.5 m to 1.85 m, whereas the weight attribute may vary from 90 lb. to 250 lb. If the scale of the attributes are not taken into consideration, the proximity measure may be dominated by differences in the weights of a person.

Bayes Theorem

- Let X and Y be a pair of random variables.
- Their **joint probability, P (X = x,Y = y)**, refers to the probability that variable X will take on the value x and variable Y will take on the value y.
- A **conditional probability** is the probability that a random variable will take on a particular value given that the outcome for another random variable is known. For example, the conditional probability P (Y = y|X = x) refers to the probability that the variable Y will take on the value y, given that the variable X is observed to have the value x.
- The joint and conditional probabilities for X and Y are related in the following way:
- P (X, Y) = P (Y |X) × P (X) = P (X|Y) × P (Y).
- $P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$

Consider a football game between two rival teams: Team 0 and Team 1. Suppose Team 0 wins 65% of the time and Team 1 wins the remaining matches. Among the games won by Team 0, only 30% of them come from playing on Team 1's football field. On the other hand, 75% of the victories for Team 1 are obtained while playing at home. If Team 1 is to host the next match between the two teams, which team will most likely emerge as the winner?

- Let X be the random variable that represents the team hosting the match and Y be the random variable that represents the winner of the match. Both X and Y can take on values from the set {0, 1}.
- We can summarize the information given in the problem as follows:
- Probability Team 0 wins is $P(Y = 0) = 0.65$.
  Probability Team 1 wins is $P(Y = 1) = 1 - P(Y = 0) = 0.35$.
  Probability Team 1 hosted the match it won is $P(X = 1|Y = 1) = 0.75$.
- Probability Team 1 hosted the match won by Team 0 is $P(X = 1|Y = 0) = 0.3$.

- We compute $P(Y = 1|X = 1)$, which is the conditional probability that Team 1 wins the next match it will be hosting, and compare it against $P(Y = 0|X = 1)$. Using the Bayes theorem, we obtain

$$P(Y=1|X=1) = P(X=1|Y=1)\times P(Y=1) / P(X = 1)$$

$$= P(X=1|Y=1)\times P(Y=1) / (P(X=1,Y=1)+P(X=1,Y=0))$$

$$= P(X=1|Y=1)\times P(Y=1) / (P(X=1|Y=1)P(Y=1)+P(X=1|Y=0)P(Y=0))$$

$$= 0.75 \times 0.35 / (0.75 \times 0.35 + 0.3 \times 0.65) = 0.5738,$$

- Furthermore, $P(Y=0|X=1)=1-P(Y=1|X=1)= 0.4262$.
- Since $P(Y=1|X=1)>P(Y=0|X=1)$, Team 1 has a better chance than Team 0 of winning the next match.

Bayes Theorem for Classification

- Let X denote the attribute set and Y denote the class variable.
- If the class variable has a non-deterministic (same input different outcomes) relationship with the attributes, then we can treat X and Y as random variables and capture their relationship probabilistically using P (Y |X).
- This conditional probability is also known as the **posterior probability** for Y , as opposed to its **prior probability**, P (Y).
- During the training phase, we need to learn the posterior probabilities P(Y|X) for every combination of X and Y based on information gathered from the training data.
- By knowing these probabilities, a test record X' can be classified by finding the class Y' that maximizes the posterior probability, P(Y'|X').

	binary	categorical	continuous	class
Tid	**Home Owner**	**Marital Status**	**Annual Income**	**Defaulted Borrower**
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Figure 5.9. Training set for predicting the loan default problem.

- Suppose we are given a test record with the following attribute set: X = (Home Owner = No, Marital Status = Married, Annual Income = $120K).
- To classify the record, we need to compute the posterior probabilities P(Yes|X) and P (No|X) based on information available in the training data.
- If P (Yes|X) > P (No|X), then the record is classified as Yes; otherwise, it is classified as No.
- When comparing the posterior probabilities for different values of Y , the denominator term, P(X), is always constant, and thus, can be ignored.
- The prior probability P (Y) can be easily estimated from the training set by computing the fraction of training records that belong to each class.
- P(Y=yes) = 3/10 = 0.3 and P(Y=no) = 0.7.

Naive Bayes Classifier

- A naive Bayes classifier estimates the class-conditional probability by assuming that the attributes are conditionally independent, given the class label y.

$$P(X \mid Y = y) = \prod_{i-1}^{d} P(X_i \mid Y = y)$$

- where each attribute set $X = \{X_1, X_2, \ldots X_d\}$ consists of d attributes.
- To classify a test record, the naïve Bayes classifier computes the posterior probability for each class Y :
- $$P(Y \mid X) = \frac{P(Y) \prod_{i=1}^{d} P(X_i \mid Y)}{P(X)}$$
- Since P (X) is fixed for every Y , it is sufficient to choose the class that maximizes the numerator term.

Estimating Conditional Probabilities for Categorical Attributes

- For a categorical attribute X_i, the conditional probability $P(X_i = x_i \mid Y = y)$ is estimated according to the fraction of training instances in class y that take on a particular attribute value x_i.
- For example, in the training set given in Figure 5.9, three out of the seven people who repaid their loans also own a home. As a result, the conditional probability for P(Home Owner=Yes|No) is equal to 3/7.
- Similarly, the conditional probability for defaulted borrowers who are single is given by P (Marital Status = Single|Yes) = 2/3.

Estimating Conditional Probabilities for Continuous Attributes

There are two ways to estimate the class-conditional probabilities for continuous attributes in naive Bayes classifiers:

- We can discretize each continuous attribute and then replace the continuous attribute value with its corresponding discrete interval. This approach transforms the continuous attributes into ordinal attributes. The conditional probability $P(X_i \mid Y = y)$ is estimated by computing the fraction of training records belonging to class y that falls within the corresponding interval for X_i. If the number of intervals is too large, there are too few training records in each interval to provide a reliable estimate for $P(X_i|Y)$. On the other hand, if the number of intervals is too small, then some intervals may aggregate records from different classes and we may miss the correct decision boundary.

.

- We can assume a certain form of probability distribution for the continuous variable and estimate the parameters of the distribution using the training data. A Gaussian distribution is usually chosen to represent the class-conditional probability for continuous attributes. The distribution is characterized by two parameters, its mean μ and variance σ^2. For each class y_j, the class-conditional probability for attribute X_i is
- $$P(X_i = x_i \mid Y = y_j) = \frac{1}{\sqrt{2\pi}\sigma_{ij}} exp^{-\frac{(x_i - \mu_{ij})^2}{2\sigma_{ij}^2}}$$
- The parameter μ_{ij} can be estimated based on the sample mean of X_i ($\bar{x}$) for all training records that belong to the class y_j.
- Similarly, σ_{ij}^2 can be estimated from the sample variance (s^2) of such training records.

$$\overline{x} = \frac{125 + 100 + 70 + \ldots + 75}{7} = 110$$

$$s^2 = \frac{(125 - 110)^2 + (100 - 110)^2 + \ldots + (75 - 110)^2}{7(6)} = 2975$$

$$s = \sqrt{2975} = 54.54.$$

- For example, consider the annual income attribute shown in Figure 5.9. The sample mean and variance for this attribute with respect to the class No are given above.
- Given a test record with taxable income equal to $120K, we can compute its class-conditional probability as follows:
- $$P(Income = 120 \mid No) = \frac{1}{\sqrt{2\pi}(54.54)} exp^{-\frac{(120 - 110)^2}{2 \times 2975}} = 0.0072$$

Example

Tid	Home Owner	Marital Status	Annual Income	Defaulted Borrower
1	Yes	Single	125K	**No**
2	No	Married	100K	**No**
3	No	Single	70K	**No**
4	Yes	Married	120K	**No**
5	No	Divorced	95K	**Yes**
6	No	Married	60K	**No**
7	Yes	Divorced	220K	**No**
8	No	Single	85K	**Yes**
9	No	Married	75K	**No**
10	No	Single	90K	**Yes**

(a)

P(Home Owner=Yes|No) = 3/7
P(Home Owner=No|No) = 4/7
P(Home Owner=Yes|Yes) = 0
P(Home Owner=No|Yes) = 1
P(Marital Status=Single|No) = 2/7
P(Marital Status=Divorced|No) = 1/7
P(Marital Status=Married|No) = 4/7
P(Marital Status=Single|Yes) = 2/3
P(Marital Status=Divorced|Yes) = 1/3
P(Marital Status=Married|Yes) = 0

For Annual Income:
If class=No: sample mean=110
sample variance=2975
If class=Yes: sample mean=90
sample variance=25

(b)

Figure 5.10. The naïve Bayes classifier for the loan classification problem.

- To predict the class label of a test record X = (Home Owner=No, Marital Status = Married, Income = $120K), we need to compute the posterior probabilities P(No|X) and P(Yes|X).

$$
\begin{aligned}
P(\mathbf{X}|\texttt{No}) &= P(\texttt{Home Owner} = \texttt{No}|\texttt{No}) \times P(\texttt{Status} = \texttt{Married}|\texttt{No}) \\
&\quad \times P(\texttt{Annual Income} = \$120\text{K}|\texttt{No}) \\
&= 4/7 \times 4/7 \times 0.0072 = 0.0024.
\end{aligned}
$$

$$
\begin{aligned}
P(\mathbf{X}|\texttt{Yes}) &= P(\texttt{Home Owner} = \texttt{No}|\texttt{Yes}) \times P(\texttt{Status} = \texttt{Married}|\texttt{Yes}) \\
&\quad \times P(\texttt{Annual Income} = \$120\text{K}|\texttt{Yes}) \\
&= 1 \times 0 \times 1.2 \times 10^{-9} = 0.
\end{aligned}
$$

- The prior probabilities of each class can be estimated by calculating the fraction of training records that belong to each class. Since there are three records that belong to the class Yes and seven records that belong to the class No, P (Yes) = 0.3 and P (No) = 0.7.
- The posterior probability for class No is P (No|X) = $\alpha \times 7/10 \times 0.0024 = 0.0016\alpha$, where $\alpha = 1/P(X)$ is a constant term.
- Using a similar approach, we can show that the posterior probability for class Yes is zero because its class-conditional probability is zero.
- Since P(No|X) > P(Yes|X), the record is classified as No.

Alternative Metrics

- Since the accuracy measure treats every class as equally important, it may not be suitable for analyzing imbalanced data sets, where the rare class is considered more interesting than the majority class.
- For binary classification, the rare class is often denoted as the positive class, while the majority class is denoted as the negative class.

Table 5.6. A confusion matrix for a binary classification problem in which the classes are not equally important.

		Predicted Class	
		$+$	$-$
Actual Class	$+$	f_{++} (TP)	f_{+-} (FN)
	$-$	f_{-+} (FP)	f_{--} (TN)

- The following terminology is often used when referring to the counts tabulated in a confusion matrix:
- True positive (TP) or f++, which corresponds to the number of positive examples correctly predicted by the classification model.
- False negative (FN) or f+-, which corresponds to the number of positive examples wrongly predicted as negative by the classification model.
- False positive (FP) or f-+, which corresponds to the number of negative examples wrongly predicted as positive by the classification model.
- True negative (TN) or f—, which corresponds to the number of negative examples correctly predicted by the classification model.

- The true positive rate (TPR) or sensitivity is defined as the fraction of positive examples predicted correctly by the model, i.e.,
  TPR = TP/(TP + FN).
- Similarly, the true negative rate (TNR) or specificity is defined as the fraction of negative examples predicted correctly by the model, i.e., TNR = TN/(TN + FP).
- Finally, the false positive rate (FPR) is the fraction of negative examples predicted as a positive class, i.e., FPR = FP/(TN + FP), while the false negative rate (FNR) is the fraction of positive examples predicted as a negative class, i.e., FNR = FN/(TP + FN).
- **Recall** and **precision** are two widely used metrics employed in applications where successful detection of one of the classes is considered more significant than detection of the other classes.
- Precision, p= TP/TP+FP
- Recall, r= TP/TP+FN

- Precision determines the fraction of records that actually turns out to be positive in the group the classifier has declared as a positive class.
- The higher the precision is, the lower the number of false positive errors committed by the classifier.
- Recall measures the fraction of positive examples correctly predicted by the classifier.
- Classifiers with large recall have very few positive examples misclassified as the negative class. In fact, the value of recall is equivalent to the true positive rate.
- It is often possible to construct baseline models that maximize one metric but not the other.
- For example, a model that declares every record to be the positive class will have a perfect recall, but very poor precision.
- Conversely, a model that assigns a positive class to every test record that matches one of the positive records in the training set has very high precision, but low recall.

- Precision and recall can be summarized into another metric known as the F1 measure.
- $$F_1 = \frac{2rp}{r+p} = \frac{2TP}{2TP + FP + FN}$$
- F1 represents a harmonic mean between recall and precision.
- The harmonic mean of two numbers x and y tends to be closer to the smaller of the two numbers. Hence, a high value of F_1-measure ensures that both precision and recall are reasonably high.
- Consider two positive numbers a = 1 and b = 5. Their arithmetic mean is $\mu = (a+b)/2 = 3$ and their geometric mean is $\mu_g = \sqrt{ab} = 2.236$. Their harmonic mean is $\mu_h = (2\times1\times5)/6 = 1.667$, which is closer to the smaller value between a and b than the arithmetic and geometric means.

$$F_\beta = \frac{(\beta^2+1)rp}{r+\beta^2 p} = \frac{(\beta^2+1)\times TP}{(\beta^2+1)TP+\beta^2 FP+FN}.$$

- The F_β measure can be used to examine the tradeoff between recall and precision.
- Both precision and recall are special cases of F_β by setting $\beta = 0$ and $\beta = \infty$, respectively.
- Low values of β make F_β closer to precision, and high values make it closer to recall.
- A more general metric that captures F_β as well as accuracy is the weighted accuracy measure, which is defined by the following equation:

$$\text{Weighted accuracy} = \frac{w_1 TP + w_4 TN}{w_1 TP + w_2 FP + w_3 FN + w_4 TN}.$$

- The relationship between weighted accuracy and other performance metrics is summarized in the following table:

.

Measure	w_1	w_2	w_3	w_4
Recall	1	1	0	0
Precision	1	0	1	0
F_β	$\beta^2 + 1$	β^2	1	0
Accuracy	1	1	1	1

Association Analysis

- Useful for discovering interesting relationships hidden in large data sets.
- The uncovered relationships can be represented in the form of association rules or sets of frequent items.

{Diapers} −→ {Beer}

- The rule suggests that a strong relationship exists between the sale of diapers and beer because many customers who buy diapers also buy beer.

Table 6.1. An example of market basket transactions.

TID	Items
1	{Bread, Milk}
2	{Bread, Diapers, Beer, Eggs}
3	{Milk, Diapers, Beer, Cola}
4	{Bread, Milk, Diapers, Beer}
5	{Bread, Milk, Diapers, Cola}

Table 6.2. A binary 0/1 representation of market basket data.

TID	Bread	Milk	Diapers	Beer	Eggs	Cola
1	1	1	0	0	0	0
2	1	0	1	1	1	0
3	0	1	1	1	0	1
4	1	1	1	1	0	0
5	1	1	1	0	0	1

- **Binary Representation** Market basket data can be represented in a binary format, where each row corresponds to a transaction and each column corresponds to an item.
- An item can be treated as a binary variable whose value is one if the item is present in a transaction and zero otherwise.
- Because the presence of an item in a transaction is often considered more important than its absence, an item is an asymmetric binary variable.

- **Itemset and Support Count** Let $I = \{i_1, i_2, \ldots i_d\}$ be the set of all items in a market basket data and $T = \{t_1, t_2, \ldots t_N\}$ be the set of all transactions. Each transaction t_i contains a subset of items chosen from I.
- In association analysis, a collection of zero or more items is termed an **itemset**. If an itemset contains k items, it is called a **k-itemset**. For instance, {Beer, Diapers, Milk} is an example of a 3-itemset.
- The null (or empty) set is an itemset that does not contain any items.
- The **transaction width** is defined as the number of items present in a transaction.
- A transaction t_i is said to contain an itemset X if X is a subset of t_i. For example, the second transaction shown in Table 6.2 contains the itemset {Bread, Diapers} but not {Bread, Milk}.
- An important property of an itemset is its **support count**, which refers to the number of transactions that contain a particular itemset. Mathematically, the support count, σ(X), for an itemset X can be stated as follows:
- $\sigma(X) = |\{t_i | X \subseteq t_i, t_i \in T\}|$
- In the data set shown in Table 6.2, the support count for {Beer, Diapers, Milk} is equal to two because there are only two transactions that contain all three items.

- **Association Rule** An association rule is an implication expression of the form X → Y, where X and Y are disjoint itemsets, i.e., X ∩Y = ∅.
- The strength of an association rule can be measured in terms of its **support** and **confidence**.
- **Support** determines how often a rule is applicable to a given data set, while **confidence** determines how frequently items in Y appear in transactions that contain X.
- $Support, s(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{N}$
- $Confidence, c(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X)}$
- Example 6.1. Consider the rule {Milk, Diapers} −→ {Beer}. Since the support count for {Milk, Diapers, Beer} is 2 and the total number of transactions is 5, the rule's support is 2/5 = 0.4. The rule's confidence is obtained by dividing the support count for {Milk, Diapers, Beer} by the support count for {Milk, Diapers}. Since there are 3 transactions that contain milk and diapers, the confidence for this rule is 2/3 = 0.67.

- **Why Use Support and Confidence?**
- Support is an important measure because a rule that has very low support may occur simply by chance. A low support rule is also likely to be uninteresting from a business perspective because it may not be profitable to promote items that customers seldom buy together. For these reasons, support is often used to eliminate uninteresting rules.
- Confidence, on the other hand, measures the reliability of the inference made by a rule. For a given rule $X \rightarrow Y$, the higher the confidence, the more likely it is for Y to be present in transactions that contain X. Confidence also provides an estimate of the conditional probability of Y given X.
- **Association Rule Discovery** Given a set of transactions T, find all the rules having support ≥ minsup and confidence ≥ minconf, where minsup and minconf are the corresponding support and confidence thresholds.

- A brute-force approach for mining association rules is to compute the support and confidence for every possible rule.
- This approach is prohibitively expensive because there are exponentially many rules that can be extracted from a data set.
- More specifically, the total number of possible rules extracted from a data set that contains d items is $R = 3^d - 2^{d+1} + 1$.
- Even for the small data set shown in Table 6.1, this approach requires us to compute the support and confidence for 602 rules.
- More than 80% of the rules are discarded after applying minsup = 20% and minconf = 50%, thus making most of the computations become wasted.

- The support of a rule X → Y depends only on the support of its corresponding itemset, X ∪ Y .
- For example, the following rules have identical support because they involve items from the same itemset, {Beer, Diapers, Milk}:
- {Beer, Diapers} −→ {Milk}, {Diapers, Milk} −→ {Beer}, {Milk} −→ {Beer,Diapers},
- {Beer, Milk} −→ {Diapers}, {Beer} −→ {Diapers, Milk}, {Diapers} −→ {Beer,Milk}.
- If the itemset is infrequent, then all six candidate rules can be pruned immediately without our having to compute their confidence values.

Decompose the problem into two major subtasks:

1. **Frequent Itemset Generation**, whose objective is to find all the itemsets that satisfy the minsup threshold. These itemsets are called frequent itemsets.
2. **Rule Generation**, whose objective is to extract all the high-confidence rules from the frequent itemsets found in the previous step. These rules are called strong rules.

Frequent Itemset Generation

- A lattice structure can be used to enumerate the list of all possible itemsets. Figure 6.1 shows an itemset lattice for I = {a, b, c, d, e}. In general, a data set that contains k items can potentially generate up to $2^k - 1$ frequent itemsets, excluding the null set.

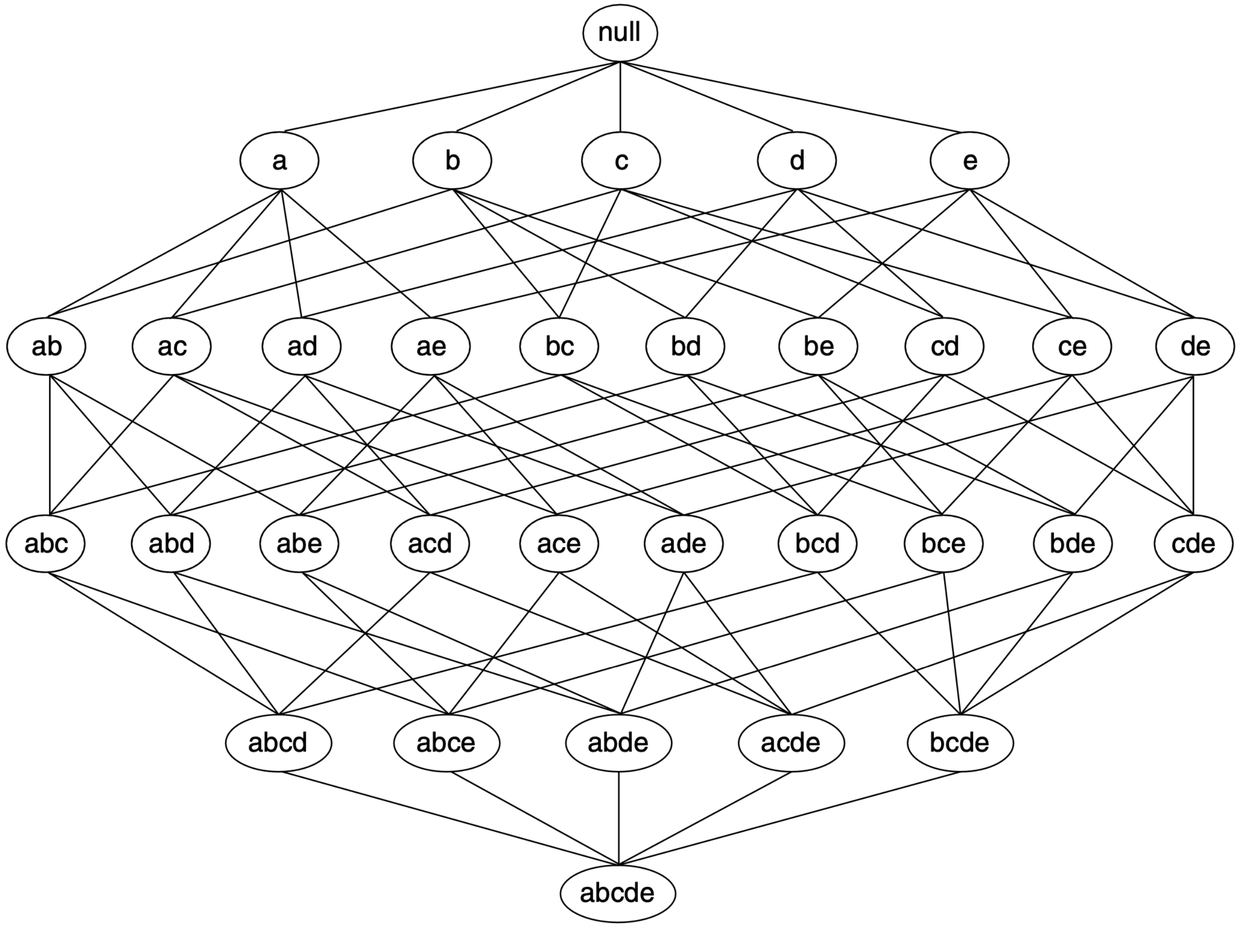


Figure 6.1. An itemset lattice.

- A brute-force approach for finding frequent itemsets is to determine the support count for every candidate itemset in the lattice structure. To do this, we need to compare each candidate against every transaction.
- If the candidate is contained in a transaction, its support count will be incremented. For example, the support for {Bread, Milk} is incremented three times because the itemset is contained in transactions 1, 4, and 5.
- Such an approach can be very expensive because it requires O(NMw) comparisons, where N is the number of transactions, $M = 2^k - 1$ is the number of candidate itemsets, and w is the maximum transaction width.

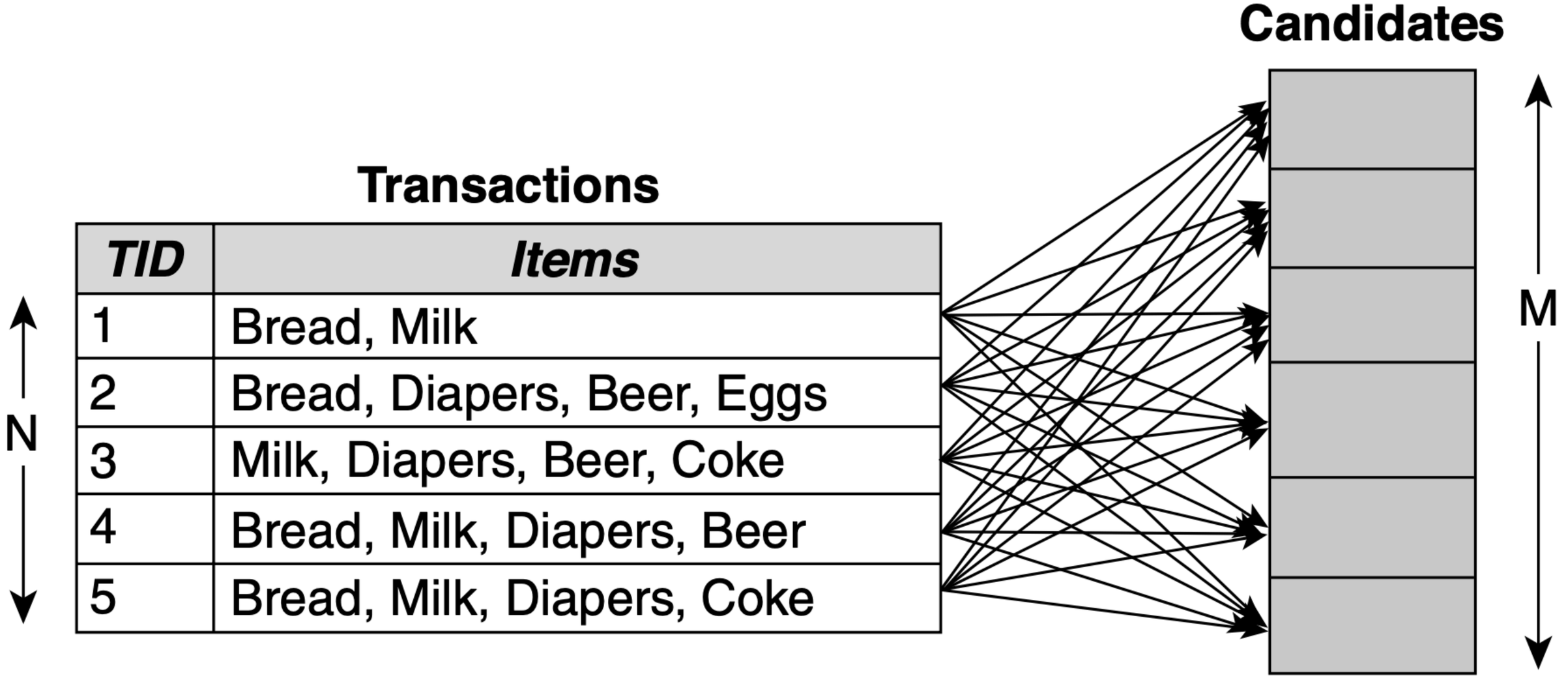


Figure 6.2. Counting the support of candidate itemsets.

- There are several ways to reduce the computational complexity of frequent itemset generation.

1. **Reduce the number of candidate itemsets (M)**. The Apriori principle is an effective way to eliminate some of the candidate itemsets without counting their support values.
2. **Reduce the number of comparisons.** Instead of matching each candidate itemset against every transaction, we can reduce the number of comparisons by using more advanced data structures, either to store the candidate itemsets or to compress the data set.

The Apriori Principle

If an itemset is frequent, then all of its subsets must also be frequent.

- Suppose {c,d,e} is a frequent itemset. Clearly, any transaction that contains {c,d,e} must also contain its subsets, {c,d}, {c, e}, {d, e}, {c}, {d}, and {e}.
- As a result, if {c, d, e} is frequent, then all subsets of {c,d,e} must also be frequent.
- Conversely, if an itemset such as {a, b} is infrequent, then all of its supersets must be infrequent too. Hence, all the supersets of {a,b} can be pruned immediately once {a,b} is found to be infrequent.
- This strategy of trimming the exponential search space based on the support measure is known as **support-based pruning**.
- Such a pruning strategy is made possible by a key property of the support measure, namely, that the support for an itemset never exceeds the support for its subsets. This property is also known as the **anti-monotone property** of the support measure.

- **Monotonicity Property** Let I be a set of items, and $J = 2^I$ be the power set of I. A measure f is monotone (or upward closed) if

$$\forall X, Y \in J : (X \subseteq Y) \rightarrow f(X) \leq f(Y)$$

which means that if X is a subset of Y , then f(X) must not exceed f(Y).

On the other hand, f is anti-monotone (or downward closed) if

$$\forall X, Y \in J : (X \subseteq Y) \rightarrow f(Y) \leq f(X)$$

which means that if X is a subset of Y , then f(Y) must not exceed f(X).

Frequent Itemset Generation in the Apriori Algorithm

Table 6.1. An example of market basket transactions.

TID	Items
1	{Bread, Milk}
2	{Bread, Diapers, Beer, Eggs}
3	{Milk, Diapers, Beer, Cola}
4	{Bread, Milk, Diapers, Beer}
5	{Bread, Milk, Diapers, Cola}

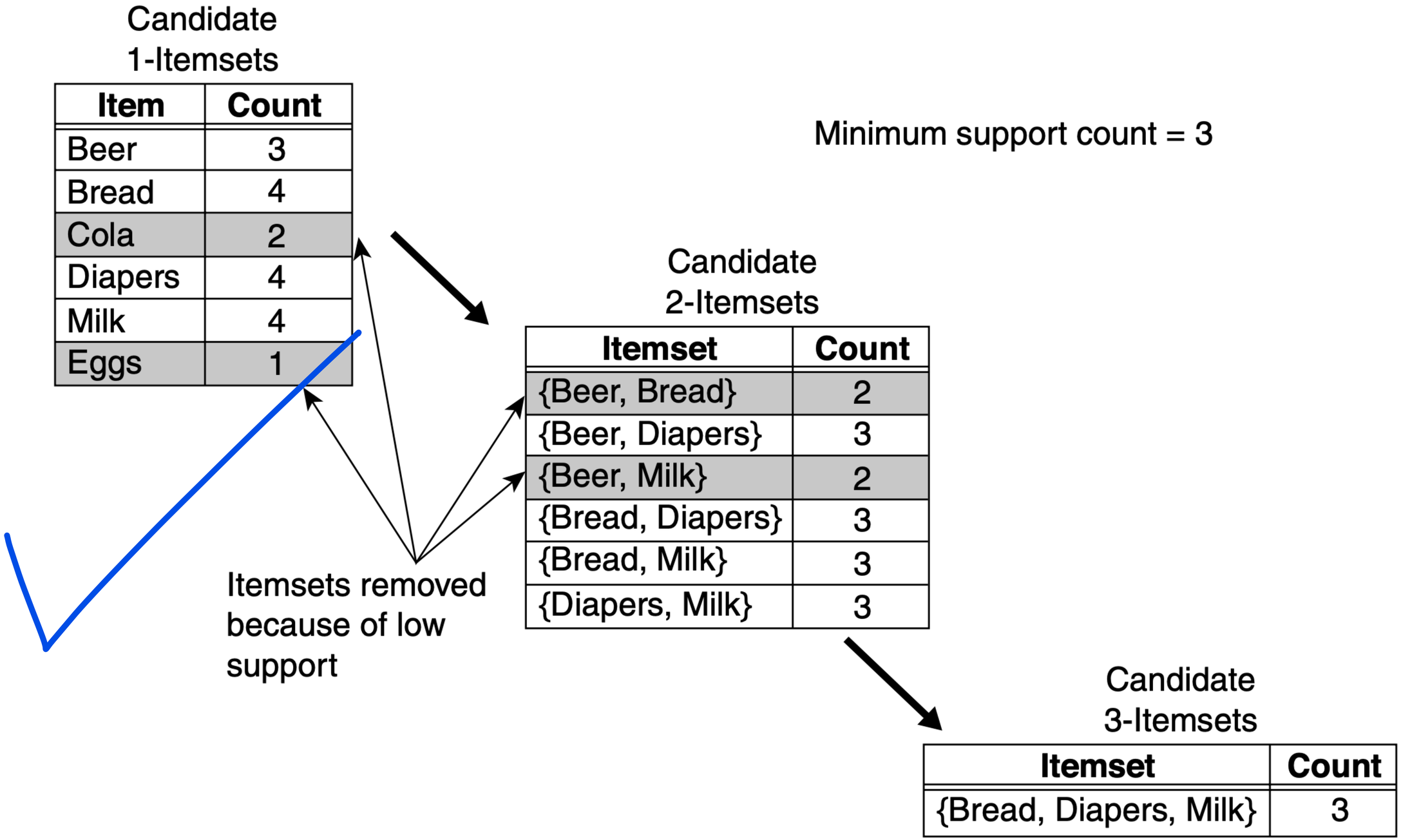


Figure 6.5. Illustration of frequent itemset generation using the *Apriori* algorithm.

- We assume that the support threshold is 60%, which is equivalent to a minimum support count equal to 3. (60% of 5 is 3)
- Initially, every item is considered as a candidate 1-itemset. After counting their supports, the candidate itemsets {Cola} and {Eggs} are discarded because they appear in fewer than three transactions.
- In the next iteration, candidate 2-itemsets are generated using only the frequent 1-itemsets because the Apriori principle ensures that all supersets of the infrequent 1-itemsets must be infrequent.
- Because there are only four frequent 1-itemsets, the number of candidate 2-itemsets generated by the algorithm is ${}^4C_2 = 6$.
- Two of these six candidates, {Beer, Bread} and {Beer, Milk}, are subsequently found to be infrequent after computing their support values. The remaining four candidates are frequent, and thus will be used to generate candidate 3-itemsets.
- Without support-based pruning, there are ${}^6C3 = 20$ candidate 3-itemsets that can be formed using the six items given in this example. With the Apriori principle, we only need to keep candidate 3-itemsets whose subsets are frequent. The only candidate that has this property is {Bread, Diapers, Milk}.
- For {Beer, Diapers, Bread}, the subset {Beer, Bread} is infrequent. Hence it is excluded.

- A brute-force strategy of enumerating all itemsets (up to size 3) as candidates will produce 6C1 + 6C2 + 6C3 = 6+15+20 = 41 candidates.
- With the Apriori principle, this number decreases to 6C1 + 4C2 + 1 = 13 candidates, which represents a 68% reduction in the number of candidate itemsets.

Let C_k denote the set of candidate k-itemsets and F_k denote the set of frequent k-itemsets:

1. The algorithm initially makes a single pass over the data set to determine the support of each item. Upon completion of this step, the set of all frequent 1-itemsets, F_1, will be known (steps 1 and 2).
2. Next, the algorithm will iteratively generate new candidate k-itemsets using the frequent (k – 1)-itemsets found in the previous iteration (step 5). Candidate generation is implemented using a function called apriori-gen.
3. To count the support of the candidates, the algorithm needs to make an additional pass over the data set (steps 6–10). The subset function is used to determine all the candidate itemsets in C_k that are contained in each transaction t.
4. After counting their supports, the algorithm eliminates all candidate itemsets whose support counts are less than minsup (step 12).
5. The algorithm terminates when there are no new frequent itemsets generated, i.e., $F_k = \varnothing$ (step 13).

Algorithm 6.1 Frequent itemset generation of the *Apriori* algorithm.

1: $k = 1$.
2: $F_k = \{ i \mid i \in I \wedge \sigma(\{i\}) \geq N \times minsup\}$. {Find all frequent 1-itemsets}
3: **repeat**
4: $k = k + 1$.
5: $C_k = \text{apriori-gen}(F_{k-1})$. {Generate candidate itemsets}
6: **for** each transaction $t \in T$ **do**
7: $C_t = \text{subset}(C_k, t)$. {Identify all candidates that belong to t}
8: **for** each candidate itemset $c \in C_t$ **do**
9: $\sigma(c) = \sigma(c) + 1$. {Increment support count}
10: **end for**
11: **end for**
12: $F_k = \{ c \mid c \in C_k \wedge \sigma(c) \geq N \times minsup\}$. {Extract the frequent k-itemsets}
13: **until** $F_k = \emptyset$
14: Result $= \bigcup F_k$.

- The frequent itemset generation part of the Apriori algorithm has two important characteristics.
- First, it is a **level-wise algorithm**; i.e., it traverses the itemset lattice one level at a time, from frequent 1-itemsets to the maximum size of frequent itemsets.
- Second, it employs a generate-and-test strategy for finding frequent itemsets. At each iteration, new candidate itemsets are generated from the frequent itemsets found in the previous iteration. The support for each candidate is then counted and tested against the minsup threshold.
- The total number of iterations needed by the algorithm is $k_{max} + 1$, where k_{max} is the maximum size of the frequent itemsets.

Candidate Generation and Pruning

- The apriori-gen function shown in Step 5 of Algorithm 6.1 generates candidate itemsets by performing the following two operations:

1. Candidate Generation. This operation generates new candidate k- itemsets based on the frequent (k – 1)-itemsets found in the previous iteration.
2. Candidate Pruning. This operation eliminates some of the candidate k-itemsets using the support-based pruning strategy.

- To illustrate the candidate pruning operation, consider a candidate k-itemset, $X = \{i_1, i_2, \ldots i_k\}$. The algorithm must determine whether all of its proper subsets, $X - \{i_j\}$ ($\forall$j = 1,2,...,k), are frequent. If one of them is infrequent, then X is immediately pruned.
- This approach can effectively reduce the number of candidate itemsets considered during support counting. The complexity of this operation is O(k) for each candidate k-itemset.

- In principle, there are many ways to generate candidate itemsets. The following is a list of requirements for an effective candidate generation procedure:

1. It should avoid generating too many unnecessary candidates. A candidate itemset is unnecessary if at least one of its subsets is infrequent. Such a candidate is guaranteed to be infrequent according to the anti- monotone property of support.
2. It must ensure that the candidate set is complete, i.e., no frequent itemsets are left out by the candidate generation procedure. To ensure completeness, the set of candidate itemsets must subsume the set of all frequent itemsets, i.e., $\forall k : F_k \subseteq C_k$.
3. It should not generate the same candidate itemset more than once. For example, the candidate itemset {a,b,c,d} can be generated in many ways—by merging {a, b, c} with {d}, {b, d} with {a, c}, {c} with {a, b, d}, etc. Generation of duplicate candidates leads to wasted computations and thus should be avoided for efficiency reasons.

Brute-Force Method

- The brute-force method considers every k-itemset as a potential candidate and then applies the candidate pruning step to remove any unnecessary candidates.
- The number of candidate itemsets generated at level k is equal to $^{d}C_{k}$, where d is the total number of items.
- Candidate pruning becomes extremely expensive because a large number of itemsets must be examined.
- Given that the amount of computations needed for each candidate is O(k), the overall complexity of this method is $O(\sum_{k=1}^{d} k \times \binom{d}{k}) = O(d.2^{d-1})$

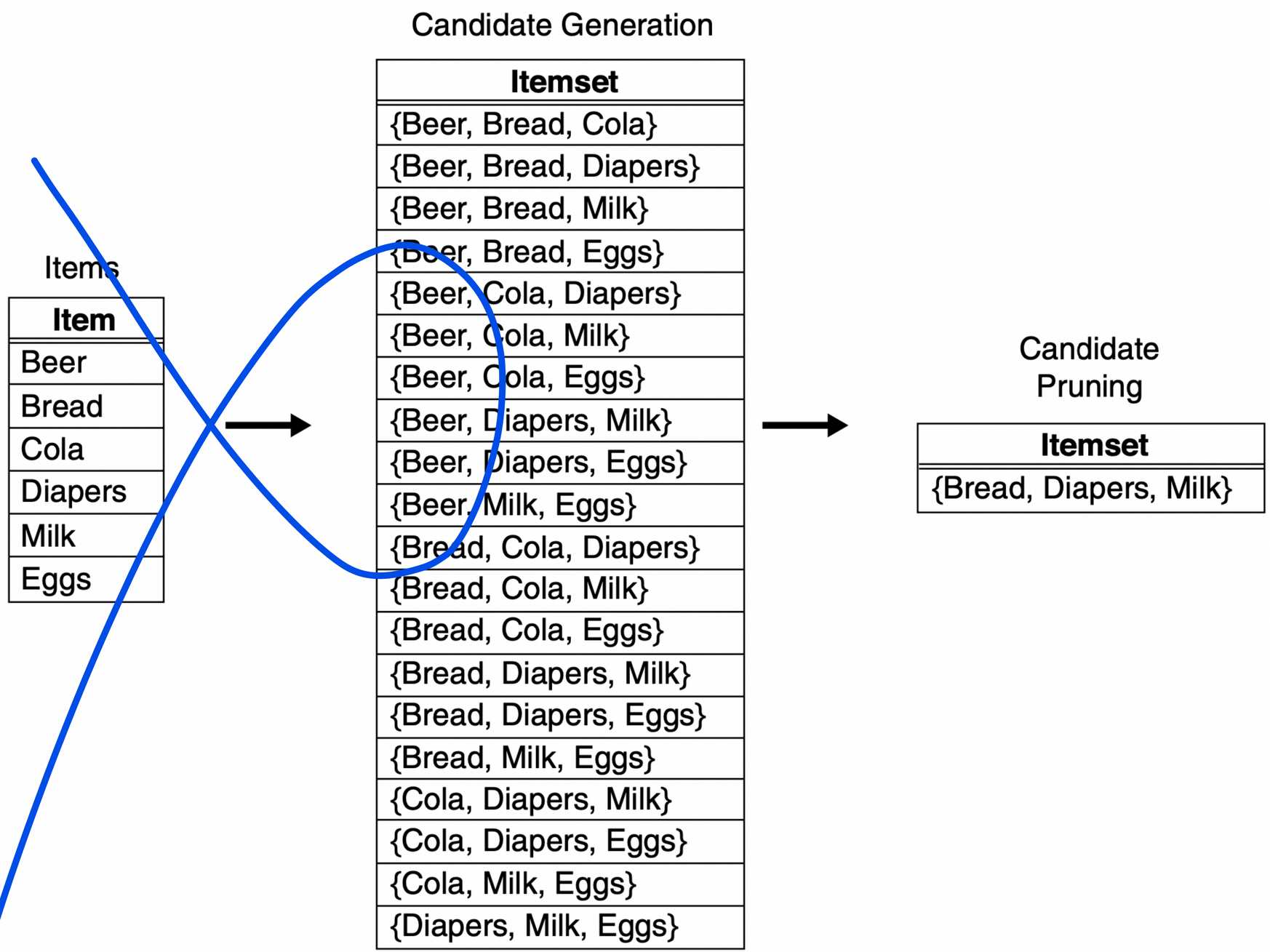


Figure 6.6. A brute-force method for generating candidate 3-itemsets.

$F_{k-1} \times F_1$ Method

- An alternative method for candidate generation is to extend each frequent (k – 1)-itemset with other frequent items.
- This method will produce O($|F_{k-1}| \times |F_1|$) candidate k-itemsets, where $|F_j|$ is the number of frequent j-itemsets.
- The overall complexity of this step is $O(\sum_k k|F_{k-1}||F_1|)$.
- The procedure is complete because every frequent k-itemset is composed of a frequent (k – 1)-itemset and a frequent 1-itemset. Therefore, all frequent k-itemsets are part of the candidate k-itemsets generated by this procedure.
- This approach, however, does not prevent the same candidate itemset from being generated more than once. For instance, {Bread, Diapers, Milk} can be generated by merging {Bread, Diapers} with {Milk}, {Bread, Milk} with {Diapers}, or {Diapers, Milk} with {Bread}.

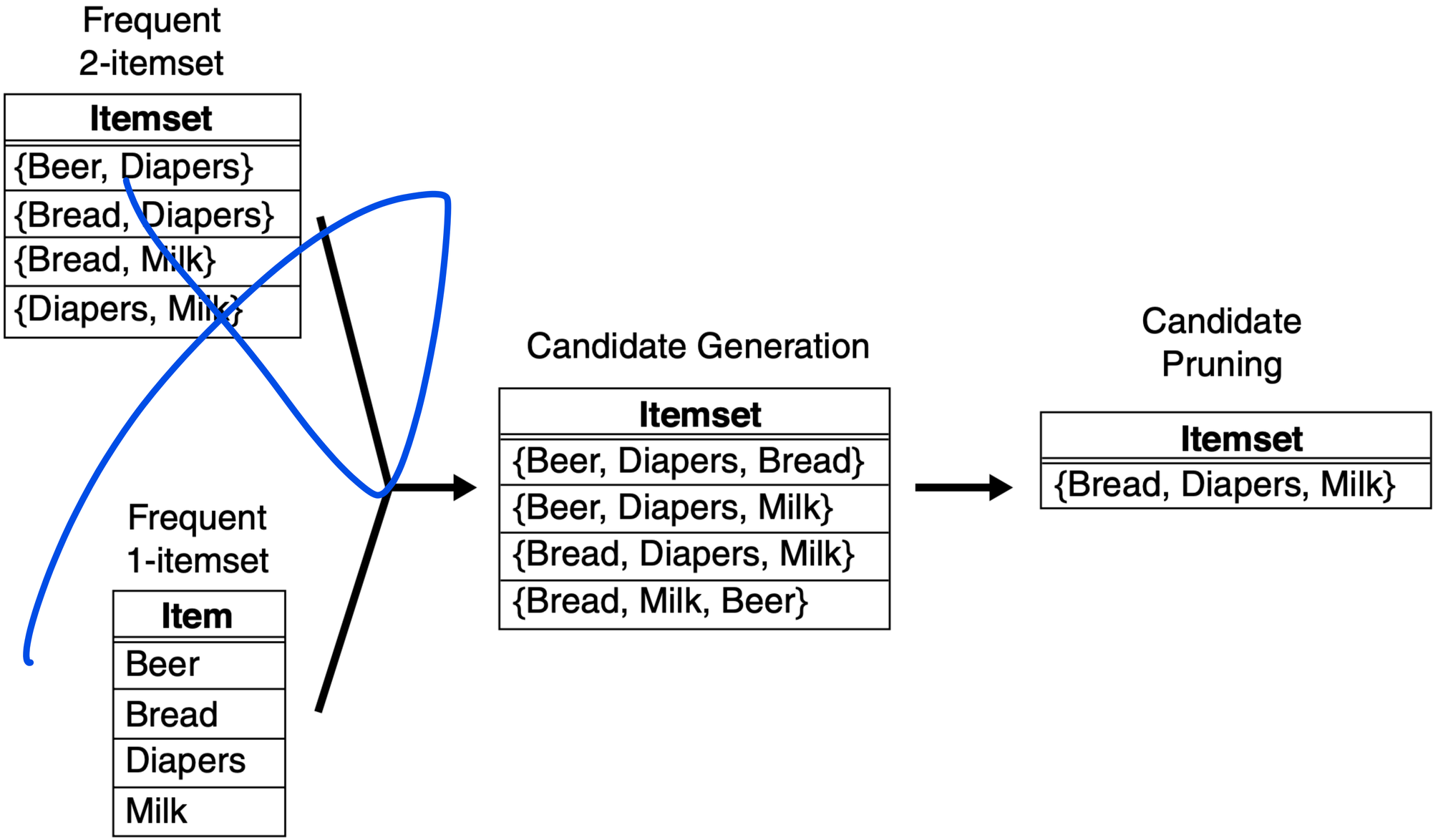


Figure 6.7. Generating and pruning candidate k-itemsets by merging a frequent $(k-1)$-itemset with a frequent item. Note that some of the candidates are unnecessary because their subsets are infrequent.

- One way to avoid generating duplicate candidates is by ensuring that the items in each frequent itemset are kept sorted in their lexicographic order.
- Each frequent (k–1)-itemset X is then extended with frequent items that are lexicographically larger than the items in X.
- For example, the itemset {Bread, Diapers} can be augmented with {Milk} since Milk is lexicographically larger than Bread and Diapers. However, we should not augment {Diapers, Milk} with {Bread} nor {Bread, Milk} with {Diapers} because they violate the lexicographic ordering condition.
- While this procedure is a substantial improvement over the brute-force method, it can still produce a large number of unnecessary candidates. For example, the candidate itemset obtained by merging {Beer, Diapers} with {Milk} is unnecessary because one of its subsets, {Beer, Milk}, is infrequent.

$F_{k-1} \times F_{k-1}$ Method

- The candidate generation procedure in the apriori-gen function merges a pair of frequent (k–1)-itemsets only if their first k–2 items are identical.
- Let $A = \{a_1, a_2, \ldots a_{k-1}\}$ and $B = \{b_1, b_2, \ldots b_{k-1}\}$ be a pair of frequent (k – 1)-itemsets. A and B are merged if they satisfy the following conditions: $a_i = b_i$ (for i = 1,2,...,k – 2) and $a_{k-1} \neq b_{k-1}$.
- In Figure 6.8, the frequent itemsets {Bread, Diapers} and {Bread, Milk} are merged to form a candidate 3-itemset {Bread, Diapers, Milk}. The algorithm does not have to merge {Beer, Diapers} with {Diapers, Milk} because the first item in both itemsets is different. Indeed, if {Beer, Diapers, Milk} is a viable candidate, it would have been obtained by merging {Beer, Diapers} with {Beer, Milk} instead.
- This example illustrates both the completeness of the candidate generation procedure and the advantages of using lexicographic ordering to prevent duplicate candidates.
- However, because each candidate is obtained by merging a pair of frequent (k–1)-itemsets, an additional candidate pruning step is needed to ensure that the remaining k – 2 subsets of the candidate are frequent.

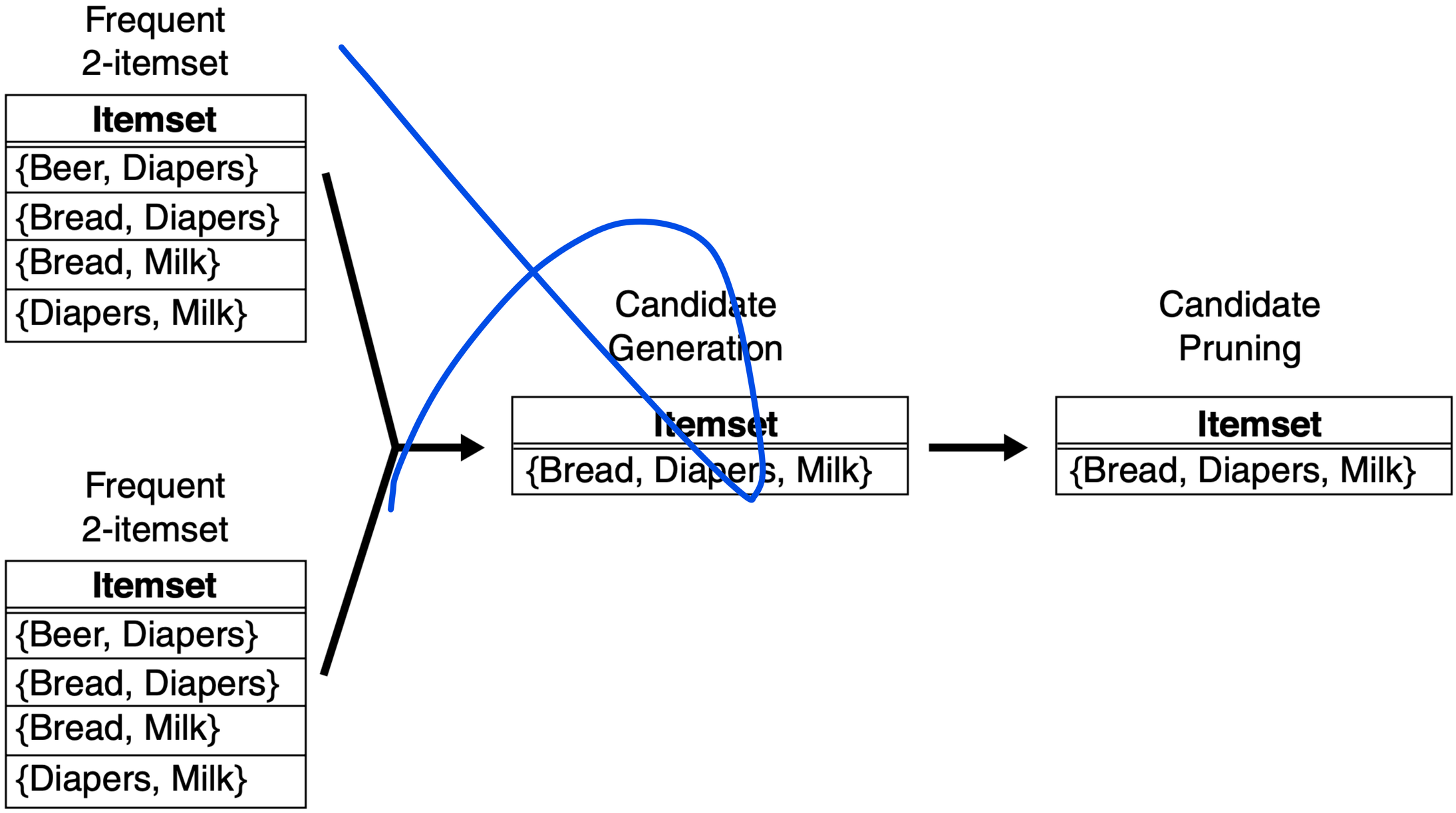


Figure 6.8. Generating and pruning candidate k-itemsets by merging pairs of frequent $(k-1)$-itemsets.

Support Counting

- Support counting is the process of determining the frequency of occurrence for every candidate itemset that survives the candidate pruning step of the apriori-gen function.
- Support counting is implemented in steps 6 through 11 of Algorithm 6.1.
- One approach is Brute force discussed earlier.
- An alternative approach is to enumerate the itemsets contained in each transaction and use them to update the support counts of their respective candidate itemsets.
- Consider a transaction t that contains five items, {1, 2, 3, 5, 6}. There are $\binom{5}{3} = 10$ itemsets of size 3 contained in this transaction.
- Some of the itemsets may correspond to the candidate 3-itemsets under investigation, in which case, their support counts are incremented. Other subsets of t that do not correspond to any candidates can be ignored.

- Figure 6.9 shows a systematic way for enumerating the 3-itemsets contained in t. Assuming that each itemset keeps its items in increasing lexicographic order, an itemset can be enumerated by specifying the smallest item first, followed by the larger items.
- For instance, given t = {1, 2, 3, 5, 6}, all the 3- itemsets contained in t must begin with item 1, 2, or 3. It is not possible to construct a 3-itemset that begins with items 5 or 6 because there are only two items in t whose labels are greater than or equal to 5.
- The number of ways to specify the first item of a 3-itemset contained in t is illustrated by the Level 1 prefix structures depicted in Figure 6.9.
- After fixing the first item, the prefix structures at Level 2 represent the number of ways to select the second item.
- Finally, the prefix structures at Level 3 represent the complete set of 3-itemsets contained in t.
- We still have to determine whether each enumerated 3-itemset corresponds to an existing candidate itemset. If it matches one of the candidates, then the support count of the corresponding candidate is incremented.

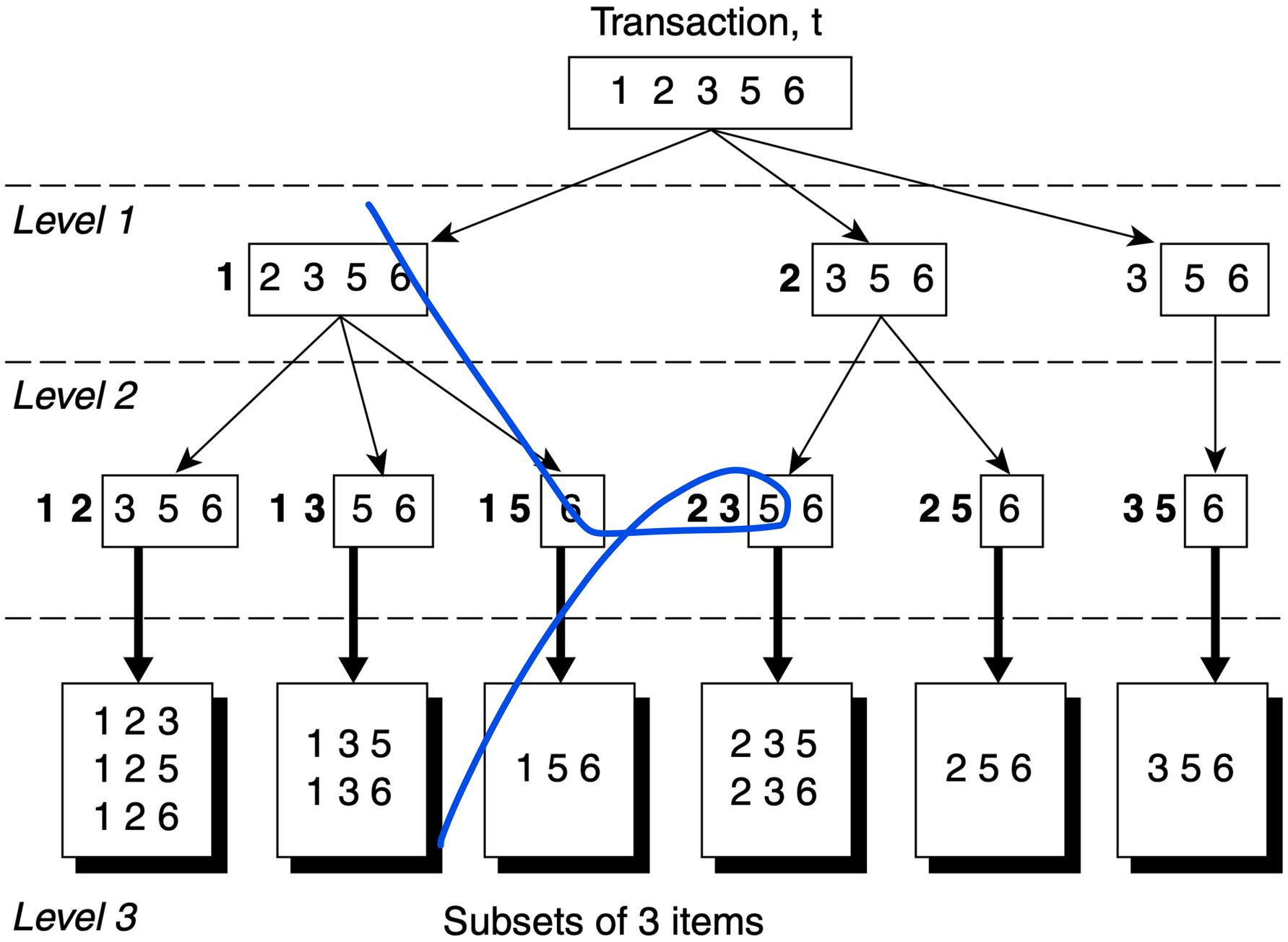


Figure 6.9. Enumerating subsets of three items from a transaction t.

Support Counting Using a Hash Tree

- Figure 6.11 shows an example of a hash tree structure.
- Each internal node of the tree uses the following hash function, h(p) = p mod 3, to determine which branch of the current node should be followed next.
- For example, items 1, 4, and 7 are hashed to the same branch (i.e., the leftmost branch) because they have the same remainder after dividing the number by 3.
- All candidate itemsets are stored at the leaf nodes of the hash tree.
- The hash tree shown in Figure 6.11 contains 15 candidate 3-itemsets, distributed across 9 leaf nodes.
- A node is split into three branches when there are itemsets in it belonging to all three branches.

- In the Apriori algorithm, candidate itemsets are partitioned into different buckets and stored in a hash tree.
- During support counting, itemsets contained in each transaction are also hashed into their appropriate buckets.
- That way, instead of comparing each itemset in the transaction with every candidate itemset, it is matched only against candidate itemsets that belong to the same bucket.
- Consider a transaction, t = {1, 2, 3, 5, 6}.
- Recall that the 3-itemsets contained in t must begin with items 1, 2, or 3, as indicated by the Level 1 prefix structures shown in Figure 6.9.
- To update the support counts of the candidate itemsets, the hash tree must be traversed in such a way that all the leaf nodes containing candidate 3-itemsets belonging to t must be visited at least once.

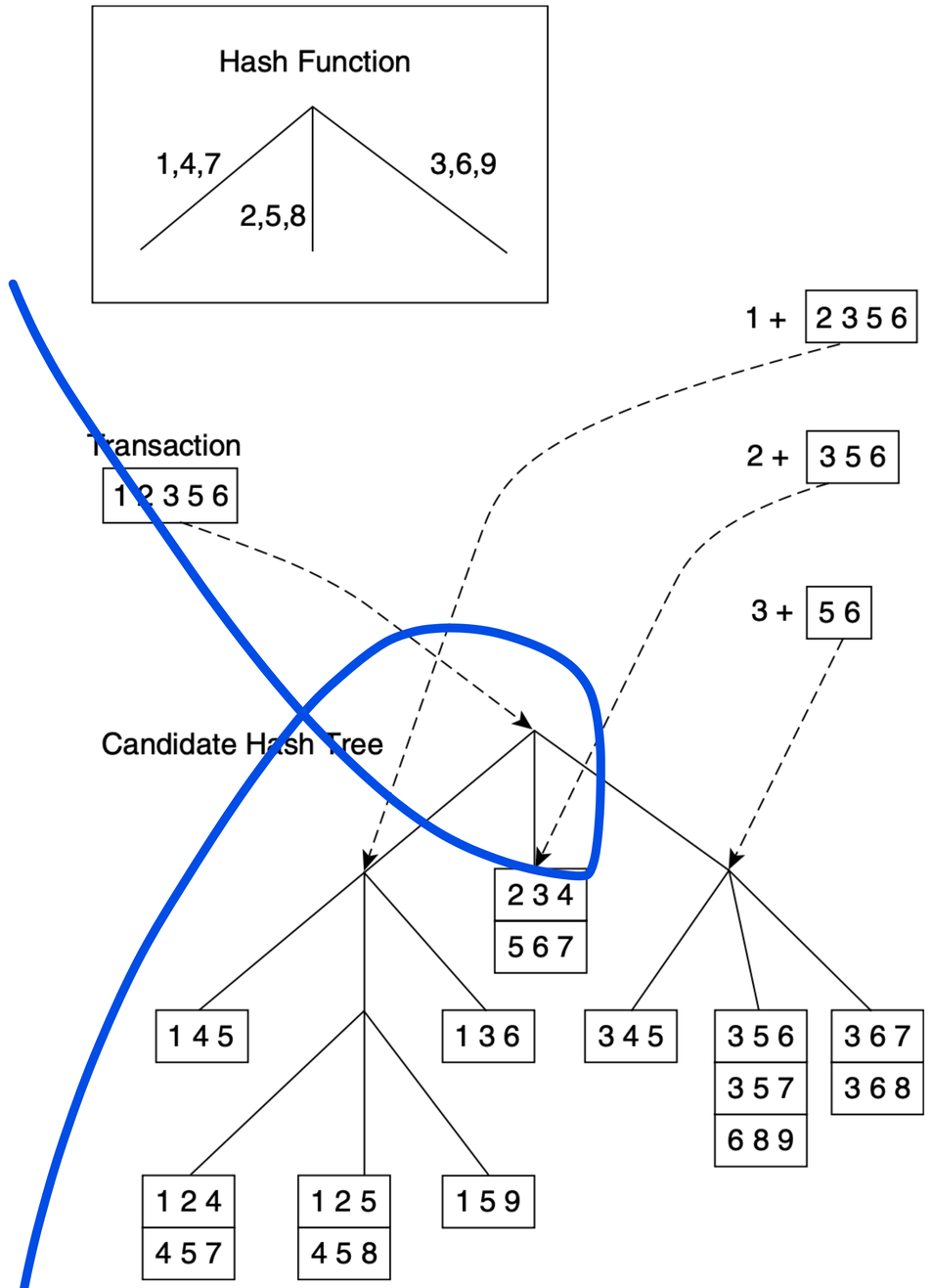


Figure 6.11. Hashing a transaction at the root node of a hash tree.

- At the root node of the hash tree, the items 1, 2, and 3 of the transaction are hashed separately. Item 1 is hashed to the left child of the root node, item 2 is hashed to the middle child, and item 3 is hashed to the right child.
- At the next level of the tree, the transaction is hashed on the second item listed in the Level 2 structures.
- For example, after hashing on item 1 at the root node, items 2, 3, and 5 of the transaction are hashed. Items 2 and 5 are hashed to the middle child, while item 3 is hashed to the right child.
- This process continues until the leaf nodes of the hash tree are reached.
- The candidate itemsets stored at the visited leaf nodes are compared against the transaction. If a candidate is a subset of the transaction, its support count is incremented.
- In this example, 5 out of the 9 leaf nodes are visited and 9 out of the 15 itemsets are compared against the transaction.

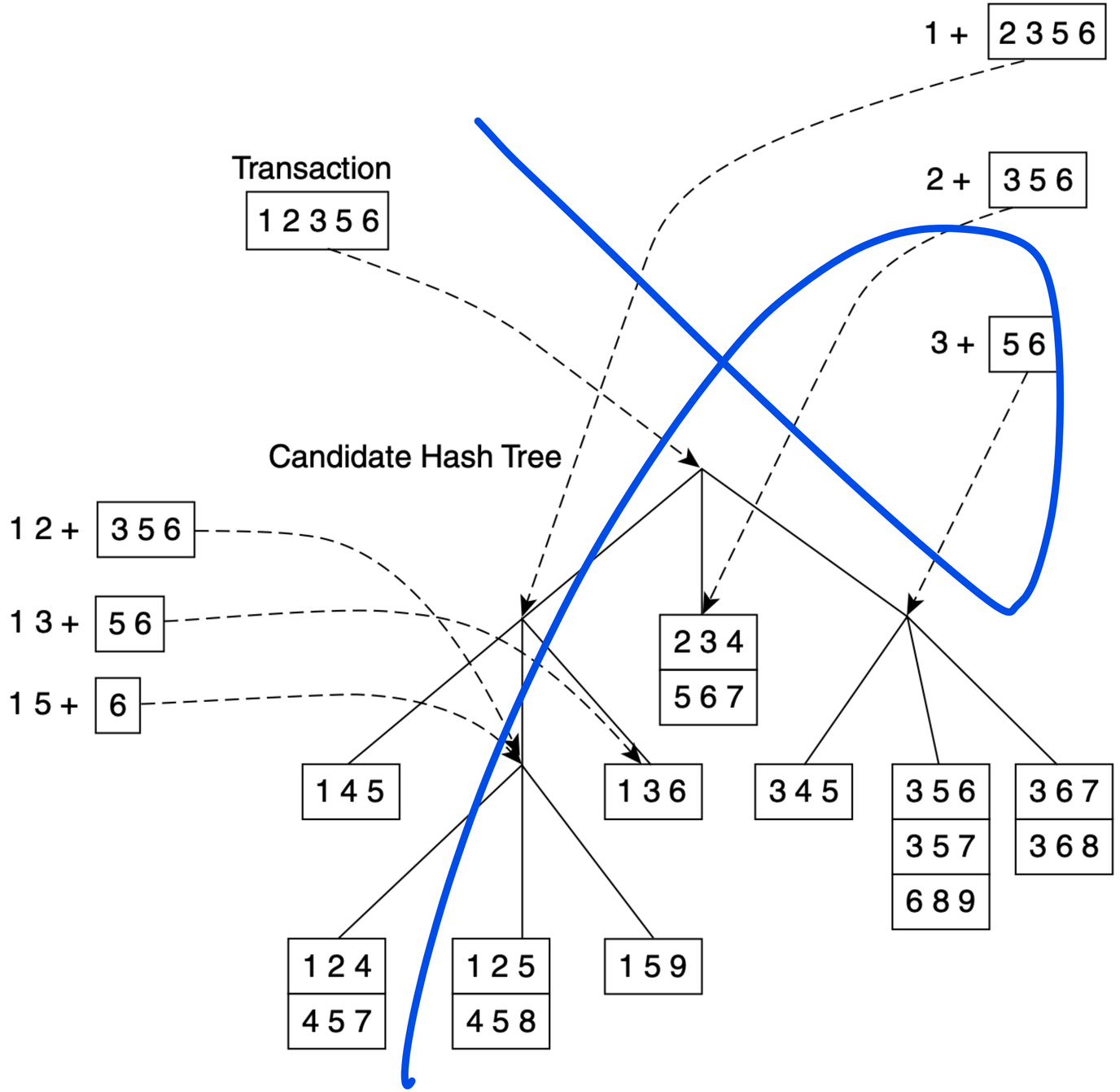


Figure 6.12. Subset operation on the leftmost subtree of the root of a candidate hash tree.

1. **Generation of frequent 1-itemsets** For each transaction, we need to update the support count for every item present in the transaction. Assuming that w is the average transaction width, this operation requires O(Nw) time, where N is the total number of transactions.

2. **Candidate generation** To generate candidate k-itemsets, pairs of frequent (k – 1)-itemsets are merged to determine whether they have at least k – 2 items in common. Each merging operation requires at most k – 2 equality comparisons. In the best-case scenario, every merging step produces a viable candidate k-itemset. In the worst-case scenario, the algorithm must merge every pair of frequent (k–1)-itemsets found in the previous iteration. Therefore, the overall cost of merging frequent itemsets is

$$\sum_{k=2}^{w}(k-2)\,|C_k| < \text{Cost of merging} < \sum_{k=2}^{w}(k-2)\,|F_{k-1}|^2$$

A hash tree is also constructed during candidate generation to store the candidate itemsets. Because the maximum depth of the tree is k, the cost for populating the hash tree with candidate itemsets is $O(\sum_{k=2}^{w} k\,|C_k|)$.

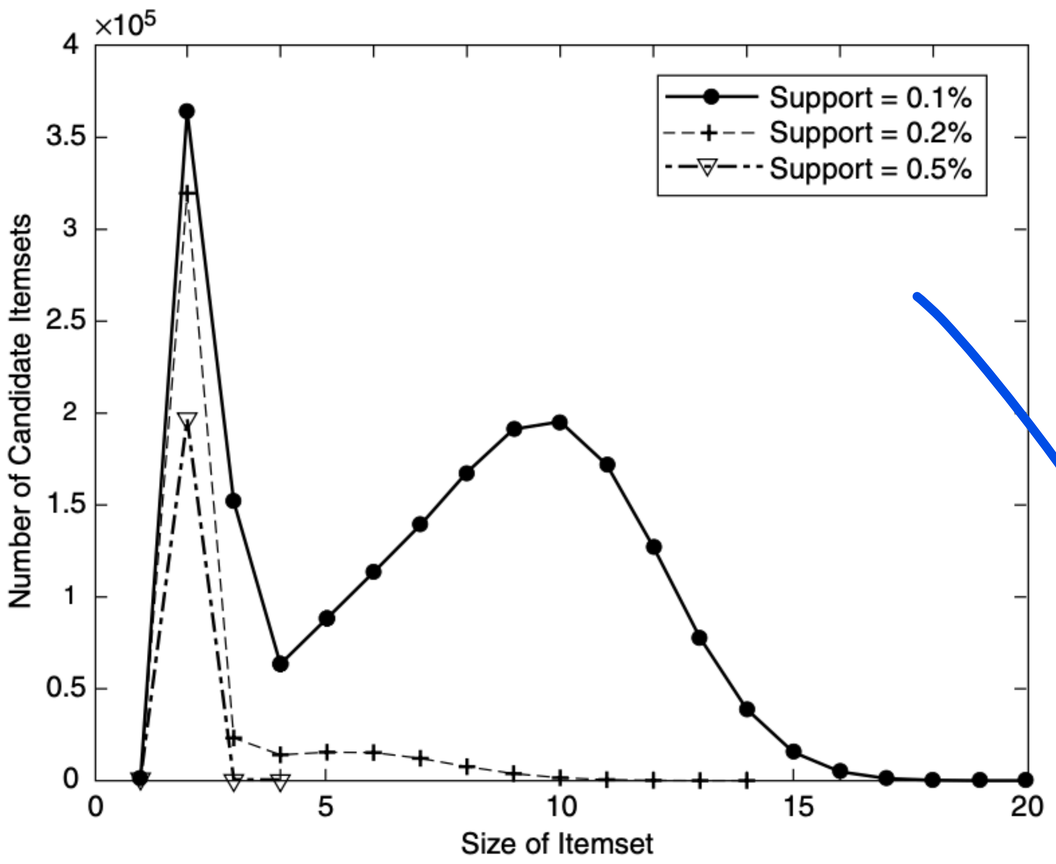
During candidate pruning, we need to verify that the k – 2 subsets of every candidate k-itemset are frequent. Since the cost for looking up a candidate in a hash tree is O(k), the candidate pruning step requires $O(\sum_{k=2}^{w} k(k-2)\,|C_k|)$

3. **Support counting** Each transaction of length |t| produces $\binom{|t|}{k}$ itemsets of size k. This is also the effective number of hash tree traversals performed for each transaction. The cost for support counting is $O(N\sum_{k}\binom{w}{k}\alpha_k)$, where w is the maximum transaction width and α_k is the cost for updating the support count of a candidate k-itemset in the hash tree.

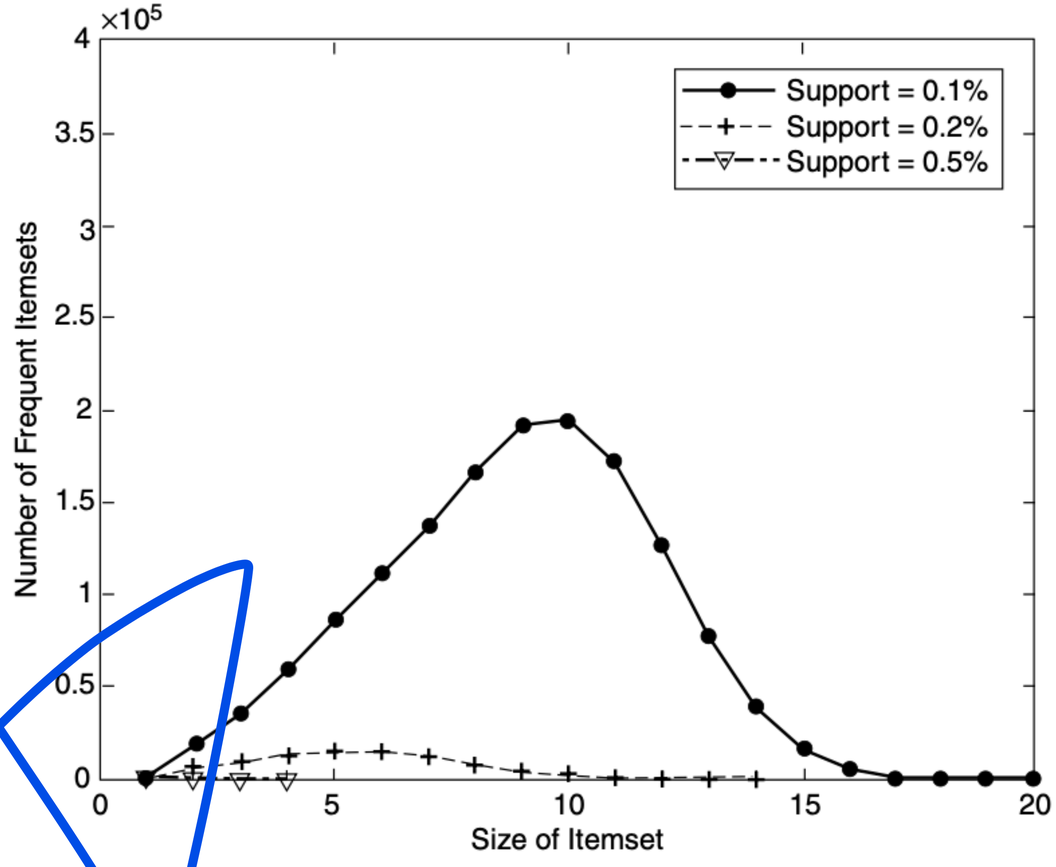
Computational Complexity

The computational complexity of the Apriori algorithm can be affected by the following factors.

- **Support Threshold** Lowering the support threshold often results in more itemsets being declared as frequent. This has an adverse effect on the computational complexity of the algorithm because more candidate itemsets must be generated and counted. The maximum size of frequent itemsets also tends to increase with lower support thresholds. As the maximum size of the frequent itemsets increases, the algorithm will need to make more passes over the data set.
- **Number of Items (Dimensionality)** As the number of items increases, more space will be needed to store the support counts of items. If the number of frequent items also grows with the dimensionality of the data, the computation and I/O costs will increase because of the larger number of candidate itemsets generated by the algorithm.
- **Number of Transactions** Since the Apriori algorithm makes repeated passes over the data set, its run time increases with a larger number of transactions.



(a) Number of candidate itemsets.

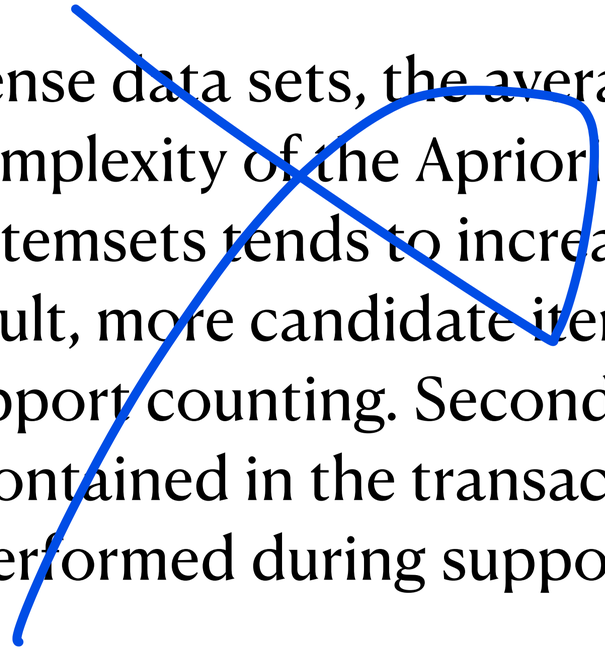


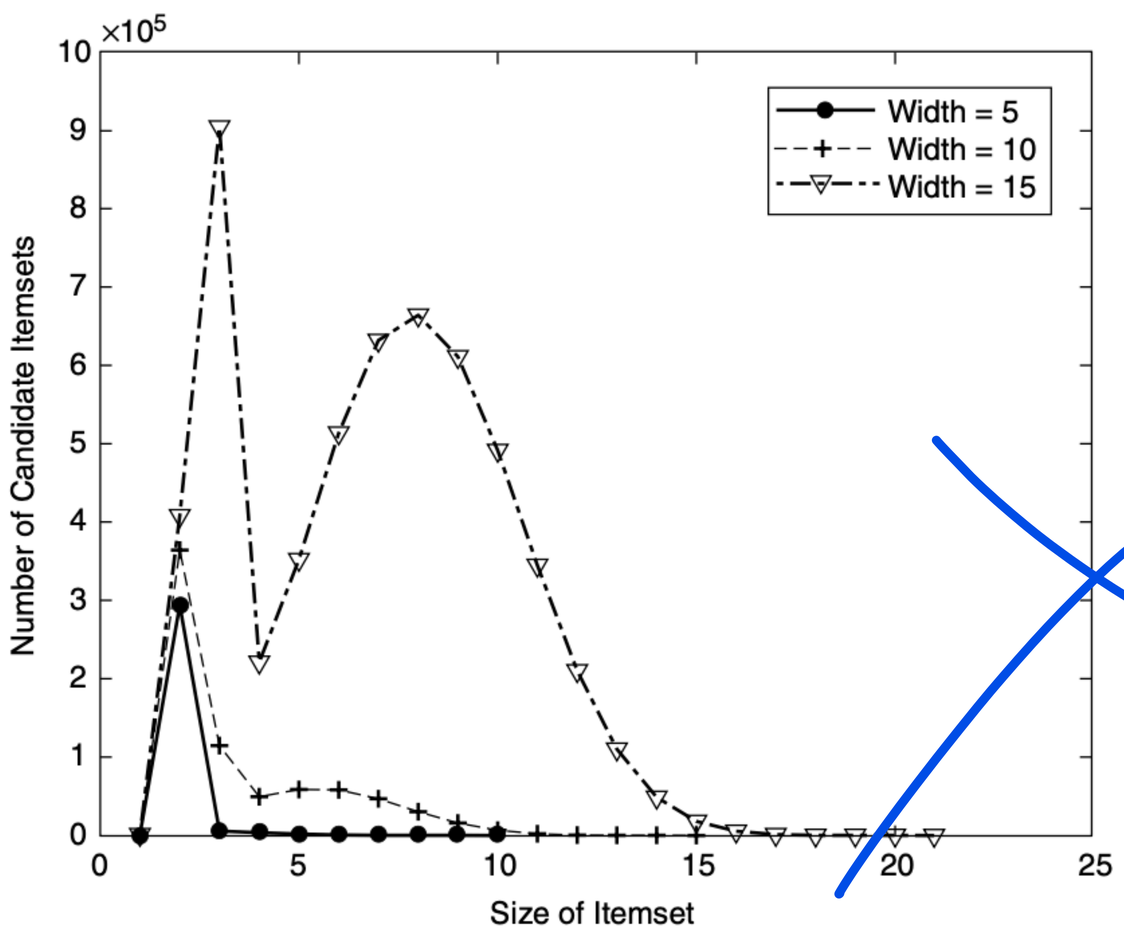
(b) Number of frequent itemsets.

Figure 6.13. Effect of support threshold on the number of candidate and frequent itemsets.

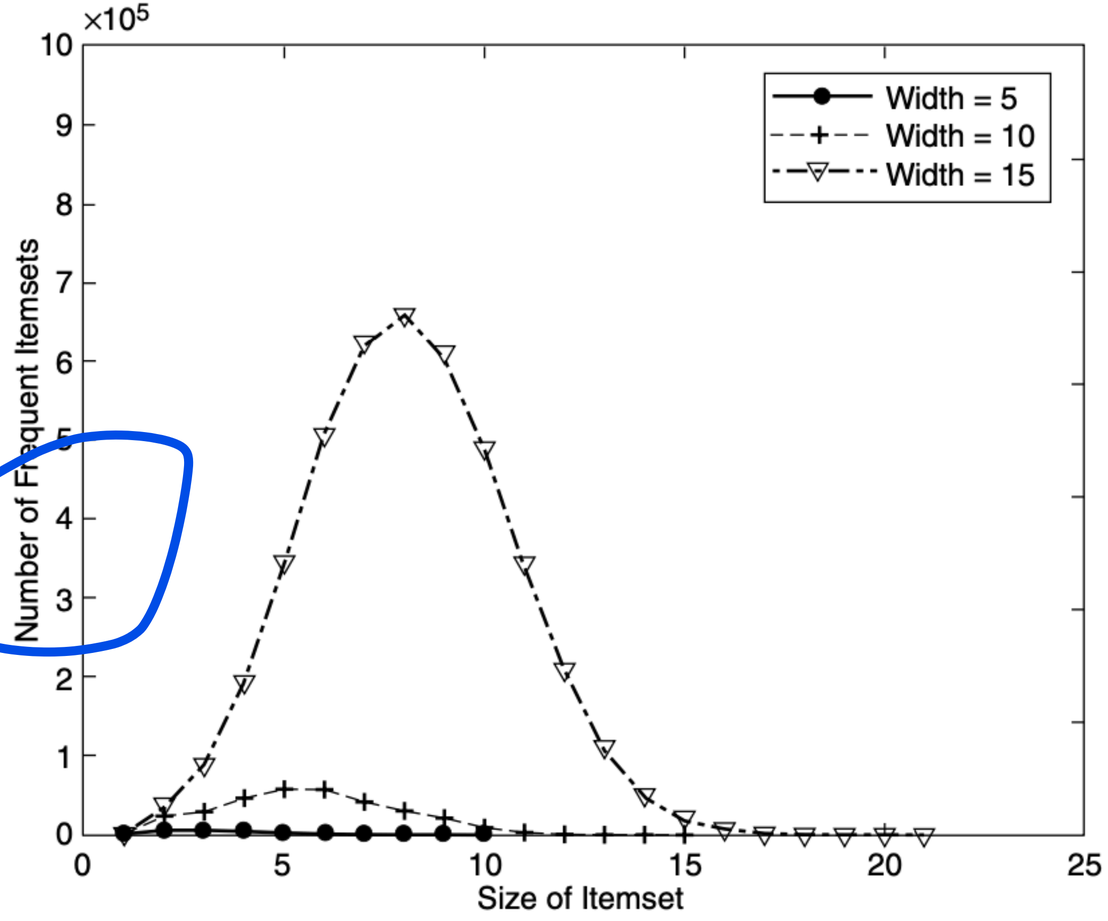
Computational Complexity

- **Average Transaction Width** For dense data sets, the average transaction width can be very large. This affects the complexity of the Aprior algorithm in two ways. First, the maximum size of frequent itemsets tends to increase as the average transaction width increases. As a result, more candidate itemsets must be examined during candidate generation and support counting. Second, as the transaction width increases, more itemsets are contained in the transaction. This will increase the number of hash tree traversals performed during support counting.





(a) Number of candidate itemsets.



(b) Number of Frequent Itemsets.

Figure 6.14. Effect of average transaction width on the number of candidate and frequent itemsets.

Rule Generation

- Each frequent k-itemset, Y , can produce up to $2^k - 2$ association rules, ignoring rules that have empty antecedents or consequents ($\emptyset \longrightarrow Y$ or $Y \longrightarrow \emptyset$).
- An association rule can be extracted by partitioning the itemset Y into two non-empty subsets, X and Y –X, such that X –→ Y –X satisfies the confidence threshold.
- Note that all such rules must have already met the support threshold because they are generated from a frequent itemset.
- Let X = {1, 2, 3} be a frequent itemset. There are six candidate association rules that can be generated from X: {1, 2} –→ {3}, {1, 3} –→ {2}, {2,3} –→ {1}, {1} –→ {2,3}, {2} –→ {1,3}, and {3} –→ {1,2}. As each of their support is identical to the support for X, the rules must satisfy the support threshold.

- Computing the confidence of an association rule does not require additional scans of the transaction data set.
- Consider the rule $\{1, 2\} \longrightarrow \{3\}$, which is generated from the frequent itemset $X = \{1, 2, 3\}$. The confidence for this rule is $\sigma(\{1, 2, 3\})/\sigma(\{1, 2\})$.
- Because $\{1, 2, 3\}$ is frequent, the anti-monotone property of support ensures that $\{1,2\}$ must be frequent, too.
- Since the support counts for both itemsets were already found during frequent itemset generation, there is no need to read the entire data set again.

Confidence-Based Pruning

- Unlike the support measure, confidence does not have any monotone property. For example, the confidence for $X \rightarrow Y$ can be larger, smaller, or equal to the confidence for another rule $\tilde{X} \rightarrow \tilde{Y}$, where $\tilde{X} \subseteq X$ and $\tilde{Y} \subseteq Y$.
- Let {1,2} -> {3,4} be the rule with confidence c1. Then
- $c1 = \sigma(\{1,2,3,4\})/\sigma(\{1,2\}) = 5/10 = 0.5$
- Then, for rule {1} -> {3}, confidence c2 will be $c1 = \sigma(\{1,3\})/\sigma(\{1\}) = 5/12 = 0.41$
- Or $c1 = \sigma(\{1,3\})/\sigma(\{1\}) = 6/11 = 0.54$
- Or $c1 = \sigma(\{1,3\})/\sigma(\{1\}) = 5/10 = 0.5$

- Nevertheless, if we compare rules generated from the same frequent itemset I , the following theorem holds for the confidence measure.

If a rule $X \rightarrow I - X$ does not satisfy the confidence threshold, then any rule $X' \rightarrow I - X'$, where X' is a subset of X, must not satisfy the confidence threshold as well.

- Consider the following two rules: $X \rightarrow I - X$ and $X \rightarrow I - X'$, where $X' \subset X$.
- The confidence of the rules are $\sigma(I)/\sigma(X)$ and $\sigma(I)/\sigma(X')$, respectively.
- Since X' is a subset of X, $\sigma(X') \geq \sigma(X)$. Therefore, the former rule cannot have a higher confidence than the latter rule.

Rule Generation in Apriori Algorithm

- The Apriori algorithm uses a level-wise approach for generating association rules, where each level corresponds to the number of items that belong to the rule consequent.
- Initially, all the high-confidence rules that have only one item in the rule consequent are extracted. These rules are then used to generate new candidate rules.
- For example, if {acd} −→ {b} and {abd} −→ {c} are high-confidence rules, then the candidate rule {ad} −→ {bc} is generated by merging the consequents of both rules.
- Figure 6.15 shows a lattice structure for the association rules generated from the frequent itemset {a, b, c, d}.
- If any node in the lattice has low confidence, then according to previous Theorem, the entire subgraph spanned by the node can be pruned immediately.
- Suppose the confidence for {bcd} −→ {a} is low. All the rules containing item a in its consequent, including {cd} −→ {ab}, {bd} −→ {ac}, {bc} −→ {ad}, and {d} −→ {abc} can be discarded.

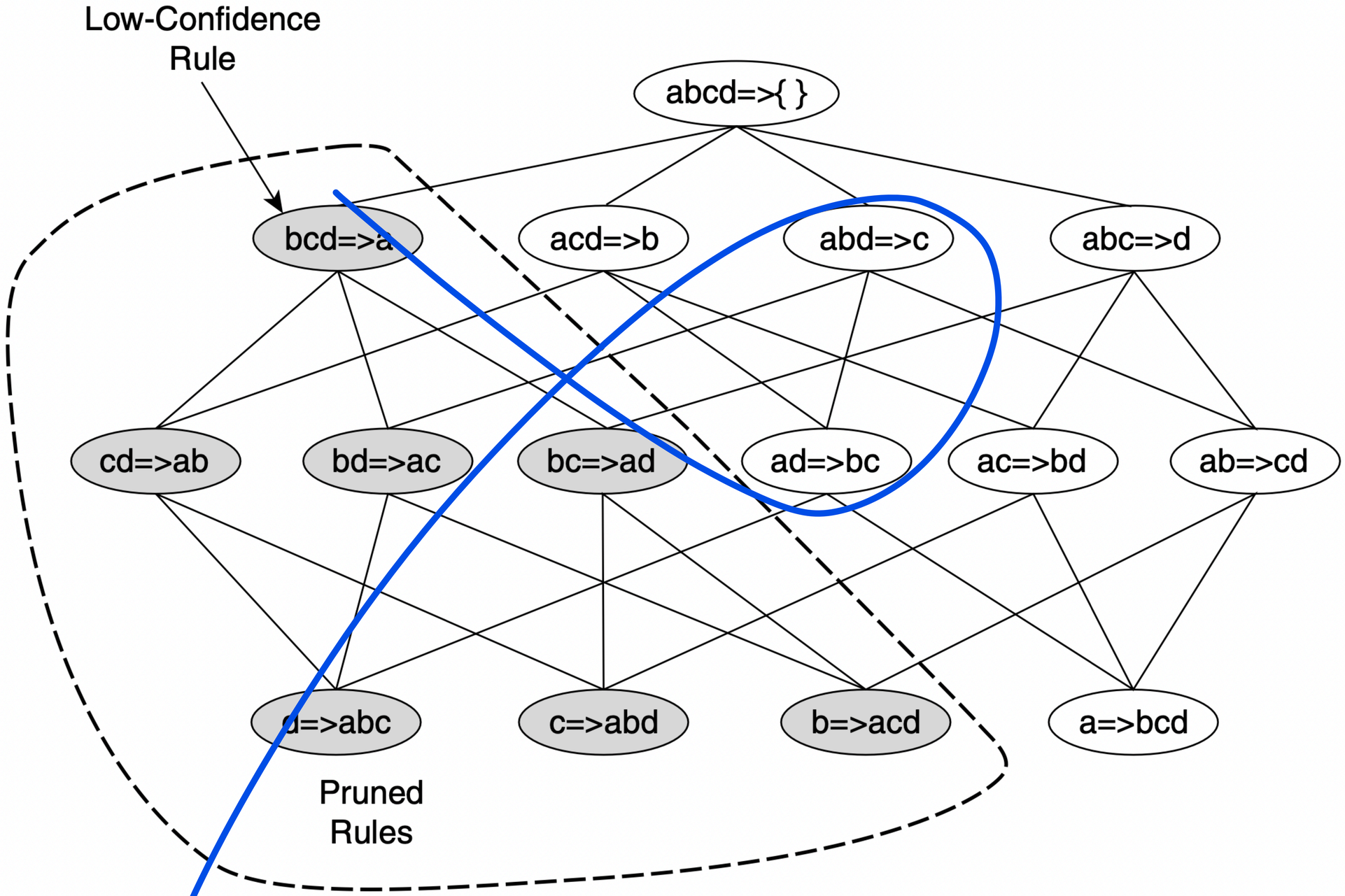


Figure 6.15. Pruning of association rules using the confidence measure.

- A pseudocode for the rule generation step is shown in Algorithms 6.2 and 6.3.
- In rule generation, we do not have to make additional passes over the data set to compute the confidence of the candidate rules.
- Instead, we determine the confidence of each rule by using the support counts computed during frequent itemset generation.

Algorithm 6.2 Rule generation of the *Apriori* algorithm.

1: **for** each frequent k-itemset f_k, $k \geq 2$ **do**
2: $H_1 = \{i \mid i \in f_k\}$ {1-item consequents of the rule.}
3: **call** `ap-genrules`$(f_k, H_1.)$
4: **end for**

Algorithm 6.3 Procedure `ap-genrules`(f_k, H_m).

1: $k = |f_k|$ {size of frequent itemset.}
2: $m = |H_m|$ {size of rule consequent.}
3: **if** $k > m + 1$ **then**
4: $H_{m+1} =$ apriori-gen(H_m).
5: **for** each $h_{m+1} \in H_{m+1}$ **do**
6: $conf = \sigma(f_k)/\sigma(f_k - h_{m+1})$.
7: **if** $conf \geq minconf$ **then**
8: **output** the rule $(f_k - h_{m+1}) \longrightarrow h_{m+1}$.
9: **else**
10: **delete** h_{m+1} from H_{m+1}.
11: **end if**
12: **end for**
13: **call** ap-genrules$(f_k, H_{m+1}.)$
14: **end if**

Cluster Analysis

- Cluster analysis divides data into groups (clusters) that are meaningful, useful, or both.
- Classes, or conceptually meaningful groups of objects that share common characteristics, play an important role in how people analyze and describe the world.
- In the context of understanding data, clusters are potential classes and cluster analysis is the study of techniques for automatically finding classes.
- **Information Retrieval** - The World Wide Web consists of billions of Web pages, and the results of a query to a search engine can return thousands of pages. Clustering can be used to group these search results into a small number of clusters, each of which captures a particular aspect of the query. For instance, a query of “movie” might return Web pages grouped into categories such as reviews, trailers, stars, and theaters. Each category (cluster) can be broken into subcategories (sub- clusters), producing a hierarchical structure that further assists a user’s exploration of the query results.

- **Business**. Businesses collect large amounts of information on current and potential customers. Clustering can be used to segment customers into a small number of groups for additional analysis and marketing activities.
- **Cluster analysis** groups data objects based only on information found in the data that describes the objects and their relationships.
- The goal is that the objects within a group be similar (or related) to one another and different from (or unrelated to) the objects in other groups.
- The greater the similarity (or homogeneity) within a group and the greater the difference between groups, the better or more distinct the clustering.
- The figure illustrates that the definition of a cluster is imprecise and that the best definition depends on the nature of data and the desired results.

(a) Original points.

(b) Two clusters.

(c) Four clusters.

(d) Six clusters.

Figure 8.1. Different ways of clustering the same set of points.

- Clustering can be regarded as a form of classification in that it creates a labeling of objects with class (cluster) labels. However, it derives these labels only from the data.
- In contrast, classification is supervised classification; i.e., new, unlabeled objects are assigned a class label using a model developed from objects with known class labels.
- For this reason, cluster analysis is sometimes referred to as unsupervised classification.
- The term **partitioning** is often used in connection with techniques that divide graphs into subgraphs and that are not strongly connected to clustering.
- **Segmentation** often refers to the division of data into groups using simple techniques; e.g., an image can be split into segments based only on pixel intensity and color, or people can be divided into groups based on their income.

Different Types of Clusterings

Hierarchical versus Partitional

- A **partitional** clustering is simply a division of the set of data objects into non-overlapping subsets (clusters) such that each data object is in exactly one subset.
- If we permit clusters to have subclusters, then we obtain a **hierarchical** clustering, which is a set of nested clusters that are organized as a tree.
- Each node (cluster) in the tree (except for the leaf nodes) is the union of its children (subclusters), and the root of the tree is the cluster containing all the objects.
- Often, but not always, the leaves of the tree are singleton clusters of individual data objects.
- A hierarchical clustering can be viewed as a sequence of partitional clusterings and a partitional clustering can be obtained by taking any member of that sequence; i.e., by cutting the hierarchical tree at a particular level.

Different Types of Clusterings

Exclusive versus Overlapping versus Fuzzy

- An **exclusive** clustering assign each object to a single cluster.
- An **overlapping** or non-exclusive clustering is used to reflect the fact that an object can simultaneously belong to more than one group (class).
- For instance, a person at a university can be both an enrolled student and an employee of the university.
- A non-exclusive clustering is also often used when, for example, an object is "between" two or more clusters and could reasonably be assigned to any of these clusters.
- In a **fuzzy** clustering, every object belongs to every cluster with a membership weight that is between 0 (absolutely doesn't belong) and 1 (absolutely belongs). In other words, clusters are treated as fuzzy sets - in which an object belongs to any set with a weight that is between 0 and 1.
- In fuzzy clustering, we often impose the additional constraint that the sum of the weights for each object must equal 1.

Different Types of Clusterings

Exclusive versus Overlapping versus Fuzzy

- Similarly, probabilistic clustering techniques compute the probability with which each point belongs to each cluster, and these probabilities must also sum to 1.
- Because the membership weights or probabilities for any object sum to 1, a fuzzy or probabilistic clustering does not address true multiclass situations, such as the case of a student employee, where an object belongs to multiple classes.
- Instead, these approaches are most appropriate for avoiding the arbitrariness of assigning an object to only one cluster when it may be close to several.
- In practice, a fuzzy or probabilistic clustering is often converted to an exclusive clustering by assigning each object to the cluster in which its membership weight or probability is highest.

Different Types of Clusterings

Complete versus Partial

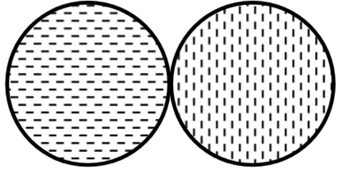
- A **complete** clustering assigns every object to a cluster, whereas a **partial** clustering does not.
- The motivation for a partial clustering is that some objects in a data set may not belong to well-defined groups. Many times objects in the data set may represent noise, outliers, or "uninteresting background."
- For example, some newspaper stories may share a common theme, such as global warming, while other stories are more generic or one-of-a-kind. Thus, to find the important topics in last month's stories, we may want to search only for clusters of documents that are tightly related by a common theme.
- In other cases, a complete clustering of the objects is desired. For example, an application that uses clustering to organize documents for browsing needs to guarantee that all documents can be browsed.

Different Types of Clusters

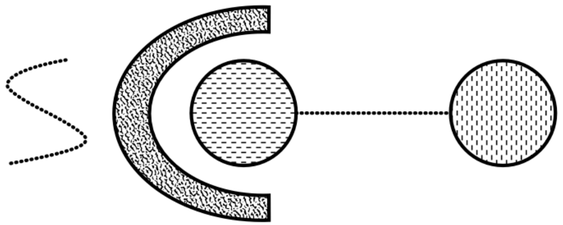
Well-Separated

- A cluster is a set of objects in which each object is closer (or more similar) to every other object in the cluster than to any object not in the cluster.
- Sometimes a threshold is used to specify that all the objects in a cluster must be sufficiently close (or similar) to one another.
- This idealistic definition of a cluster is satisfied only when the data contains natural clusters that are quite far from each other.

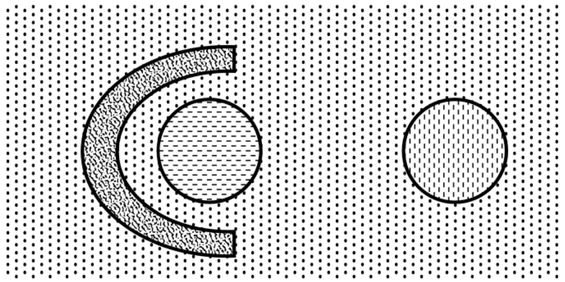
(a) Well-separated clusters. Each point is closer to all of the points in its cluster than to any point in another cluster.



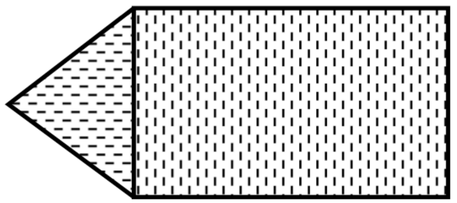
(b) Center-based clusters. Each point is closer to the center of its cluster than to the center of any other cluster.

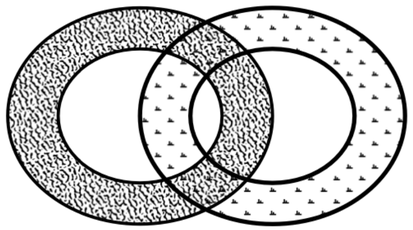


(c) Contiguity-based clusters. Each point is closer to at least one point in its cluster than to any point in another cluster.



(d) Density-based clusters. Clusters are regions of high density separated by regions of low density.





(e) Conceptual clusters. Points in a cluster share some general property that derives from the entire set of points. (Points in the intersection of the circles belong to both.)

Figure 8.2. Different types of clusters as illustrated by sets of two-dimensional points.

Different Types of Clusters

Prototype-Based

- A cluster is a set of objects in which each object is closer (more similar) to the prototype that defines the cluster than to the prototype of any other cluster.
- For data with continuous attributes, the prototype of a cluster is often a centroid, i.e., the average (mean) of all the points in the cluster.
- When a centroid is not meaningful, such as when the data has categorical attributes, the prototype is often a medoid, i.e., the most representative point of a cluster.
- For many types of data, the prototype can be regarded as the most central point, and in such instances, we commonly refer to prototype- based clusters as center-based clusters.

Different Types of Clusters

Graph-Based

- If the data is represented as a graph, where the nodes are objects and the links represent connections among objects, then a cluster can be defined as a connected component; i.e., a group of objects that are connected to one another, but that have no connection to objects outside the group.
- An important example of graph-based clusters are contiguity-based clusters, where two objects are connected only if they are within a specified distance of each other. This implies that each object in a contiguity-based cluster is closer to some other object in the cluster than to any point in a different cluster.
- This can have trouble when noise is present since, as illustrated by the two spherical clusters of Figure 8.2(c), a small bridge of points can merge two distinct clusters.
- Other types of graph-based clusters are also possible. One such approach defines a cluster as a clique; i.e., a set of nodes in a graph that are completely connected to each other.

Different Types of Clusters

Density-Based

- A cluster is a dense region of objects that is surrounded by a region of low density.
- Figure 8.2(d) shows some density-based clusters for data created by adding noise to the data of Figure 8.2(c).
- The two circular clusters are not merged, as in Figure 8.2(c), because the bridge between them fades into the noise. Likewise, the curve that is present in Figure 8.2(c) also fades into the noise and does not form a cluster in Figure 8.2(d).
- A density- based definition of a cluster is often employed when noise and outliers are present.

Different Types of Clusters

Shared-Property (Conceptual Clusters)

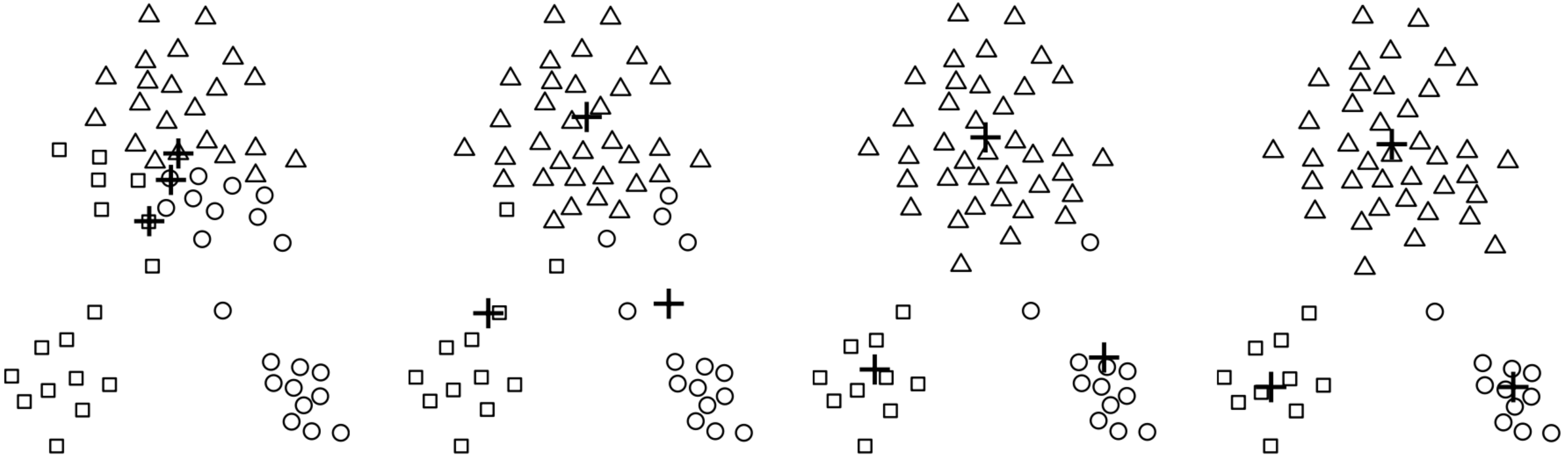
- We can define a cluster as a set of objects that share some property.
- This definition encompasses all the previous definitions of a cluster; e.g., objects in a center-based cluster share the property that they are all closest to the same centroid or medoid.
- However, the shared-property approach also includes new types of clusters. Consider the clusters shown in Figure 8.2(e). A triangular area (cluster) is adjacent to a rectangular one, and there are two intertwined circles (clusters).
- In both cases, a clustering algorithm would need a very specific concept of a cluster to successfully detect these clusters. The process of finding such clusters is called conceptual clustering.

K-means Algorithm

- We first choose K initial centroids, where K is a user- specified parameter, namely, the number of clusters desired.
- Each point is then assigned to the closest centroid, and each collection of points assigned to a centroid is a cluster.
- The centroid of each cluster is then updated based on the points assigned to the cluster.
- We repeat the assignment and update steps until no point changes clusters, or equivalently, until the centroids remain the same.

Algorithm 8.1 Basic K-means algorithm.

1: Select K points as initial centroids.
2: **repeat**
3: Form K clusters by assigning each point to its closest centroid.
4: Recompute the centroid of each cluster.
5: **until** Centroids do not change.



(a) Iteration 1. (b) Iteration 2. (c) Iteration 3. (d) Iteration 4.

Figure 8.3. Using the K-means algorithm to find three clusters in sample data.

- To assign a point to the closest centroid, we need a proximity measure that quantifies the notion of “closest” for the specific data under consideration.
- **Euclidean (L2) distance** is often used for data points in Euclidean space, while cosine similarity is more appropriate for documents.
- For our objective function, which measures the quality of a clustering, we use the **sum of the squared error (SSE)**, which is also known as scatter.
- In other words, we calculate the error of each data point, i.e., its Euclidean distance to the closest centroid, and then compute the total sum of the squared errors.
- Given two different sets of clusters that are produced by two different runs of K-means, we prefer the one with the smallest squared error since this means that the prototypes (centroids) of this clustering are a better representation of the points in their cluster.
- The centroid that minimizes the SSE of the cluster is the mean.

$$SSE = \sum_{i=1}^{K} \sum_{x \in C_i} dist(c_i, x)^2$$

where dist is the standard Euclidean (L2) distance between two objects in Euclidean space.

The centroid (mean) of the i-th cluster is defined by $c_i = \frac{1}{m_i} \sum_{x \in C_i} x$

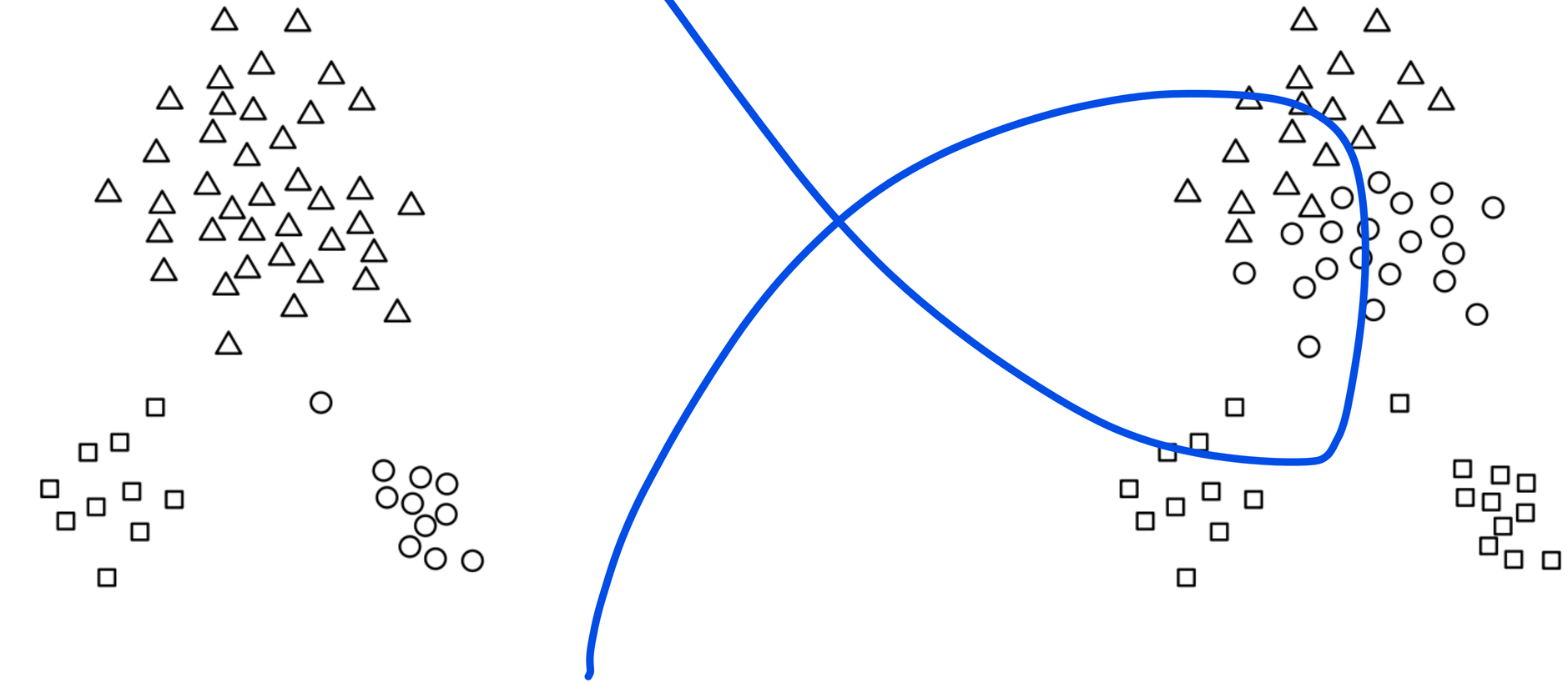
To illustrate, the centroid of a cluster containing the three two-dimensional points, (1,1), (2,3), and (6,2), is ((1 + 2 + 6)/3, ((1 + 3 + 2)/3) = (3, 2).

Table 8.1. Table of notation.

Symbol	Description
$\mathbf{x}$	An object.
C_i	The i^{th} cluster.
$\mathbf{c}_i$	The centroid of cluster C_i.
$\mathbf{c}$	The centroid of all points.
m_i	The number of objects in the i^{th} cluster.
m	The number of objects in the data set.
K	The number of clusters.

- **Document Data** Here we assume that the document data is represented as a document-term matrix.
- Our objective is to maximize the similarity of the documents in a cluster to the cluster centroid; this quantity is known as the cohesion of the cluster.
- For this objective it can be shown that the cluster centroid is, as for Euclidean data, the mean. The analogous quantity to the total SSE is the total cohesion
- $$Total\ Cohesion = \sum_{i=1}^{K} \sum_{x \in C_i} cosine(x, c_i)$$

- When random initialization of centroids is used, different runs of K-means typically produce different total SSEs.

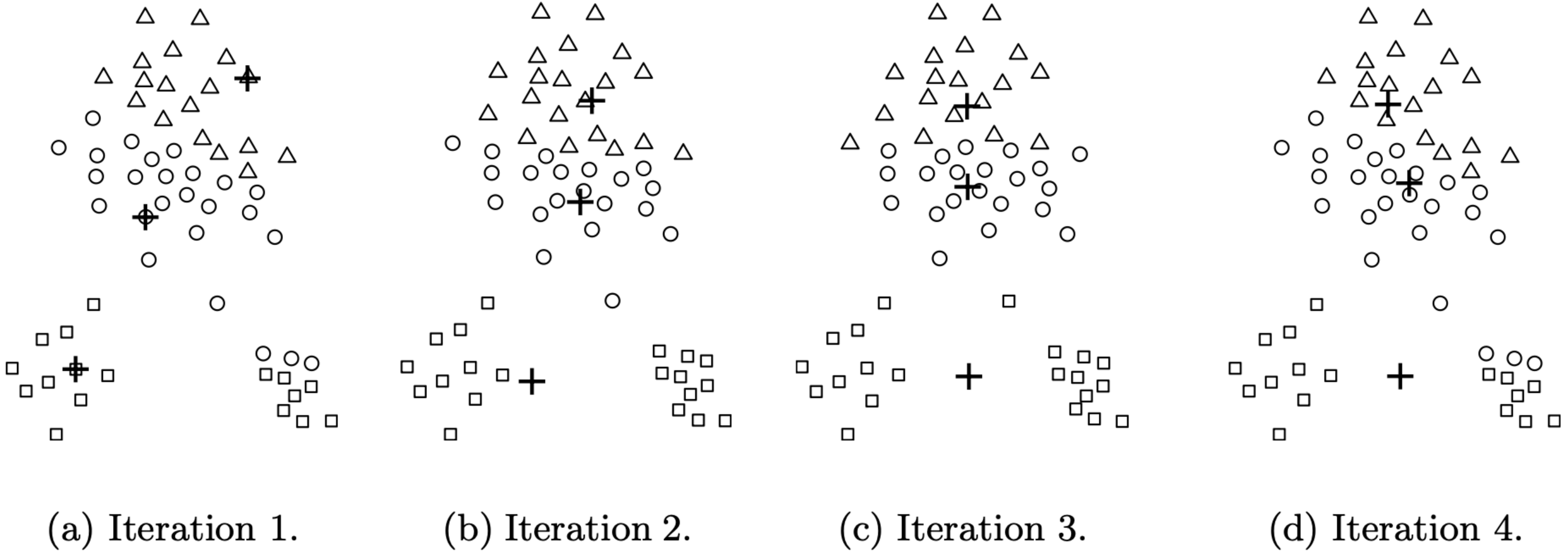


(a) Optimal clustering.

(b) Suboptimal clustering.

Figure 8.4. Three optimal and non-optimal clusters.

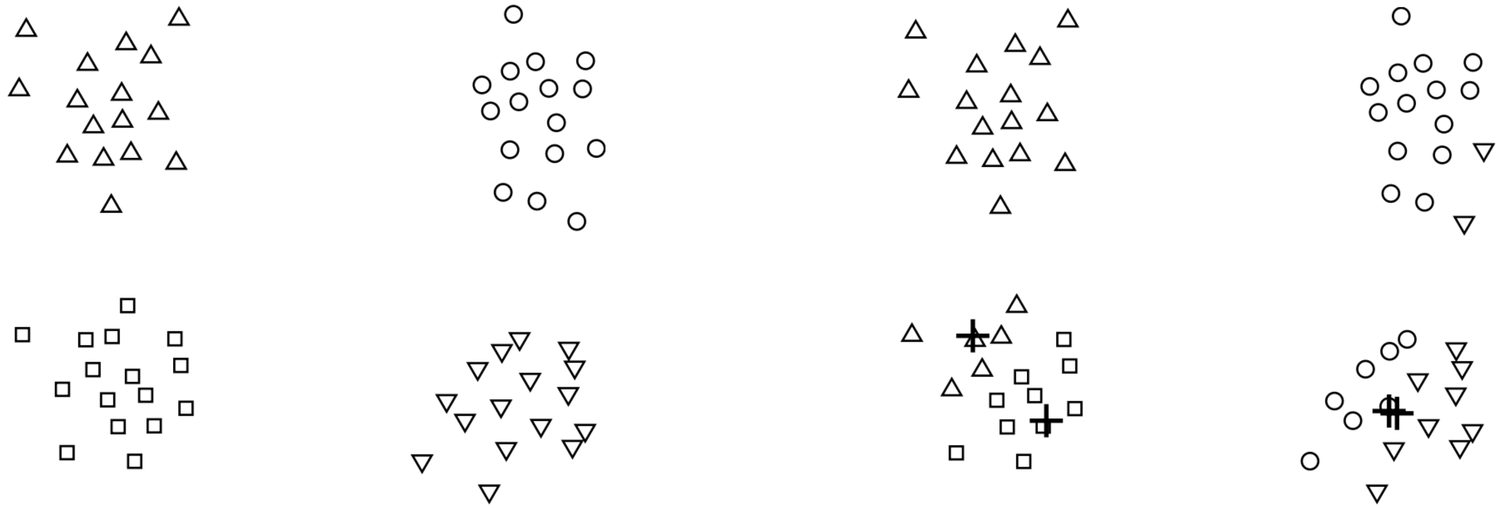
- **Poor Initial Centroid**s. Randomly selected initial centroids may be poor. Even though the initial centroids seem to be better distributed, we obtain a suboptimal clustering, with higher squared error.



(a) Iteration 1. (b) Iteration 2. (c) Iteration 3. (d) Iteration 4.

Figure 8.5. Poor starting centroids for K-means.

- One technique that is commonly used to address the problem of choosing initial centroids is to perform multiple runs, each with a different set of randomly chosen initial centroids, and then select the set of clusters with the minimum SSE.
- While simple, this strategy may not work very well, depending on the data set and the number of clusters sought.
- In Figure 8.6(a), the data consists of two pairs of clusters, where the clusters in each (top-bottom) pair are closer to each other than to the clusters in the other pair.
- Figure 8.6 (b–d) shows that if we start with two initial centroids per pair of clusters, then even when both centroids are in a single cluster, the centroids will redistribute themselves so that the “true” clusters are found.
- However, Figure 8.7 shows that if a pair of clusters has only one initial centroid and the other pair has three, then two of the true clusters will be combined and one true cluster will be split.



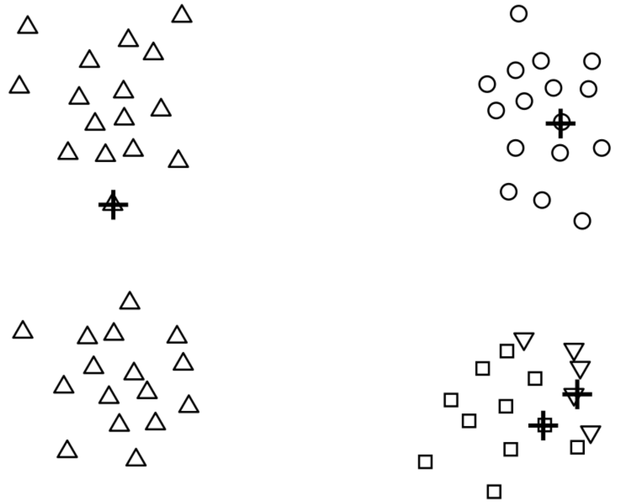
(a) Initial points.

(b) Iteration 1.

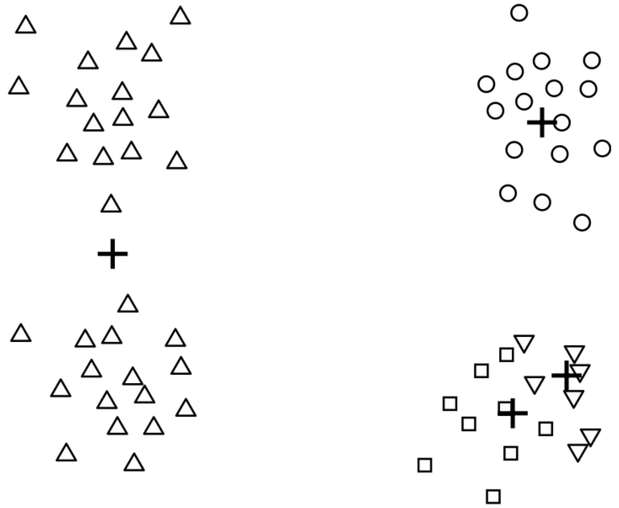
(c) Iteration 2.

(d) Iteration 3.

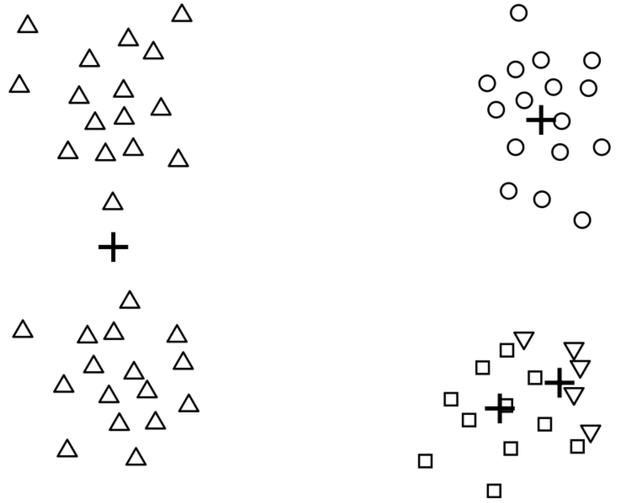
Figure 8.6. Two pairs of clusters with a pair of initial centroids within each pair of clusters.



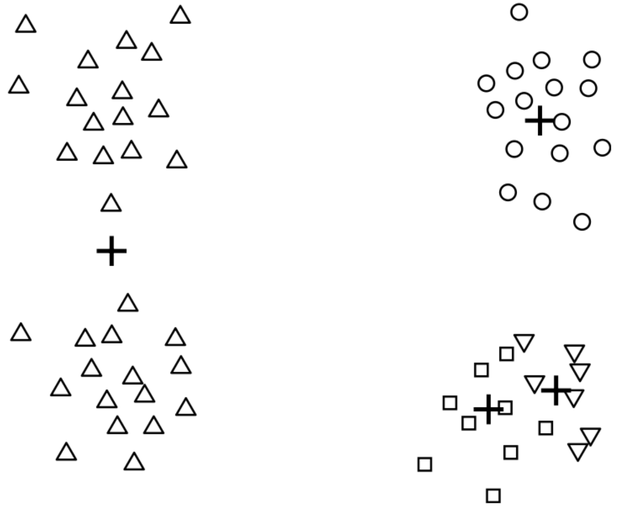
(a) Iteration 1.



(b) Iteration 2.



(c) Iteration 3.



(d) Iteration 4.

Figure 8.7. Two pairs of clusters with more or fewer than two initial centroids within a pair of clusters.

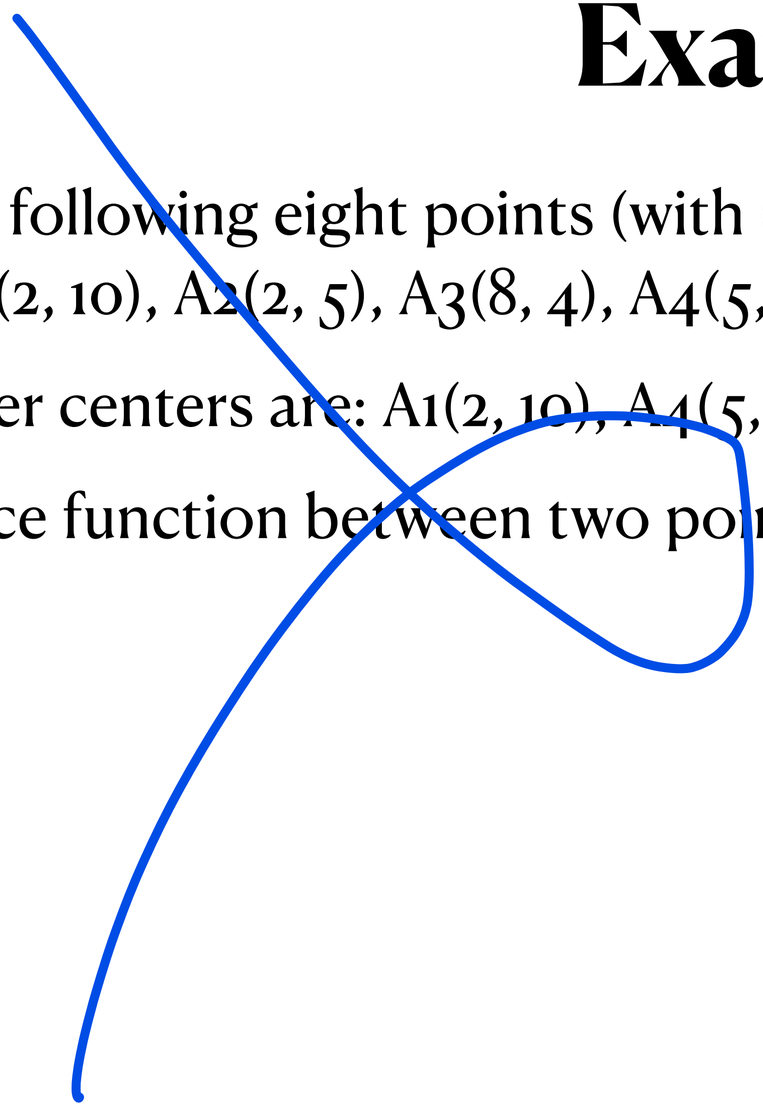
- One effective approach is to take a sample of points and cluster them using a hierarchical clustering technique.
- K clusters are extracted from the hierarchical clustering, and the centroids of those clusters are used as the initial centroids.
- This approach often works well, but is practical only if (1) the sample is relatively small, e.g., a few hundred to a few thousand (hierarchical clustering is expensive), and (2) K is relatively small compared to the sample size.
- **Another approach.** Select the first point at random or take the centroid of all points. Then, for each successive initial centroid, select the point that is farthest from any of the initial centroids already selected.
- In this way, we obtain a set of initial centroids that is guaranteed to be not only randomly selected but also well separated.
- Unfortunately, such an approach can select outliers, rather than points in dense regions (clusters). Also, it is expensive to compute the farthest point from the current set of initial centroids.
- To overcome these problems, this approach is often applied to a sample of the points. Since outliers are rare, they tend not to show up in a random sample. In contrast, points from every dense region are likely to be included unless the sample size is very small.
- Also, the computation involved in finding the initial centroids is greatly reduced because the sample size is typically much smaller than the number of points.

Time and Space Complexity

- The space requirements for K-means are modest because only the data points and centroids are stored.
- Specifically, the storage required is $O((m + K)n)$, where m is the number of points and n is the number of attributes.
- The time requirements for K-means are also modest—basically linear in the number of data points.
- In particular, the time required is $O(I * K * m * n)$, where I is the number of iterations required for convergence.

Example

- Cluster the following eight points (with (x, y) representing locations) into three clusters: A1(2, 10), A2(2, 5), A3(8, 4), A4(5, 8), A5(7, 5), A6(6, 4), A7(1, 2), A8(4, 9)
- Initial cluster centers are: A1(2, 10), A4(5, 8) and A7(1, 2)
- The distance function between two points is Euclidean.



K-Means Issues

- **Handling Empty Clusters** Empty clusters can be obtained if no points are allocated to a cluster during the assignment step. If this happens, then a strategy is needed to choose a replacement centroid, since otherwise, the squared error will be larger than necessary.
- One approach is to choose the point that is farthest away from any current centroid. If nothing else, this eliminates the point that currently contributes most to the total squared error.
- Another approach is to choose the replacement centroid from the cluster that has the highest SSE. This will typically split the cluster and reduce the overall SSE of the clustering.
- If there are several empty clusters, then this process can be repeated several times.

K-Means Issues

- **Outliers** When the squared error criterion is used, outliers can unduly influence the clusters that are found.
- In particular, when outliers are present, the resulting cluster centroids (prototypes) may not be as representative as they otherwise would be and thus, the SSE will be higher as well.
- Because of this, it is often useful to discover outliers and eliminate them beforehand.
- It is important, however, to appreciate that there are certain clustering applications for which outliers should not be eliminated.
- When clustering is used for data compression, every point must be clustered, and in some cases, such as financial analysis, apparent outliers, e.g., unusually profitable customers, can be the most interesting points.
- Outliers can also be identified in a postprocessing step. For instance, we can keep track of the SSE contributed by each point, and eliminate those points with unusually high contributions, especially over multiple runs. Also, we may want to eliminate small clusters since they frequently represent groups of outliers.

K-Means Issues

- **Reducing the SSE with Postprocessing** An obvious way to reduce the SSE is to find more clusters, i.e., to use a larger K.
- Various techniques are used to “fix up” the resulting clusters in order to produce a clustering that has lower SSE.
- The strategy is to focus on individual clusters since the total SSE is simply the sum of the SSE contributed by each cluster. (We will use the terminology total SSE and cluster SSE, respectively, to avoid any potential confusion.)
- Two strategies that decrease the total SSE by increasing the number of clusters are the following:

1. Split a cluster: The cluster with the largest SSE is usually chosen, but we could also split the cluster with the largest standard deviation for one particular attribute.
2. Introduce a new cluster centroid: Often the point that is farthest from any cluster center is chosen. We can easily determine this if we keep track of the SSE contributed by each point. Another approach is to choose randomly from all points or from the points with the highest SSE.

K-Means Issues

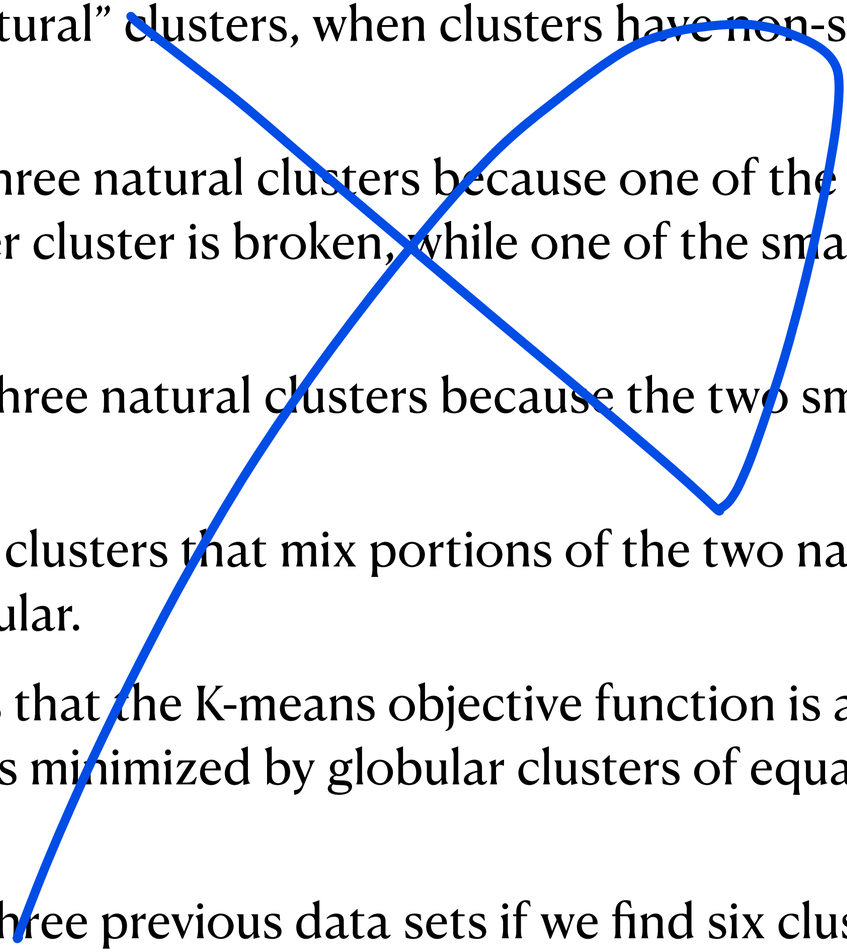
- Two strategies that decrease the total SSE by decreasing the number of clusters are the following:

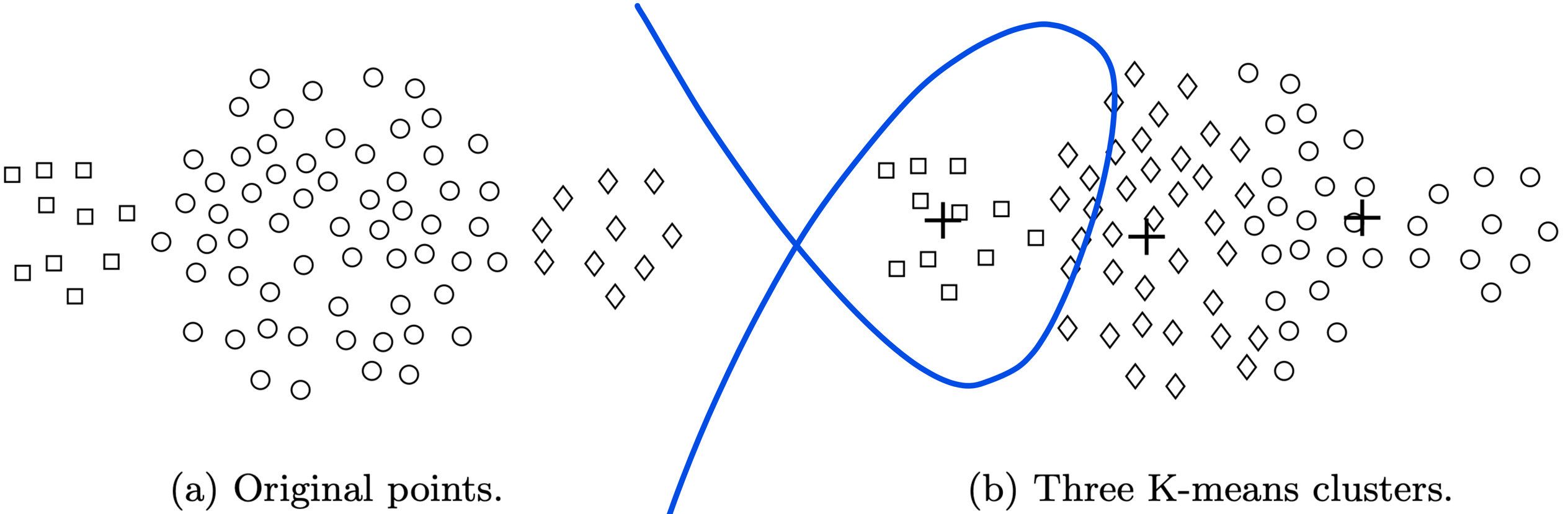
1. Disperse a cluster: This is accomplished by removing the centroid that corresponds to the cluster and reassigning the points to other clusters. Ideally, the cluster that is dispersed should be the one that increases the total SSE the least.

2. Merge two clusters: The clusters with the closest centroids are typically chosen, although another, perhaps better, approach is to merge the two clusters that result in the smallest increase in total SSE.

K-means and Different Types of Clusters

- K-means has difficulty detecting the “natural” clusters, when clusters have non-spherical shapes or widely different sizes or densities.
- In Figure 8.9, K-means cannot find the three natural clusters because one of the clusters is much larger than the other two, and hence, the larger cluster is broken, while one of the smaller clusters is combined with a portion of the larger cluster.
- In Figure 8.10, K-means fails to find the three natural clusters because the two smaller clusters are much denser than the larger cluster.
- Finally, in Figure 8.11, K-means finds two clusters that mix portions of the two natural clusters because the shape of the natural clusters is not globular.
- The difficulty in these three situations is that the K-means objective function is a mismatch for the kinds of clusters we are trying to find since it is minimized by globular clusters of equal size and density or by clusters that are well separated.
- Figure 8.12 shows what happens to the three previous data sets if we find six clusters instead of two or three. Each smaller cluster is pure in the sense that it contains only points from one of the natural clusters.

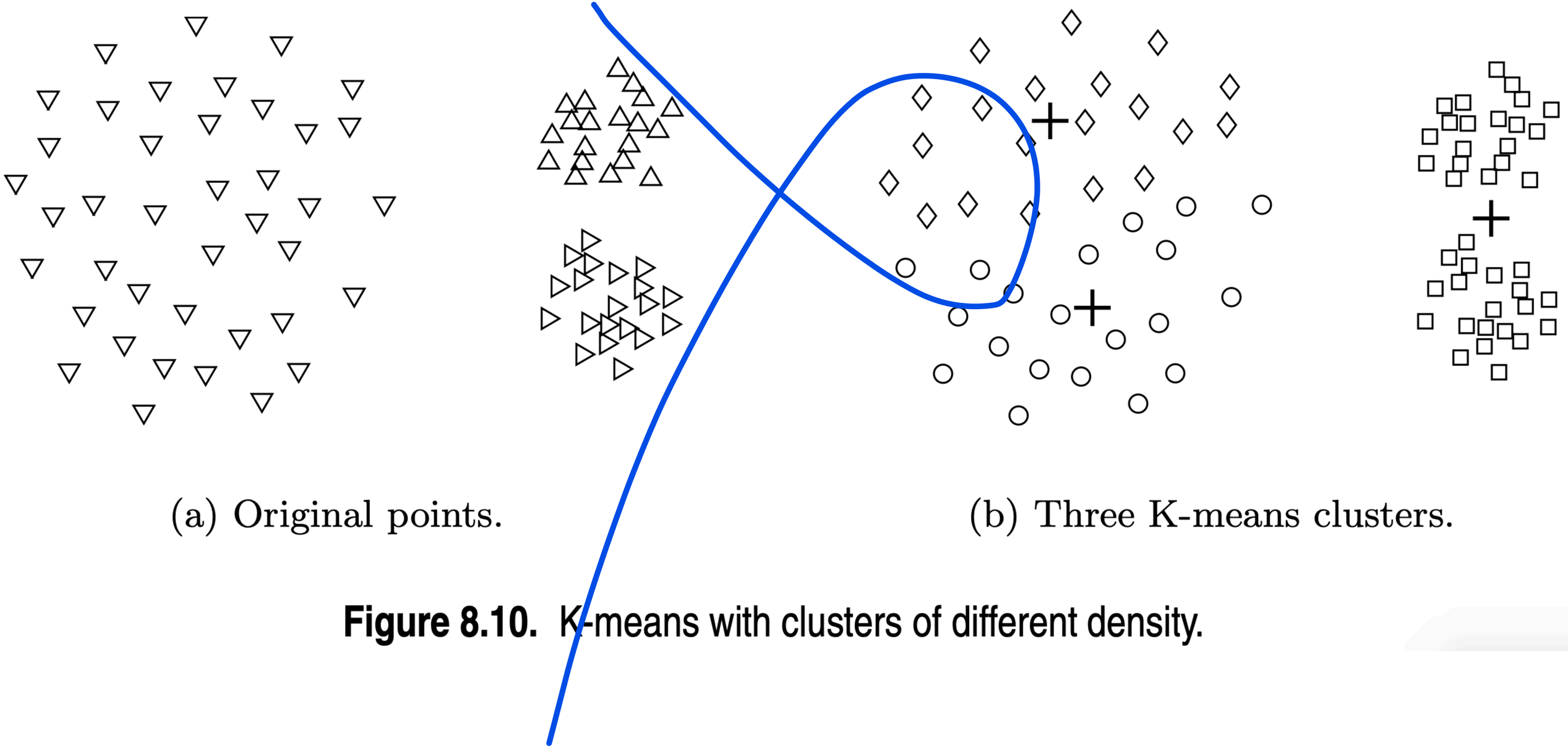




(a) Original points.

(b) Three K-means clusters.

Figure 8.9. K-means with clusters of different size.



(a) Original points.

(b) Three K-means clusters.

Figure 8.10. K-means with clusters of different density.

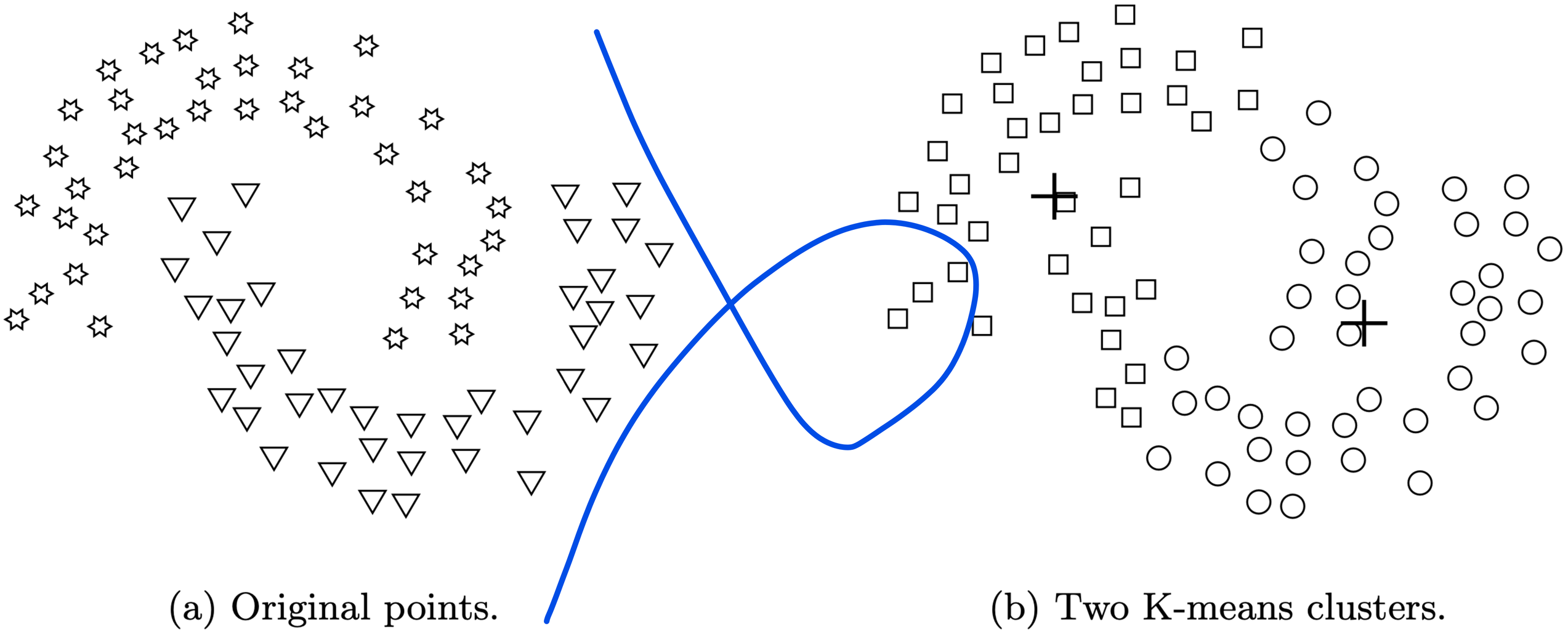
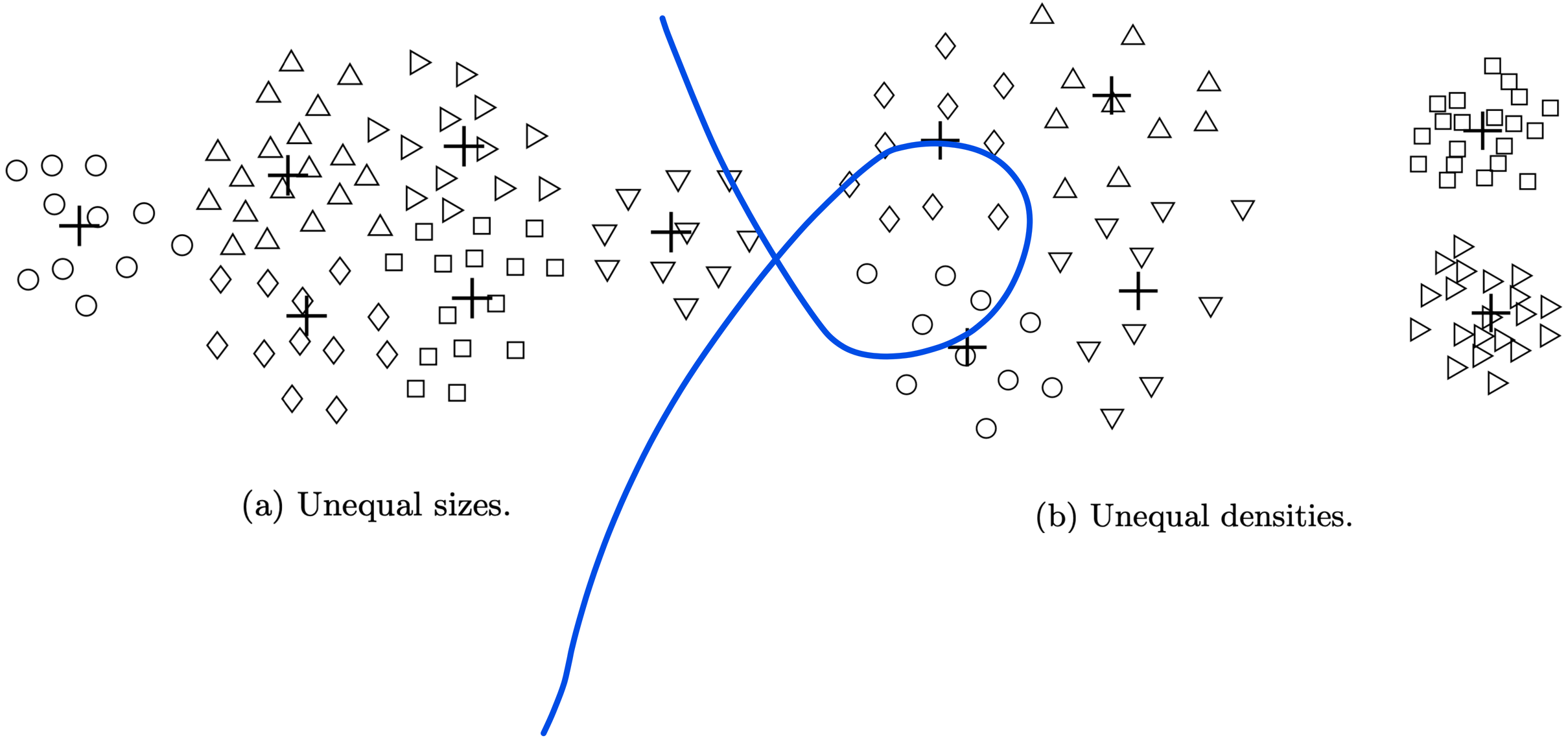
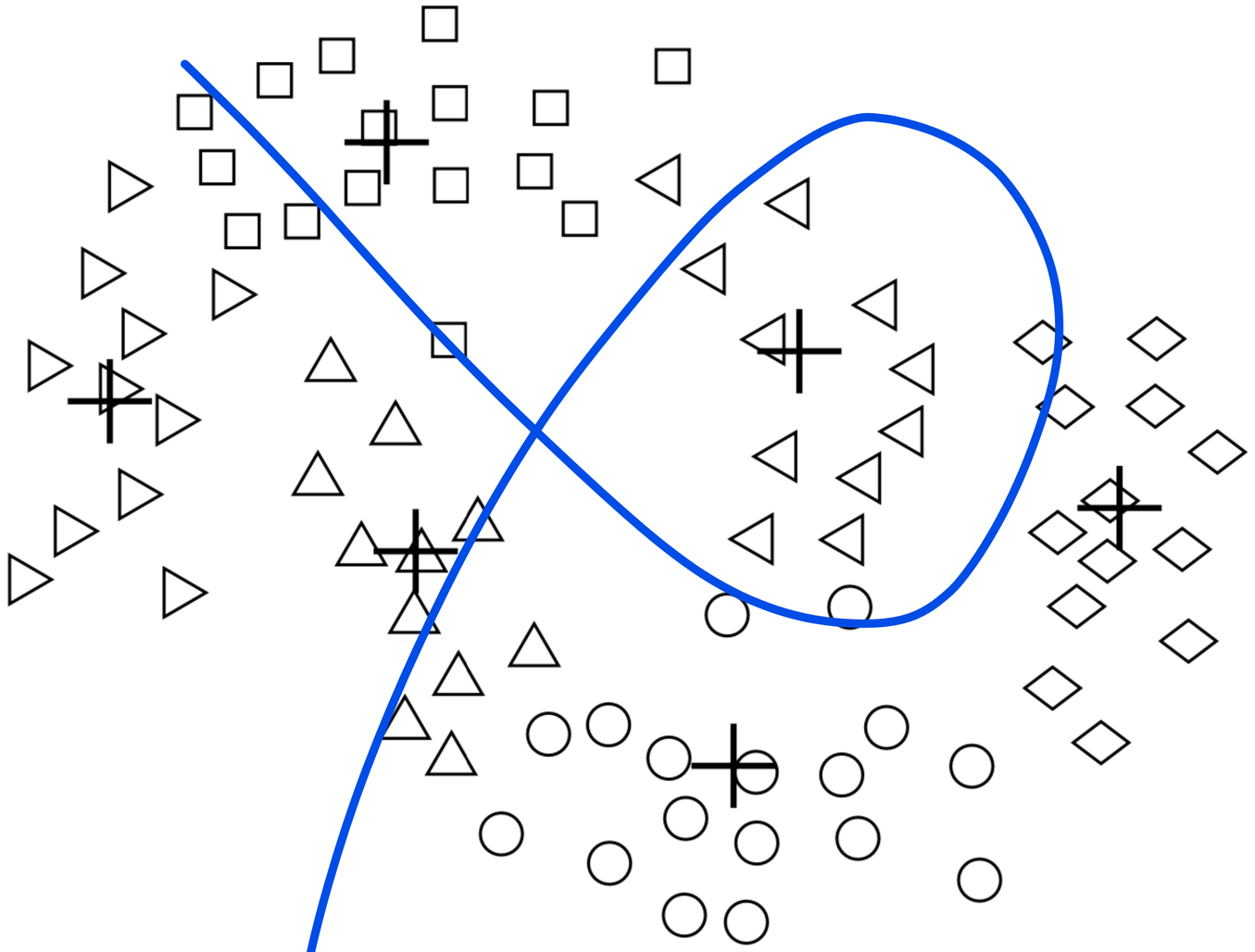


Figure 8.11. K-means with non-globular clusters.



(a) Unequal sizes.

(b) Unequal densities.



(c) Non-spherical shapes.

Figure 8.12. Using K-means to find clusters that are subclusters of the natural clusters.

Strengths and Weaknesses

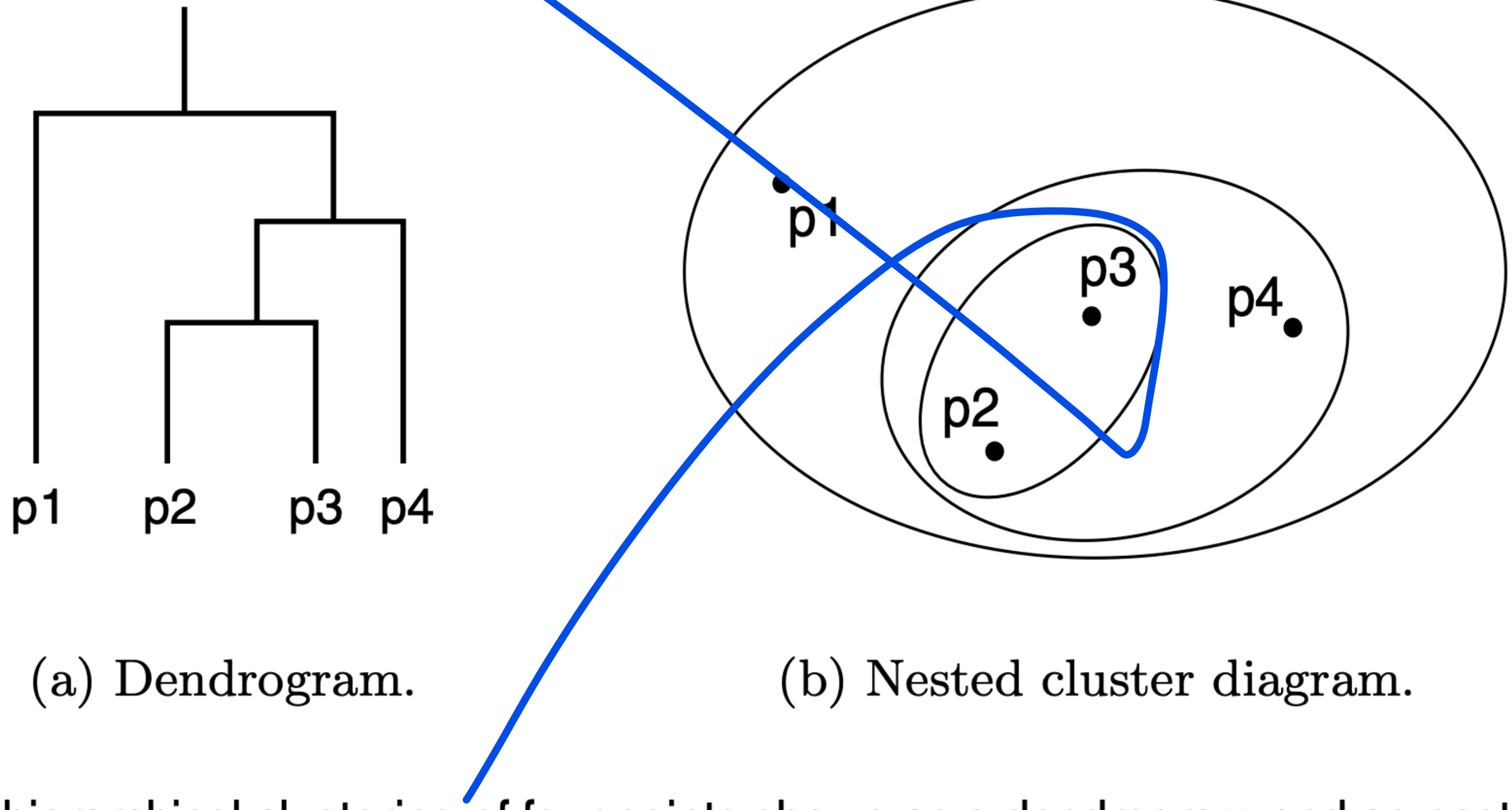
- K-means is simple and can be used for a wide variety of data types. It is also quite efficient, even though multiple runs are often performed.
- K-means is not suitable for all types of data, however. It cannot handle non-globular clusters or clusters of different sizes and densities, although it can typically find pure subclusters if a large enough number of clusters is specified.
- K-means also has trouble clustering data that contains outliers. Outlier detection and removal can help significantly in such situations.
- Finally, K-means is restricted to data for which there is a notion of a center (centroid).

Hierarchical Clustering

- There are two basic approaches for generating a hierarchical clustering:

1. **Agglomerative**: Start with the points as individual clusters and, at each step, merge the closest pair of clusters. This requires defining a notion of cluster proximity.
2. **Divisive**: Start with one, all-inclusive cluster and, at each step, split a cluster until only singleton clusters of individual points remain. In this case, we need to decide which cluster to split at each step and how to do the splitting.

- A hierarchical clustering is often displayed graphically using a tree-like diagram called a **dendrogram**, which displays both the cluster-subcluster relationships and the order in which the clusters were merged (agglomerative view) or split (divisive view).
- For sets of two-dimensional points, a hierarchical clustering can also be graphically represented using a nested cluster diagram.



(a) Dendrogram.

(b) Nested cluster diagram.

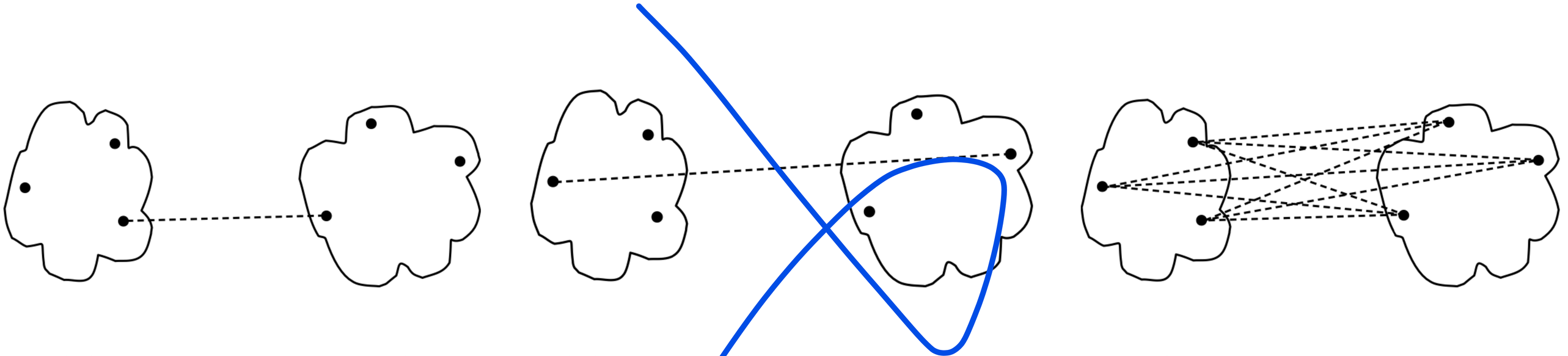
Figure 8.13. A hierarchical clustering of four points shown as a dendrogram and as nested clusters.

Agglomerative Hierarchical Clustering

Algorithm 8.3 Basic agglomerative hierarchical clustering algorithm.

1: Compute the proximity matrix, if necessary.
2: **repeat**
3: Merge the closest two clusters.
4: Update the proximity matrix to reflect the proximity between the new cluster and the original clusters.
5: **until** Only one cluster remains.

- The definition of cluster proximity differentiates the various agglomerative hierarchical techniques.
- **MIN** or **single link** defines cluster proximity as the proximity between the closest two points that are in different clusters, or using graph terms, the shortest edge between two nodes in different subsets of nodes. This yields contiguity-based clusters.
- Alternatively, **MAX** or **complete link** takes the proximity between the farthest two points in different clusters to be the cluster proximity, or using graph terms, the longest edge between two nodes in different subsets of nodes.
- Another graph-based approach, the **group average** technique, defines cluster proximity to be the average pairwise proximities (average length of edges) of all pairs of points from different clusters.
- If, instead, we take a prototype-based view, in which each cluster is represented by a centroid, the cluster proximity is commonly defined as the proximity between cluster centroids.



(a) MIN (single link.) (b) MAX (complete link.) (c) Group average.

Figure 8.14. Graph-based definitions of cluster proximity

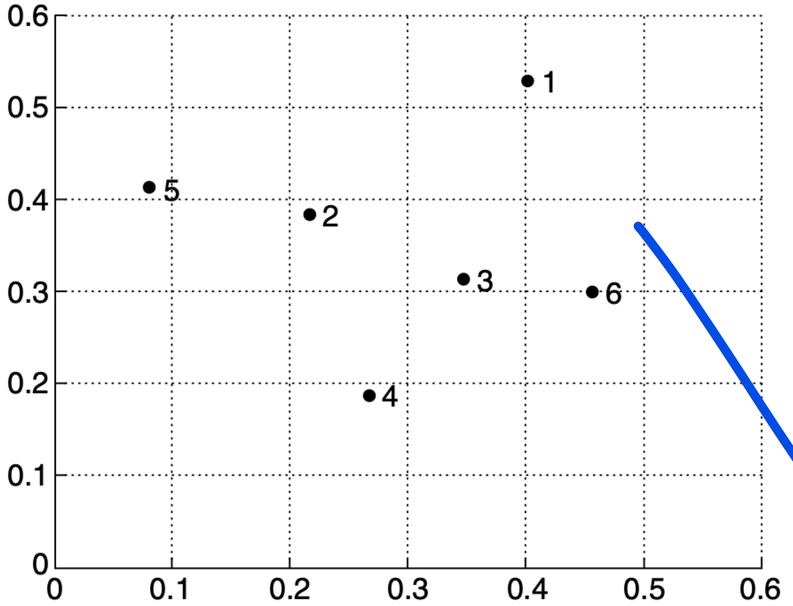


Figure 8.15. Set of 6 two-dimensional points.

Point	x Coordinate	y Coordinate
p1	0.40	0.53
p2	0.22	0.38
p3	0.35	0.32
p4	0.26	0.19
p5	0.08	0.41
p6	0.45	0.30

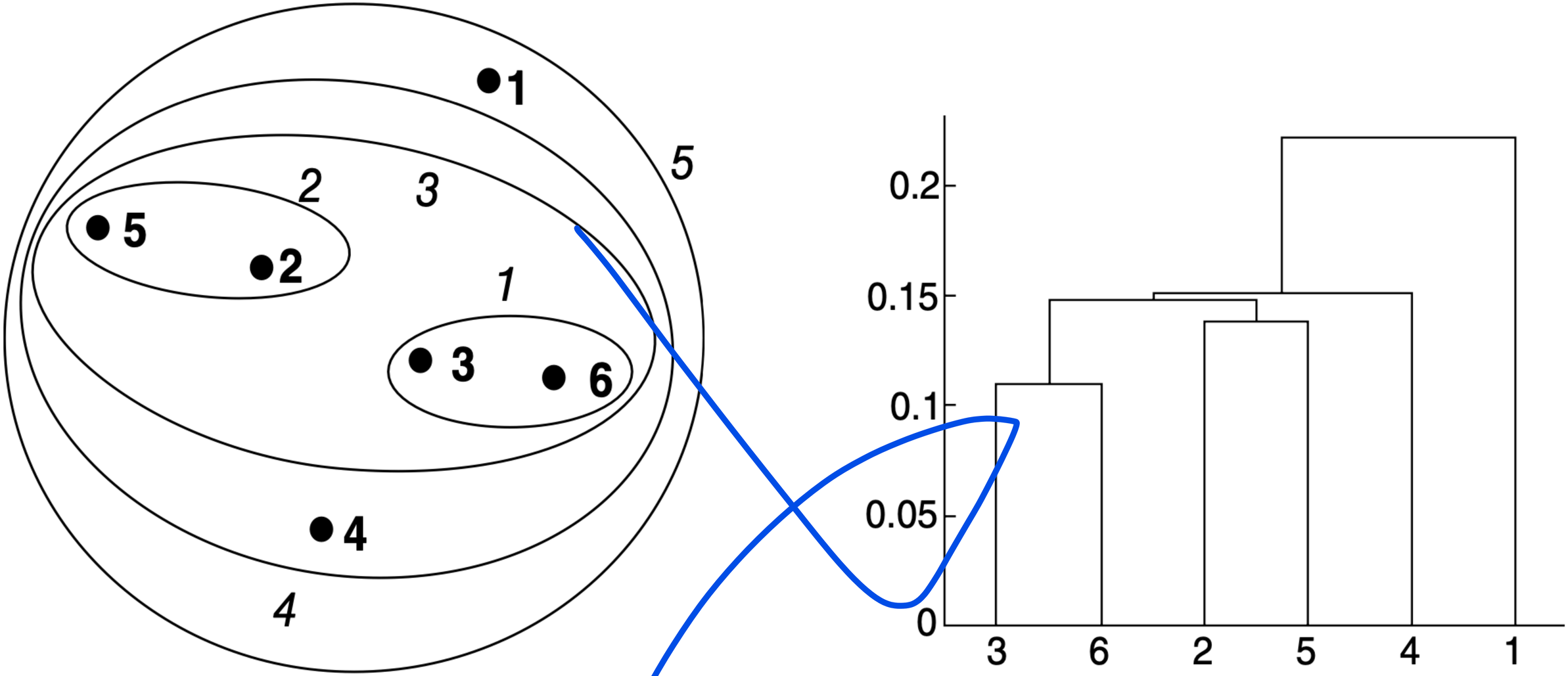
Table 8.3. xy coordinates of 6 points.

	p1	p2	p3	p4	p5	p6
p1	0.00	0.24	0.22	0.37	0.34	0.23
p2	0.24	0.00	0.15	0.20	0.14	0.25
p3	0.22	0.15	0.00	0.15	0.28	0.11
p4	0.37	0.20	0.15	0.00	0.29	0.22
p5	0.34	0.14	0.28	0.29	0.00	0.39
p6	0.23	0.25	0.11	0.22	0.39	0.00

Table 8.4. Euclidean distance matrix for 6 points.

Single Link

- The proximity of two clusters is defined as the minimum of the distance (maximum of the similarity) between any two points in the two different clusters.
- Using graph terminology, if you start with all points as singleton clusters and add links between points one at a time, shortest links first, then these single links com- bine the points into clusters.
- The height at which two clusters are merged in the dendrogram reflects the distance of the two clusters.
- From Table 8.4, we see that the distance between points 3 and 6 is 0.11, and that is the height at which they are joined into one cluster in the dendrogram.
- As another example, the distance between clusters {3,6} and {2, 5} is given by
- dist({3, 6}, {2, 5}) = min(dist(3, 2), dist(6, 2), dist(3, 5), dist(6, 5)) = min(0.15, 0.25, 0.28, 0.39) = 0.15.



(a) Single link clustering.

(b) Single link dendrogram.

Figure 8.16. Single link clustering of the six points shown in Figure 8.15.

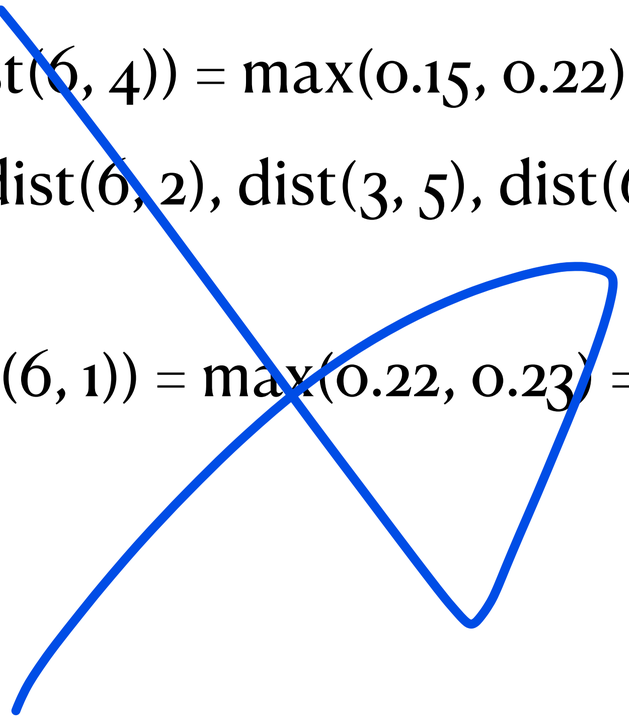
Complete Link

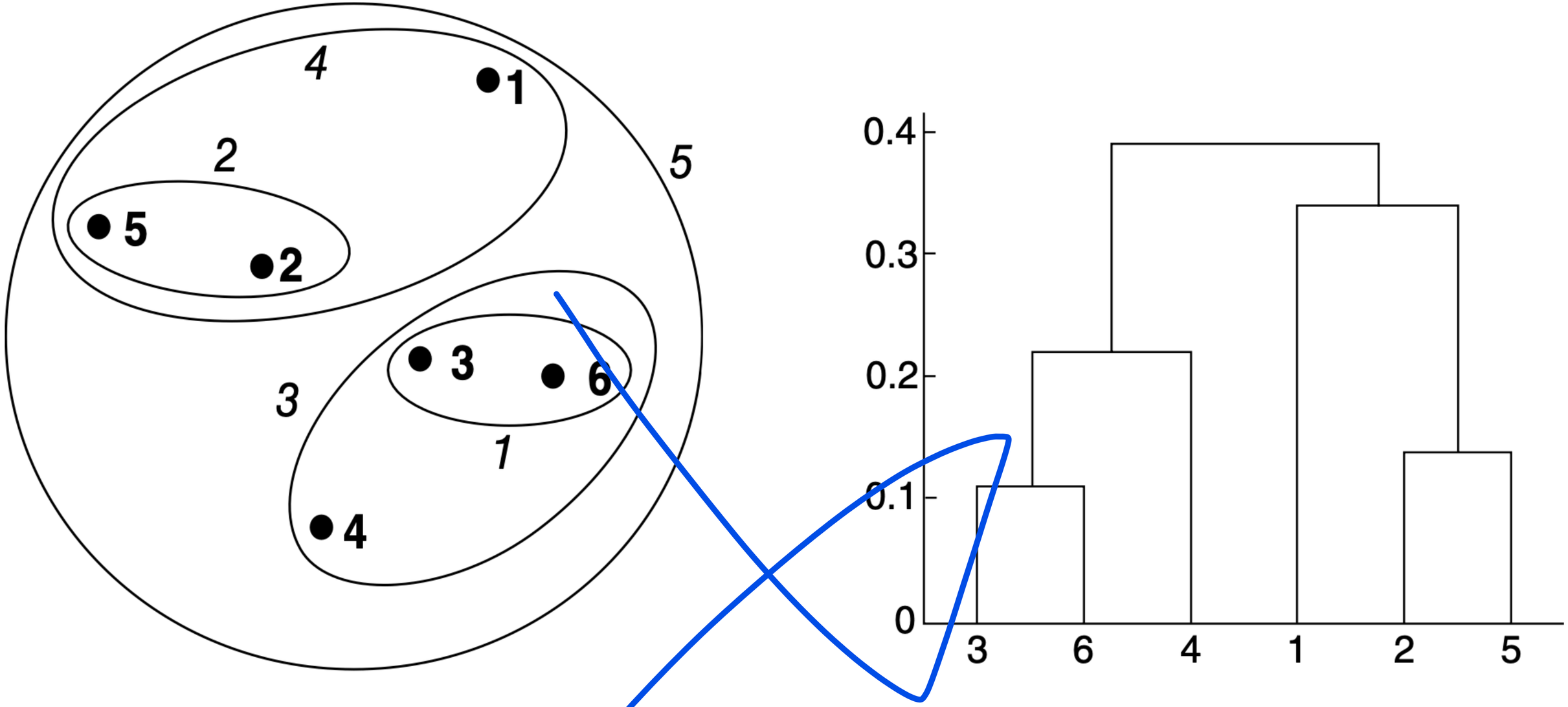
- As with single link, points 3 and 6 are merged first.
- However, {3, 6} is merged with {4}, instead of {2, 5} or {1} because

dist({3, 6}, {4}) = max(dist(3, 4), dist(6, 4)) = max(0.15, 0.22) = 0.22.

dist({3, 6}, {2, 5}) = max(dist(3, 2), dist(6, 2), dist(3, 5), dist(6, 5)) = max(0.15, 0.25, 0.28, 0.39) = 0.39.

dist({3, 6}, {1}) = max(dist(3, 1), dist(6, 1)) = max(0.22, 0.23) = 0.23.





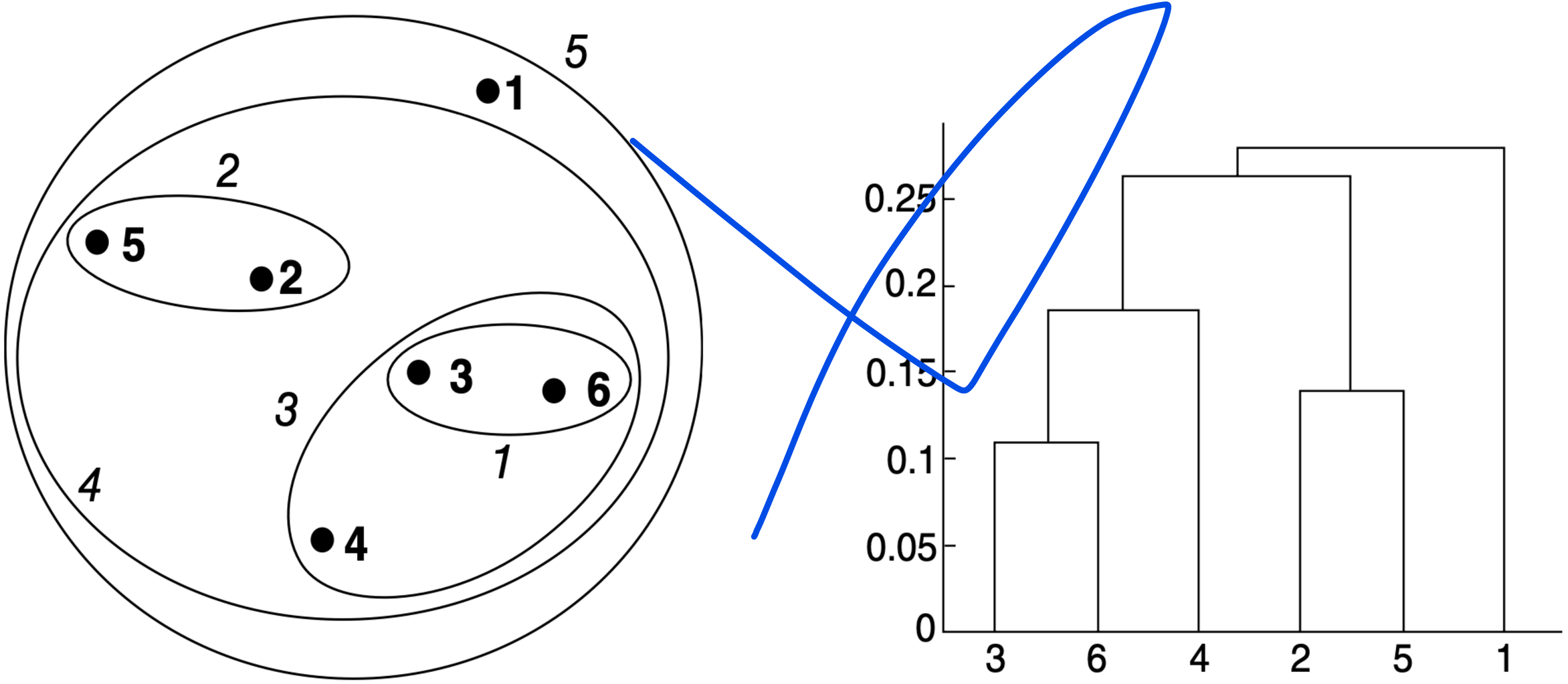
(a) Complete link clustering.

(b) Complete link dendrogram.

Figure 8.17. Complete link clustering of the six points shown in Figure 8.15.

Group Average

- The proximity of two clusters is defined as the average pairwise proximity among all pairs of points in the different clusters.
- The cluster proximity $proximity(C_i, C_j)$ of clusters C_i and C_j, which are of size m_i and m_j respectively, is expressed by the following equation:
- $$proximity(C_i, C_j) = \frac{\sum_{x \in C_i, y \in C_j} proximity(x, y)}{m_i * m_j}$$
- dist({3, 6, 4}, {1}) = (0.22 + 0.37 + 0.23)/(3 * 1) = 0.28
- dist({2, 5}, {1}) = (0.2357 + 0.3421)/(2 * 1) = 0.2889
- dist({3, 6, 4}, {2, 5}) = (0.15 + 0.28 + 0.25 + 0.39 + 0.20 + 0.29)/(3 * 2) = 0.26
- Because dist({3, 6, 4}, {2, 5}) is smaller than dist({3, 6, 4}, {1}) and dist({2, 5}, {1}), clusters {3, 6, 4} and {2, 5} are merged at the fourth stage.



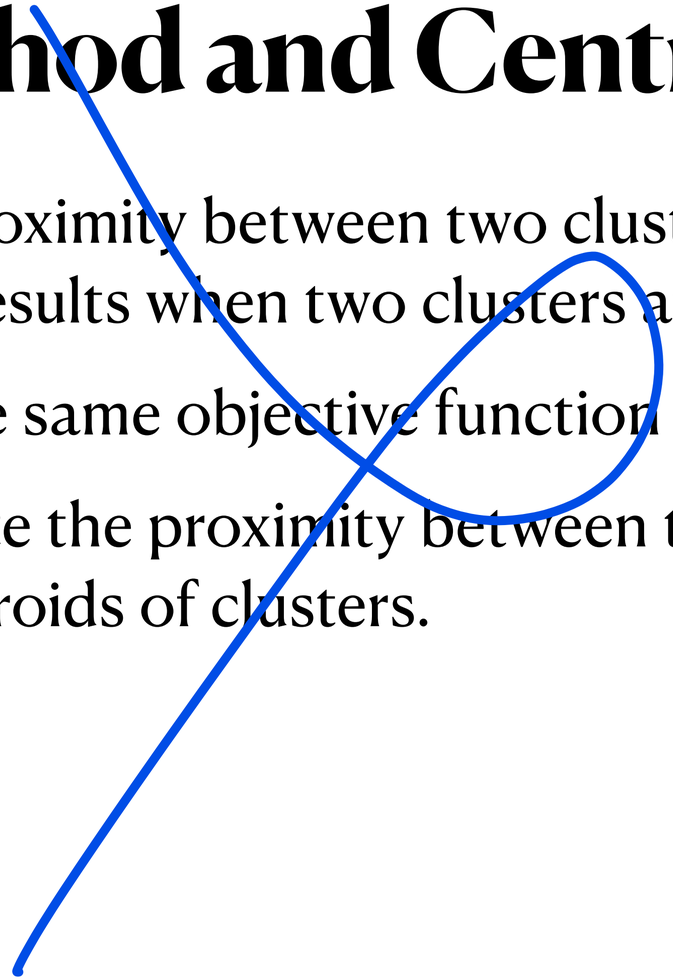
(a) Group average clustering.

(b) Group average dendrogram.

Figure 8.18. Group average clustering of the six points shown in Figure 8.15.

Ward's Method and Centroid Methods

- For Ward's method, the proximity between two clusters is defined as the in- crease in the squared error that results when two clusters are merged.
- Thus, this method uses the same objective function as K-means clustering.
- Centroid methods calculate the proximity between two clusters by calcu- lating the distance between the centroids of clusters.



DBSCAN

- Density-based clustering locates regions of high density that are separated from one another by regions of low density.
- In the center-based approach, density is estimated for a particular point in the data set by counting the number of points within a specified radius, Eps, of that point. The number of points within a radius of Eps of point A is 7, including A itself.
- The density of any point will depend on the specified radius.
- If the radius is large enough, then all points will have a density of m, the number of points in the data set.
- Likewise, if the radius is too small, then all points will have a density of 1.

DBSCAN

- The center-based approach to density allows us to classify a point as being (1) in the interior of a dense region (a **core** point), (2) on the edge of a dense region (a **border** point), or (3) in a sparsely occupied region (a **noise** or **background** point).
- **Core points:** These points are in the interior of a density-based cluster. A point is a core point if the number of points within a given neighborhood around the point as determined by the distance function and a user- specified distance parameter, Eps, exceeds a certain threshold, **MinPts**, which is also a user-specified parameter. In Figure 8.21, point A is a core point, for the indicated radius (Eps) if MinPts ≤ 7.
- **Border points:** A border point is not a core point, but falls within the neighborhood of a core point. In Figure 8.21, point B is a border point. A border point can fall within the neighborhoods of several core points.
- **Noise points:** A noise point is any point that is neither a core point nor a border point. In Figure 8.21, point C is a noise point.

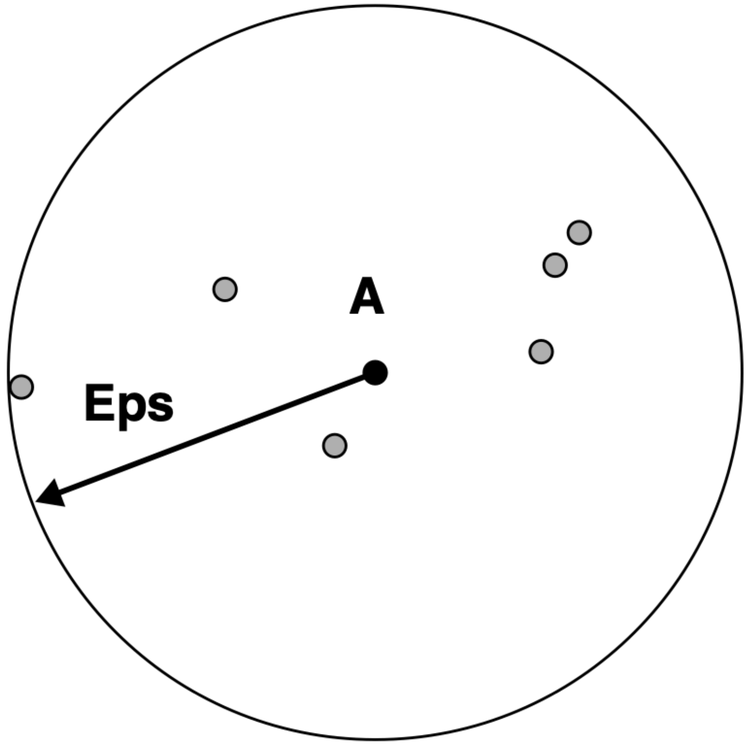


Figure 8.20. Center-based density.

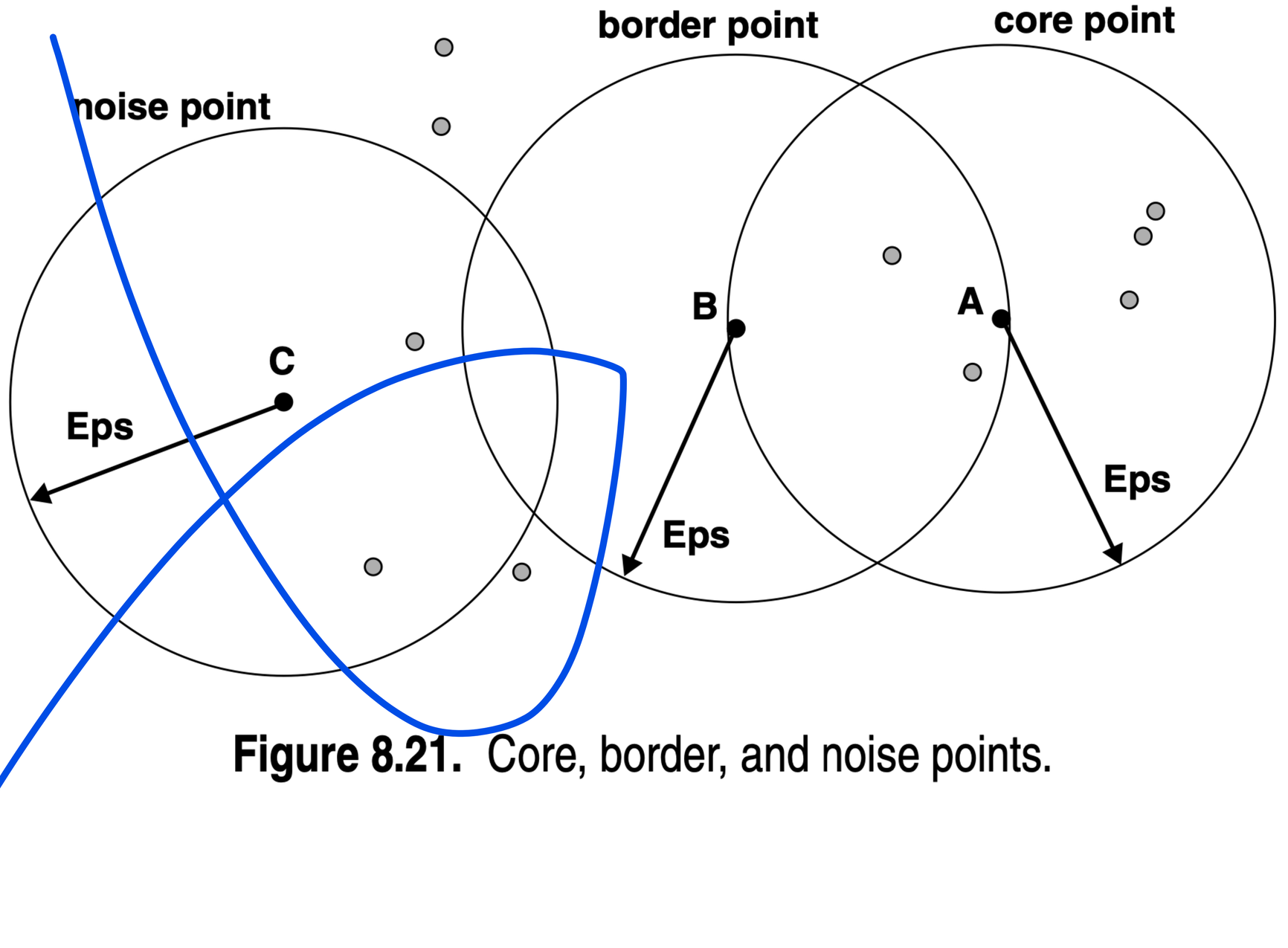


Figure 8.21. Core, border, and noise points.

- Any two core points that are close enough—within a distance Eps of one another—are put in the same cluster.
- Likewise, any border point that is close enough to a core point is put in the same cluster as the core point. (Ties may need to be resolved if a border point is close to core points from different clusters.)
- Noise points are discarded.

Algorithm 8.4 DBSCAN algorithm.

1: Label all points as core, border, or noise points.
2: Eliminate noise points.
3: Put an edge between all core points that are within *Eps* of each other.
4: Make each group of connected core points into a separate cluster.
5: Assign each border point to one of the clusters of its associated core points.

- The basic time complexity of the DBSCAN algorithm is O(m × time to find points in the Eps-neighborhood), where m is the number of points.
- In the worst case, this complexity is $O(m^2)$.
- The space requirement of DBSCAN, even for high-dimensional data, is O(m) because it is only necessary to keep a small amount of data for each point, i.e., the cluster label and the identification of each point as a core, border, or noise point.
- The distance from a point to its k-th nearest neighbor, which we will call the k-dist. If we compute the k-dist for all the data points for some k, sort them in increasing order, and then plot the sorted values, we expect to see a sharp change at the value of k-dist that corresponds to a suitable value of Eps.
- If we select this distance as the Eps parameter and take the value of k as the MinPts parameter, then points for which k-dist is less than Eps will be labeled as core points, while other points will be labeled as noise or border points.

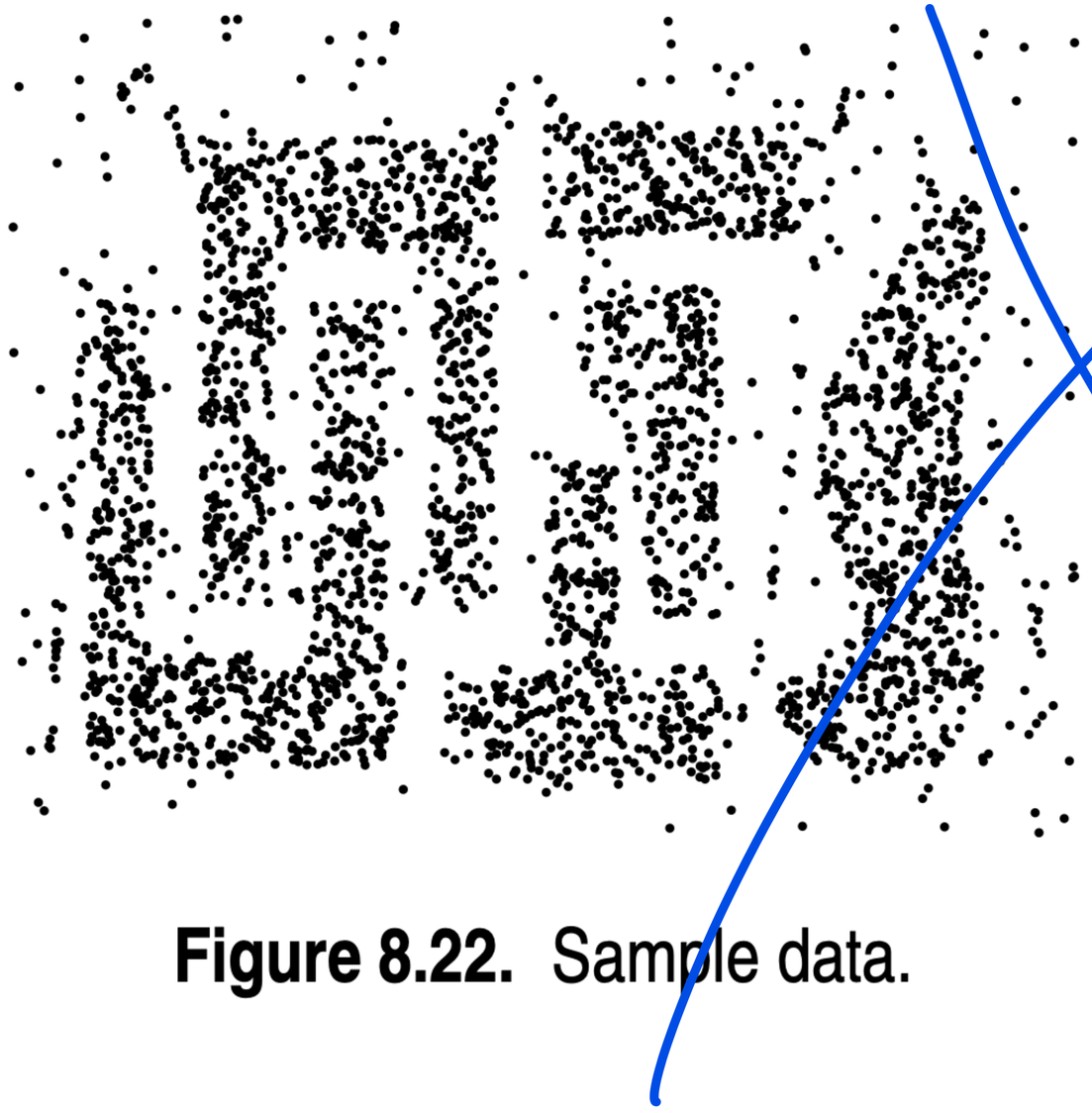


Figure 8.22. Sample data.

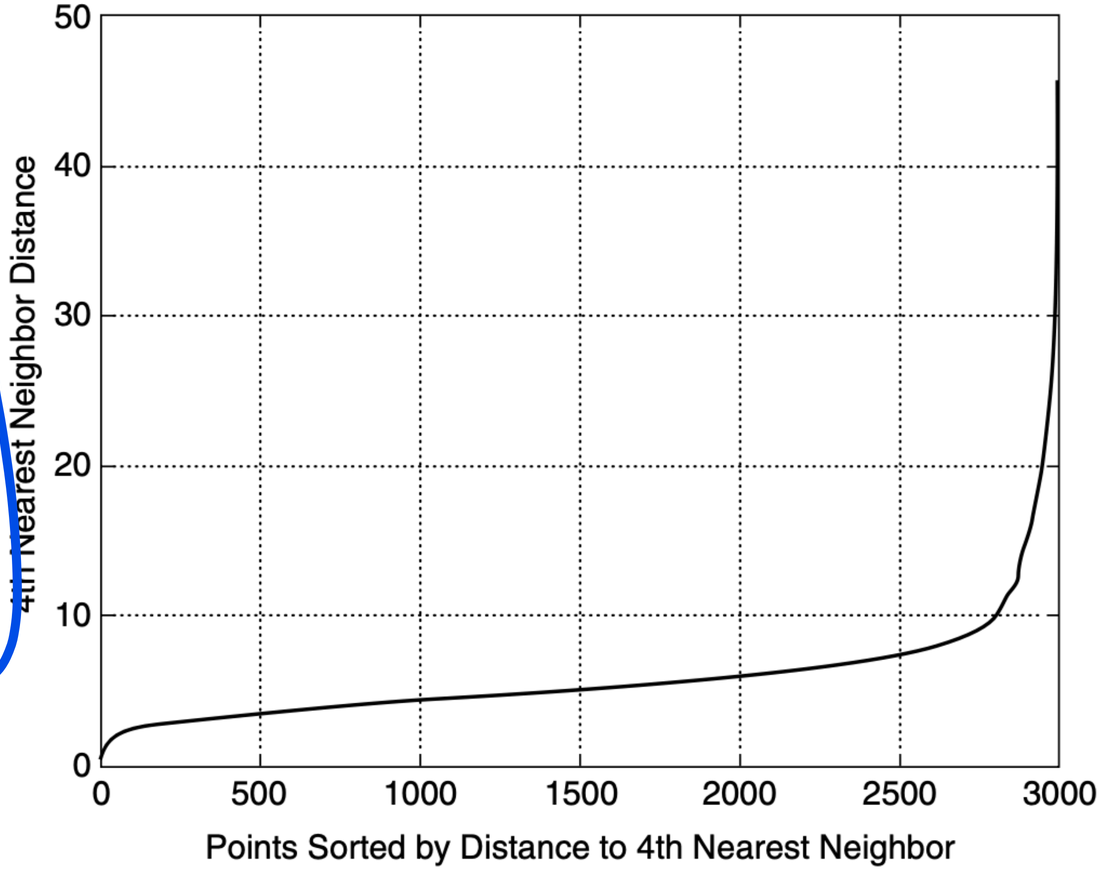


Figure 8.23. K-dist plot for sample data.

- DBSCAN can have trouble with density if the density of clusters varies widely.
- Consider Figure 8.24, which shows four clusters embedded in noise. The density of the clusters and noise regions is indicated by their darkness.
- The noise around the pair of denser clusters, A and B, has the same density as clusters C and D.
- If the Eps threshold is low enough that DBSCAN finds C and D as clusters, then A and B and the points surrounding them will become a single cluster.
- If the Eps threshold is high enough that DBSCAN finds A and B as separate clusters, and the points surrounding them are marked as noise, then C and D and the points surrounding them will also be marked as noise.

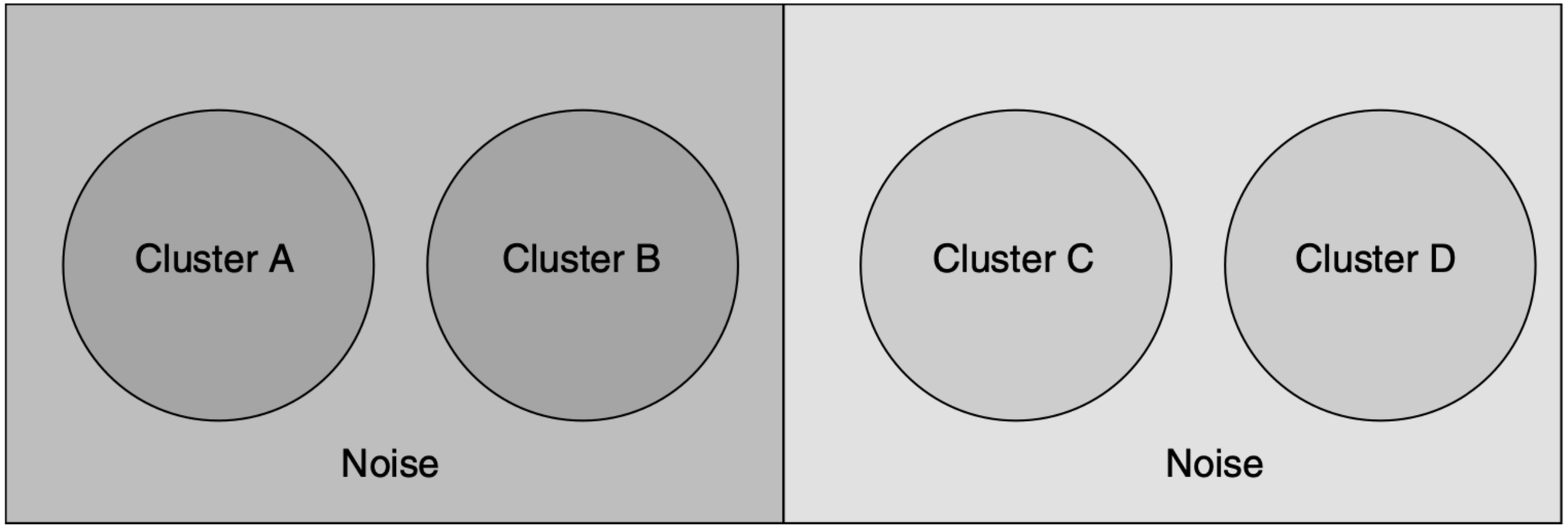


Figure 8.24. Four clusters embedded in noise.

- Because DBSCAN uses a density-based definition of a cluster, it is relatively resistant to noise and can handle clusters of arbitrary shapes and sizes.
- DBSCAN can find many clusters that could not be found using K-means.
- As indicated previously, however, DBSCAN has trouble when the clusters have widely varying densities.
- It also has trouble with high-dimensional data because density is more difficult to define for such data.
- Finally, DBSCAN can be expensive when the computation of nearest neighbors requires computing all pairwise proximities, as is usually the case for high-dimensional data.