Unit 1
Introduction

introduction, KDD, data pre-processing, similarity measurement, data visualization

Rupak Raj Ghimire

Objective

- Introduction
- KDD
- Data pre-processing,
- Similarity measurement
- Summary Statistics
- Data visualization

Data

- Data is a collection of facts, figures, statistics, or any other type of information that can be recorded and analyzed.
  - **Forms**: text, numbers, images, audio, video
  - **Source of Data**:
    - created by people, machines, or sensors, and
    - collected from a wide range of sources, such as websites, social media, databases, and sensors.

Information

- Information is a collection of data that has been processed, organized, or structured in a way that makes it meaningful, useful, and relevant to a particular context or purpose
- Information is created when data is analyzed, interpreted, and presented in a way that can be easily understood and used by humans or machines
- Information provides knowledge, insight, and understanding about a particular topic, situation, or phenomenon
- It can be used to support decision-making, problem-solving, and communication
- It conveyed in the form of reports, charts, graphs, tables, or other visual representations

Example of Data and Information

- Set of Marks = { 2, 5, 7, 9, 11 }
  - It is a data
  - This dataset is considered to be data because it is a collection of raw, unorganized numbers that don't necessarily convey any meaning or context on their own.
- Average of these numbers = 6.8
  - It is now information
  - Interpreted, presented it in a report or a chart, along with some context and explanation, such as "the average score on a test for a group of students", then we have turned the data into information.

Database

- A database is an organized collection of data that is stored and managed using specialized software
- A database allows users to store, retrieve, update, and delete data in an efficient manner (Operations)
- Example
  - The data in a relational database is organized into tables, which are composed of rows and columns.
    - Each row represents a record, and each column represents a specific piece of information about that record.
  - For example, a table in a customer database might include columns for the customer's name, address, phone number, and email address.

Database Management System (DBMS)

- A DBMS is a software system that is designed to manage and manipulate databases
- It provides a set of tools and services that allow users to store, access, modify, and maintain data in an organized and secure way
- A DBMS provides a way to create and manage databases, define the data structures, and enforce data integrity
  - Example: PostgreSQL, Oracle, MySQL etc.

Type of Databases

- Relational Database
- Object Oriented Database
- NoSQL
- Graph Database
- Network Database etc.

Retrieving Data from Database

- SQL is a programming language used to communicate with and manipulate databases.
- Type of SQL
  - Data Query Language (DQL) – select
  - Data Manipulation Language (DML) – insert, update,delete
  - Data Definition Language (DDL) – create
  - Data Control Language (DCL) – grant, revoke
- Reference
  - https://learnsql.com/blog/sql-basics-cheat-sheet/

Transactional Database

- An operational database system (also known as a transnational database system) is a type of database that is designed to support the day-to-day operations of an organization.
  - It is optimized for transnational processing, which involves capturing, storing, and updating data in real-time as business transactions occur
  - Typically used to support online transaction processing (OLTP), which involves frequent and rapid database access and updates by multiple users simultaneously

Transactional Database

- Operational database systems are designed to ensure data consistency, reliability, and availability
- ACID
  - Atomicity - Transaction
  - Consistency - Data Quality
  - Isolation - Concurrency
  - Durability - Recovery

Data warehouse

- A data warehouse refers to a data repository that is maintained separately from an organization’s operational databases
- A data warehouse is a subject-oriented, integrated, time-variant, and nonvolatile collection of data in support of management’s decision making process - William H. Inmon

Data warehouse

- Subject-oriented
  - A data warehouse is organized around major subjects such as customer, supplier, product, and sales.
  - Rather than concentrating on the day-to-day operations and transaction processing of an organization, a data warehouse focuses on the modeling and analysis of data for decision makers.
  - Data warehouses typically provide a simple and concise view of particular subject issues by excluding data that are not useful in the decision support process.

Data warehouse

- Integrated:
  - A data warehouse is usually constructed by integrating multiple heterogeneous sources, such as relational databases, flat files, and online transaction records.
  - Data cleaning and data integration techniques are applied to ensure consistency in naming conventions, encoding structures, attribute measures, and so on
- Time-variant:
  - Data are stored to provide information from an historic perspective (e.g., the past 5–10 years). Every key structure in the data warehouse contains,either implicitly or explicitly, a time element.

Data warehouse

- Nonvolatile:
  - A data warehouse is always a physically separate store of data transformed from the application data found in the operational environment. Due to this separation, a data warehouse does not require transaction processing, recovery, and concurrency control mechanisms. It usually requires only two operations in data accessing: initial loading of data and access of data.

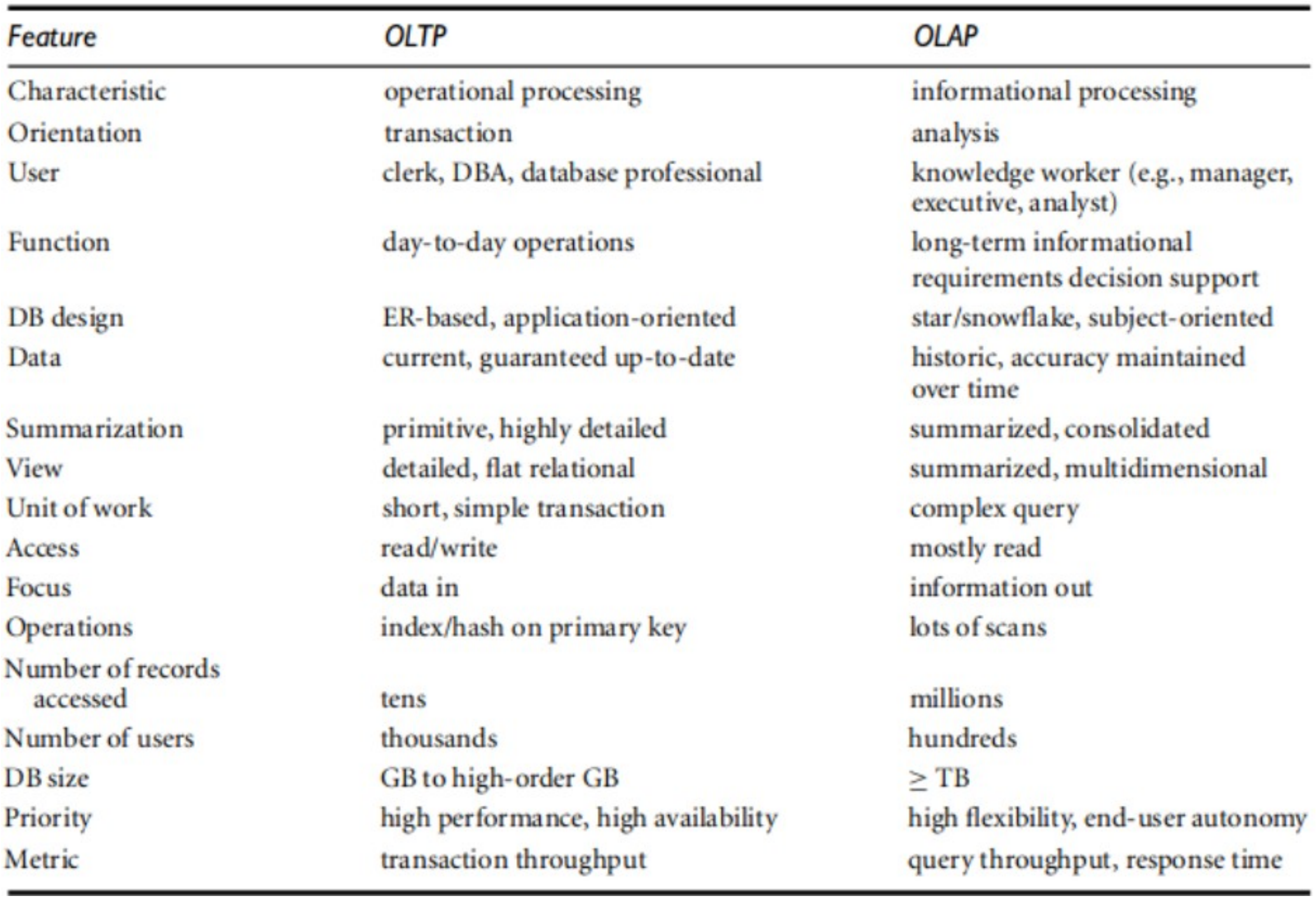
OLTP

- The major task of online operational database systems is to perform online transaction and query processing. These systems are called **online transaction processing** (OLTP) systems.
- They cover most of the day-to-day operations of an organization such as purchasing, inventory, manufacturing, banking, payroll, registration, and accounting.

OLAP

- Data warehouse systems, on the other hand, serve users or knowledge workers in the role of data analysis and decision making. Such systems can organize and present data in various formats in order to accommodate the diverse needs of different users. These systems are known as **online analytical processing** (OLAP) systems.

OLTP vs OLAP



Feature	*OLTP*	*OLAP*
Characteristic	operational processing	informational processing
Orientation	transaction	analysis
User	clerk, DBA, database professional	knowledge worker (e.g., manager, executive, analyst)
Function	day-to-day operations	long-term informational requirements decision support
DB design	ER-based, application-oriented	star/snowflake, subject-oriented
Data	current, guaranteed up-to-date	historic, accuracy maintained over time
Summarization	primitive, highly detailed	summarized, consolidated
View	detailed, flat relational	summarized, multidimensional
Unit of work	short, simple transaction	complex query
Access	read/write	mostly read
Focus	data in	information out
Operations	index/hash on primary key	lots of scans
Number of records accessed	tens	millions
Number of users	thousands	hundreds
DB size	GB to high-order GB	≥ TB
Priority	high performance, high availability	high flexibility, end-user autonomy
Metric	transaction throughput	query throughput, response time

Data Warehouse Models

- Enterprise Warehouse
- Data Marts
- Virtual Warehouse

Enterprise Warehouse

- Collects all of the information about subjects spanning the entire organization
- It provides corporate-wide data integration, usually from one or more operational systems or external information providers, and is cross-functional in scope
- It typically contains detailed data as well as summarized data
- Data range in size from a few gigabytes to hundreds of gigabytes, terabytes, or beyond
- An enterprise data warehouse may be implemented on traditional mainframes, computer super servers, or parallel architecture platforms
- It requires extensive business modeling and may take years to design and build

Data Marts

- A data mart contains a subset of corporate-wide data that is of value to a specific **group of users**.
- For example, a marketing data mart may confine its subjects to customer, item, and sales. The data contained in data marts tend to be summarized
- Depending on the source of data, data marts can be categorized as independent or dependent
- Independent data marts are sourced from data captured from one or more operational systems or external information providers, or from data generated locally within a particular department or geographic area
- Dependent data marts are sourced directly from enterprise data warehouses

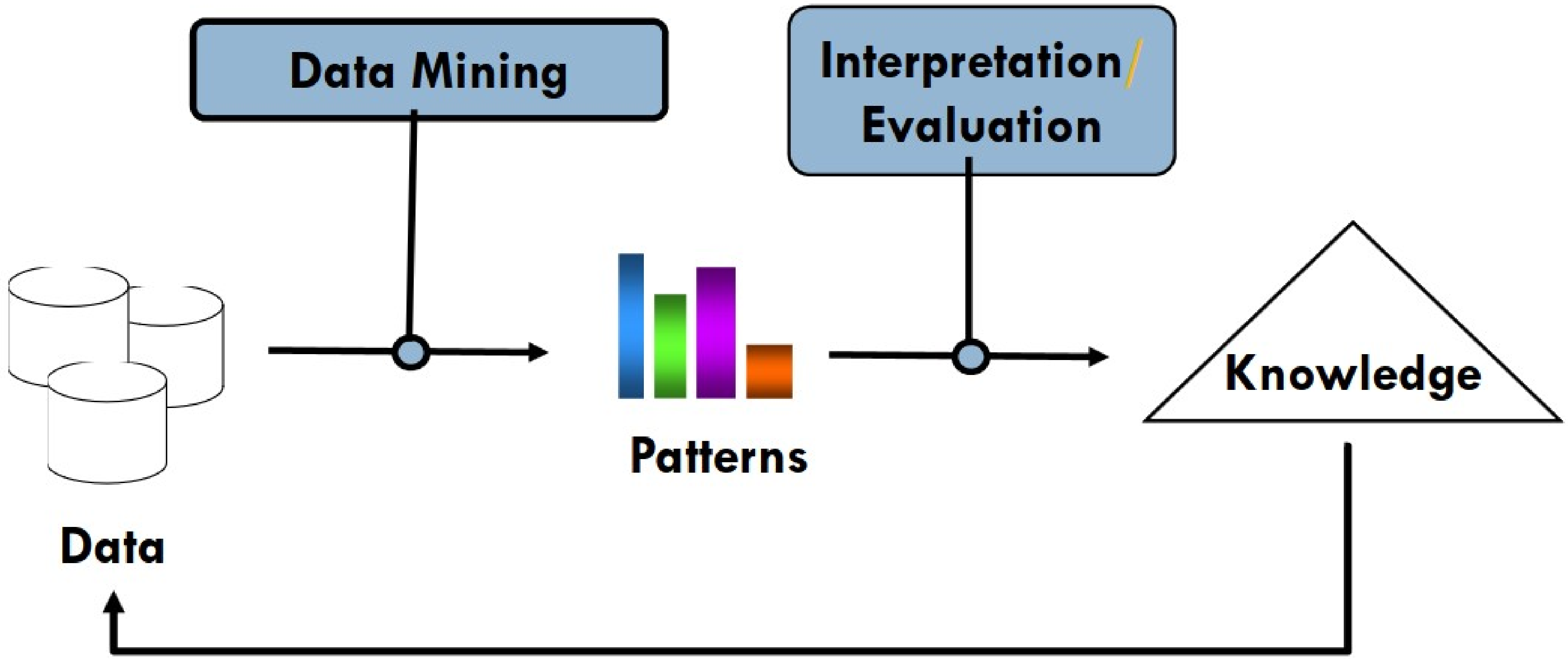
Virtual Warehouse

- **Virtual data warehouse** (VDW) is a set of views over operational databases. For efficient query processing, only some of the possible summary views may be materialized.
- A virtual warehouse is easy to build but requires excess capacity on operational database servers.
- The VDW acts as a logical view of the data, providing a unified view of multiple data sources without the need for physically storing the data in a single location

What is data mining?

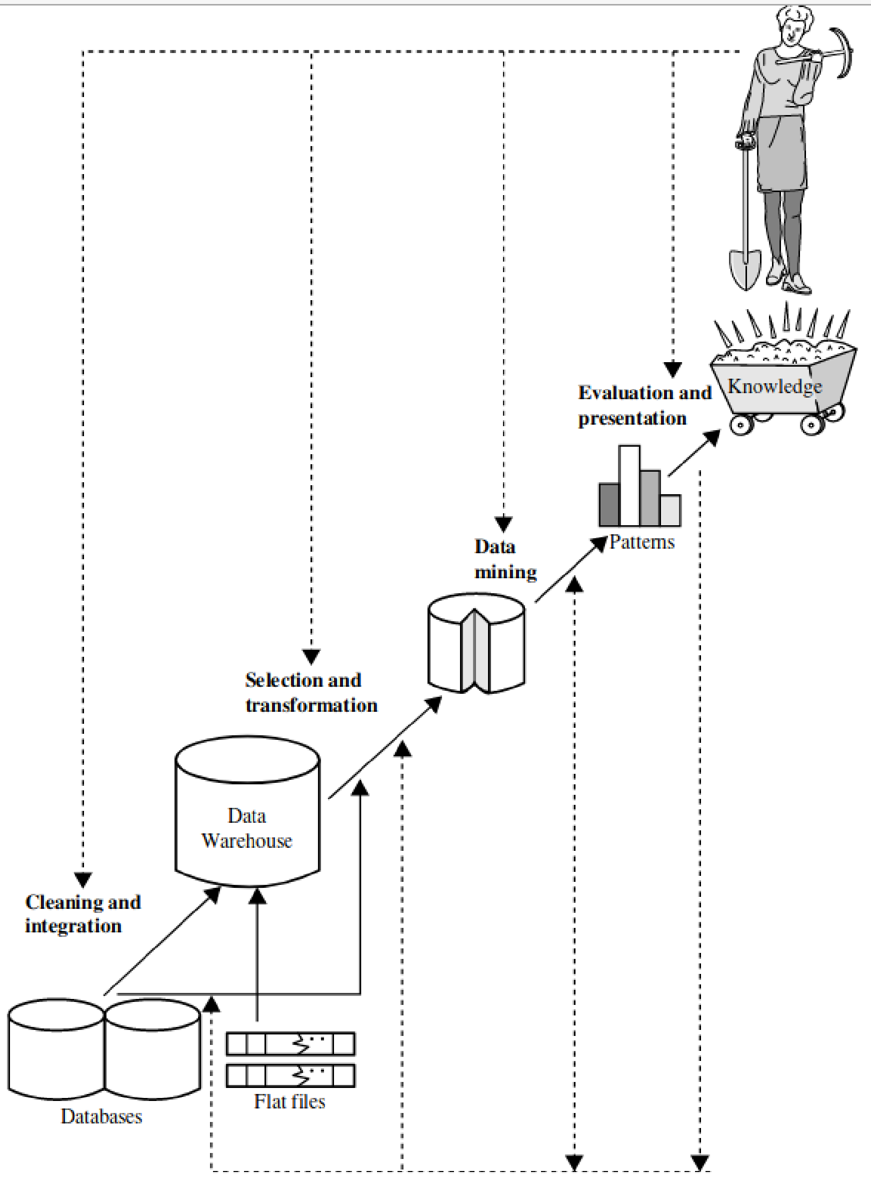
- Data mining is
  - Data mining is also called knowledge discovery in Data (KDD)
  - Extraction of useful patterns from data sources, e.g., databases, texts, web, image.
  - Patterns must be:
    - Valid
    - Novel
    - Potentially useful
    - Understandable

Knowledge Discovery in Data: Process



KDD – 7 steps

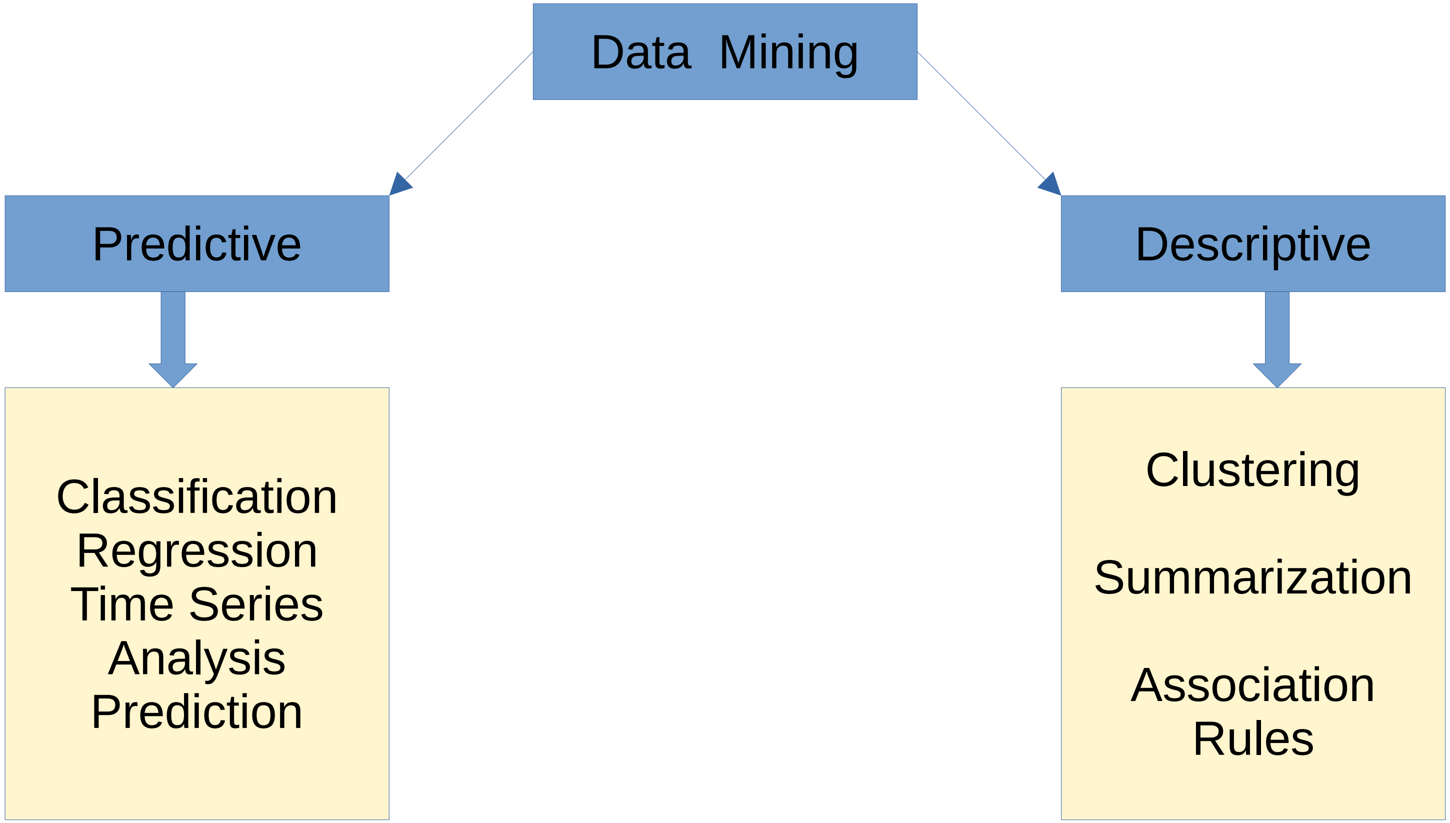
- Data Cleaning
- Data Integration
- Data Selection
- Data Transformation
- Data Mining
- Pattern Evaluation
- Knowledge Representation



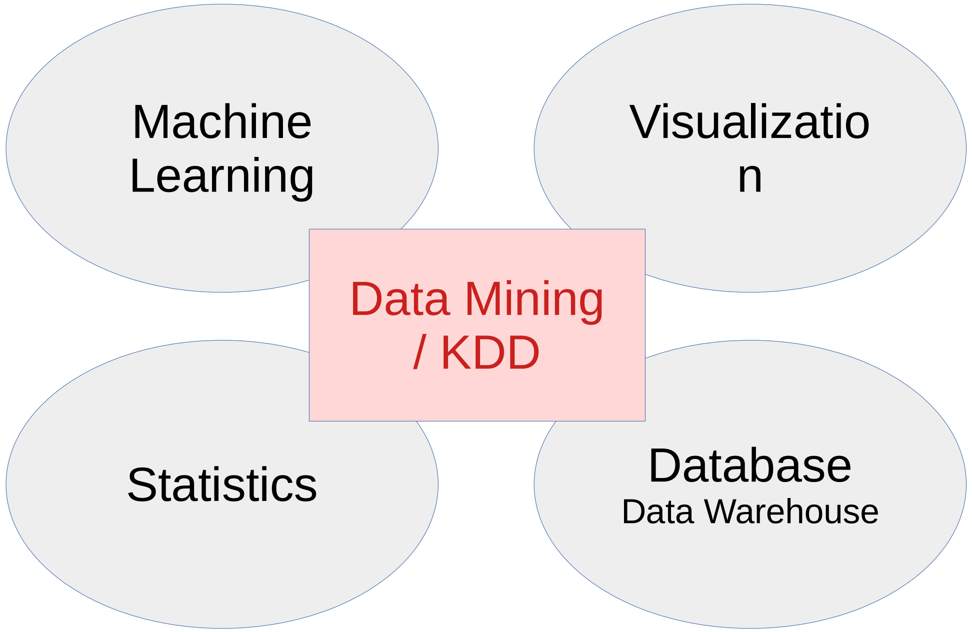
Data Mining Techniques

- The two "high-level" primary goals of data mining, in practice, are prediction and description.
  - **Prediction** involves using some variables or fields in the database to predict unknown or future values of other variables of interest.
  - **Description** focuses on finding human-interpretable patterns describing the data.
- Clustering
- Classification
- Association

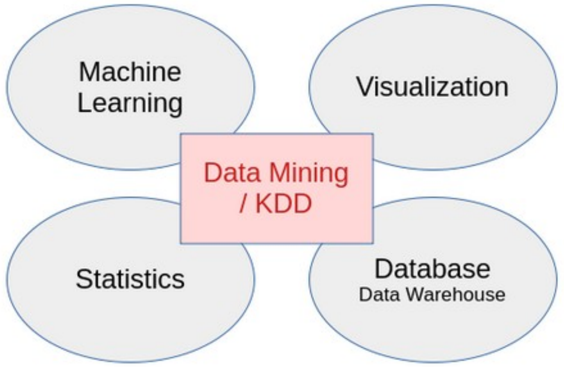
Data Mining Techniques



Related Fields



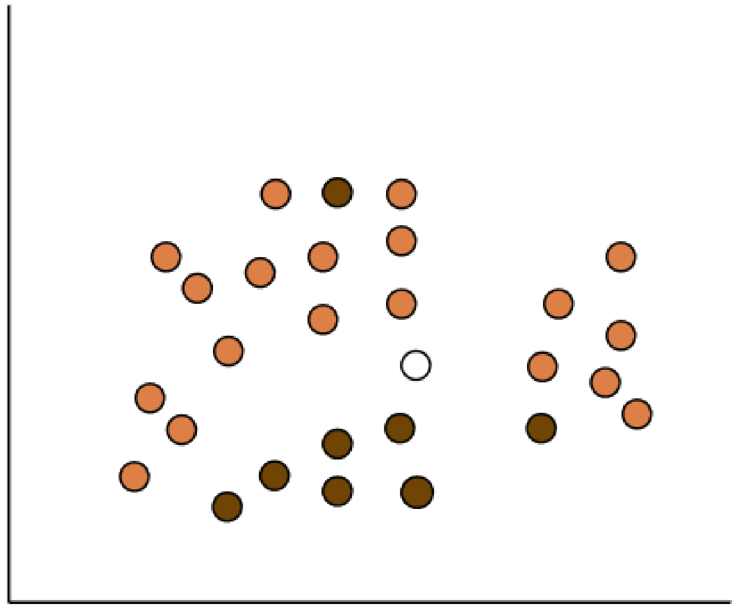
Related Fields



- Statistics
  - more theory-based
  - more focused on testing hypotheses
- Machine learning
  - more heuristic
  - focused on improving performance of a learning agent
  - also looks at real-time learning and robotics – areas not part of data mining
- Data Mining and Knowledge Discovery
  - integrates theory and heuristics
  - focus on the entire process of knowledge discovery, including data cleaning,learning, and integration and visualization of results

Classification

Learn a method for predicting the instance class from pre-labeled (classified) instances

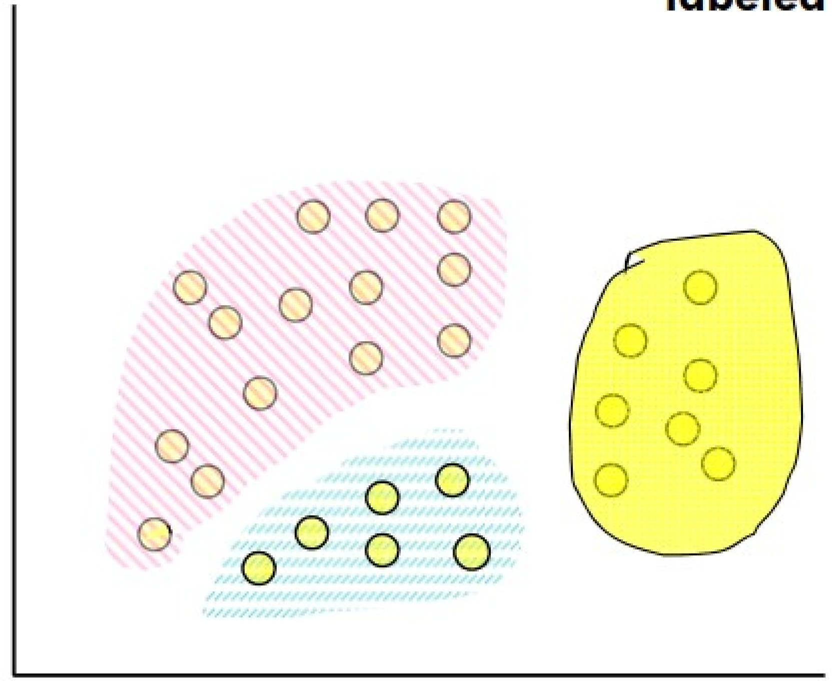


Many approaches: Statistics, Decision Trees, Neural Networks,

...

Clustering

Find "natural" grouping of instances given un-labeled data



Association Rules & Frequent Itemsets

Transactions

TID	Produce
1	MILK, BREAD, EGGS
2	BREAD, SUGAR
3	BREAD, CEREAL
4	MILK, BREAD, SUGAR
5	MILK, CEREAL
6	BREAD, CEREAL
7	MILK, CEREAL
8	MILK, BREAD, CEREAL, EGGS
9	MILK, BREAD, CEREAL

Frequent Itemsets:

Milk, Bread (4)
Bread, Cereal (3)
Milk, Bread, Cereal (2)
…

Rules:
Milk => Bread (66%)

Types of data

- Relational Data
- Graph Data
- Temporal Data
  - Time Series Data
  - Sequence Data
- Spatial Data
  - – location data , GPS, Coordinates, Map etc.
- Spatial-Temporal Data
  - Location with Time components
- Unstructured data
  - Text, review, comments etc.
- Semi-Structured Data
  - Published data, json, xml, html data etc.

Data Quality

- We need quality data for increasing the accuracy
- Pre-processing techniques can be used to enhance the data quality
  - Heterogeneous Data
    - Inconsistencies
    - Data format
  - Noise
  - Outliers
  - Redundancy

Data pre-processing Steps

- Exploratory Data Analysis
- Deal with Missing
- Deal with Duplicates and Outliers
- Encode Categorical Features
- Split dataset into training and test set
- Deal with Imbalanced Data

Measuring Similarity

- The data similarity can be computed using distance
  - Euclidean
  - Manhattan
  - Minkowski
  - Cosine
  - Pearson
  - Jaccard
  - Levenshtein
  - Hamming

Data Visualization

- Types of data visualization
  - Distribution plot
  - Box and Whisker Plot
  - Line Plot
  - Bar Plot
  - Scatter Plot
  - Histogram
  - Pie chart
  - Heatmap etc.

Thank you

Unit 2
Association Analysis

Basic concept, Use of Association Analysis, Apriori algorithm, pruning

Objective

- Association Analysis Concepts
- Application
- Algorithm for Association Analysis
- Solving a problem

Corelation between data

- It is the measure of degree of dependency between two variables
- Statistical Approach
  - Corelation Analysis
    - statistical method used to measure the strength of the linear relationship between two variables and compute their association
    - A high correlation points to a strong relationship between the two variables, while a low correlation means that the variables are weakly related

Problems

- Market Basket Analysis
  - How to arrange the items (placements) to increase the cross-selling opportunities
  - Where to display the new items
    - For example, if customers often buy bread and milk together, a store might place these items closer to each other
- Cross-Selling in Online Retail
  - Suggest additional products to customers based on the items they have added to their shopping carts or purchased.
  - This helps in increasing revenue through cross-selling.

Problems

- Customer Behavior Analysis
  - Understand purchasing patterns and preferences of customers to improve marketing strategies and personalized recommendations
    - This is widely used in e-commerce and online platforms to enhance the user experience

Problems

- Fraud Detection
  - Identify unusual patterns or associations in financial transactions that may indicate fraudulent activities.
  - For example, detecting instances where certain products are consistently bought together in fraudulent transactions.

Problems

- Telecommunications Network Optimization
  - Analyze call records to identify patterns of co-occurring calls and optimize network performance, leading to better resource allocation and improved service quality

Association Analysis

- Mining for associations among items in a large database of transactions is an important data mining function
- Association rules are statements of the form
  - {X1, X2, …, Xn} => Y, meaning that if we

    find all of X1, X2, ……… , Xn in the transaction then we have good chance of finding Y.
- Association analysis mostly applied in the field of market basket analysis, web-based mining, intruder detection

Market Basket Analysis

- It is the study of items that are purchased or grouped together in a single transaction or multiple, sequential transactions
- Used for
  - Make recommendations
  - Cross-sell
  - Up-sell
  - Offer coupons / discounts

Market Basket Analysis

- The analysis can be applied in various ways:
  - Develop combo offers based on products sold together.
  - Organize and place associated products/categories nearby inside a store.
  - Determine the layout of the catalog of an e-commerce site.
  - Control inventory based on product demands and what products sell together.

Few Terminologies

- Support
  - The support of an association pattern is the percentage of task-relevant data transaction for which the pattern is true

    Support (A): Number of tuples containing A / Total number of tuples

    Support (A = > B): Number of tuples containing A and B / Total number of tuples
  - While computing the association Minimum Support is used as threshold for computing

Few Terminologies

- Support
  - If minimum support is set too high, we could miss itemsets involving interesting rare items
    - e.g., expensive products
  - If minimum support is set too low, it is computationally expensive and the number of itemsets is very large

Few Terminologies

- Confidence
  - Confidence is defined as the measure of certainty or trustworthiness associated with each discovered pattern

    Confidence (A = > B): Number of tuples containing A and B / Total count of A

  - Confidence is usually given in percentage

Few Terminologies

- Item Set
  - A collection of one or more items.

    Example: {Milk, Bread, Diaper}

  - An itemset that contains k items is called k-itemset

- Frequent Itemset
  - An itemset whose support is greater than or equal to a minimum support threshold.

Few Terminologies

- Association Rule
  - An implication expression of the form X => Y, where X and Y are itemsets.

Example: {Milk, Diaper} => {Beer}

Few Terminologies

- Maximal Frequent Itemset
  - An itemset is maximal if none of its immediate supersets is frequent
- Closed Itemset
  - An itemset is closed if none of its immediate supersets has same support as of the itmeset

Few Terminologies

- Lift
  - Lift is a measure of the performance of a targeting model (association rule) at predicting or classifying cases as having an enhanced response with respect to the population as a whole, measured against a random choice targeting model.

$$\text{Lift} = P(Y \mid X) / P(Y)$$

  -

Association Rules Mining

- Given a set of transactions T, the goal of association rule mining is to find all rules having
  - support ≥ min_sup threshold and
  - confidence ≥ min_conf threshold
- Some of approaches for association rules mining are:
  - Brute-Force Approach
  - Frequent Itemset Generation Techniques

Brute- Force Approach

- List all possible association rules
- Compute the support and confidence for each rule
- Prune rules that fail to minimum support and minimum confidence level

- Pros
  - Easy Computation
  - Easy Implementation
  - Works perfect for smaller number of itemset
- Cons
  - Computationally expensive

Frequent Itemset Generation

- Formulate some ways to
  - Reduce the number of candidates
  - Reduce the number of transactions
  - Reduce the number of comparison
- Pros
  - Faster computation
  - Faster convergence toward solution
- Cons
  - Still slower (mostly depends on the min_support threshold)

Apriori Approach

- It is based on the Apriori Principle
  - Supersets of non-frequent item are also non-frequent
  - Or, If an itemset is frequent, then all of its subset also be frequent
- Two step Approach
  1) Frequent Itemset generation
  2) Rule Generation
- It use a level-wise search, k-itemsets are used to explore k+1 itemsets.
- At first, the set of frequent itemset is found and used to generate to frequent itemset at next level and so on

Apriori Algorithm

- Algorithm
  - Read the transaction database and get support for each itemset, compare the support with minimum support to generate frequent itemset at level 1.
  - Use join to generate a set of candidate k-itmesets at next level.
  - Generate frequent ietmsets at next level using minimum support.
  - Repeat step 2 and 3 until no frequent itme sets can be generated.
  - Generate rules form frequent itemsets from level 2 onwards using minimum confidence.

Example

- Separate PDF

Reference

- Reference Reading:
  - Book: Data Mining Concepts and Techniques – Morgan
    Chapter 6 Mining Frequent Patterns, Associations, and Correlations

Limitation of Apriori Algorithm

- Issues of Apriori Algorithm
  - Speed
  - High computational cost
  - Difficult to handle parallelism

Frequent Pattern Growth Algorithm

- Commonly Known as FP-Growth
- Improved version of Apriori Algorithm
- FP-growth algorithm is a tree-based algorithm for frequent itemset mining

- The algorithm represents the data in a tree structure known as FP-tree, responsible for maintaining the association information between the frequent items

FP-Growth

- The algorithm compresses frequent items into an FP-tree from the database while retaining association rules.

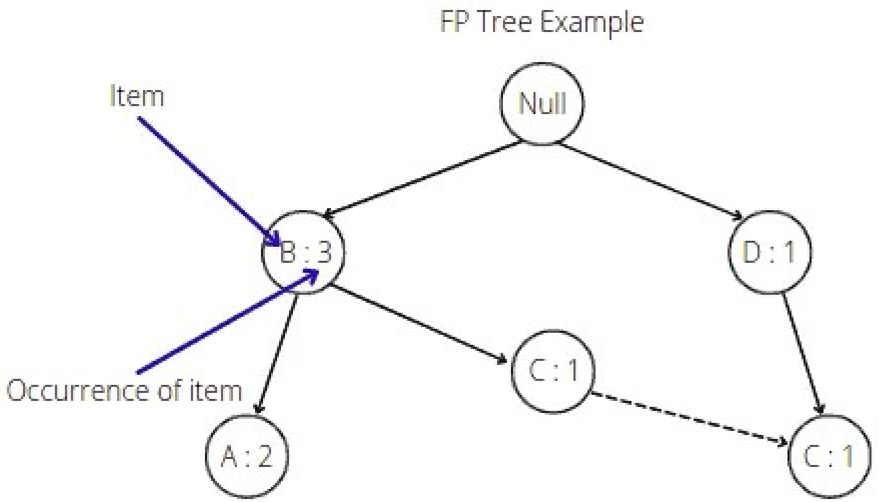
- Then it splits the database data into a set of conditional databases (a special kind of projected database), each of which is associated with one frequent data item.

FP-Tree

- FP-tree is the core concept of the FP-growth algorithm.
- The FP-tree is a compressed representation of the database itemset, storing the DB itemset in memory and keeping track of the association between items.
- The tree is constructed by taking each itemset and adding it as a subtree.
- The FP-tree's whole idea is that items that occur more frequently will be more likely to be shared.

FP-Tree

- The root node in the FP-tree is null.
- Each node of the subtree stores at least the item name and the support (or item occurrence) number.
- Additionally, the node may contain a link to the node with the same name from another subtree (represents another itemset from the database).



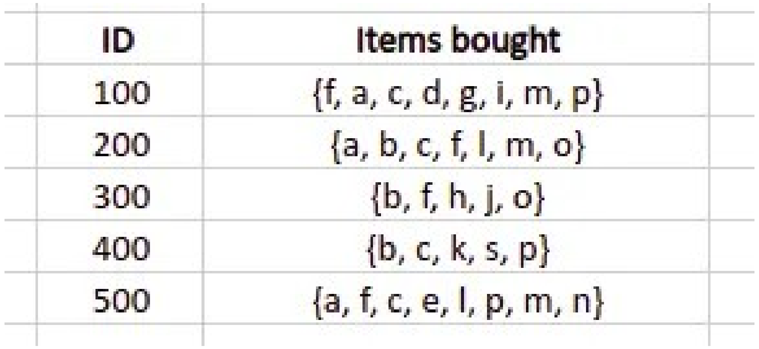
Building FP-Tree

- The FP-growth algorithm uses the following steps to build FP-tree from the database.
  - Scan itemsets from the database for the first time
  - Find frequent items (single item patterns) and order them into a list L in frequency descending order.

    For example, L = {A:5, C:3, D;2, B:1}
  - For each transaction order its frequent items according to the order in L
  - Scan the database the second time and construct FP-tree by putting each frequency ordered transaction onto it

FP-Tree (Example)

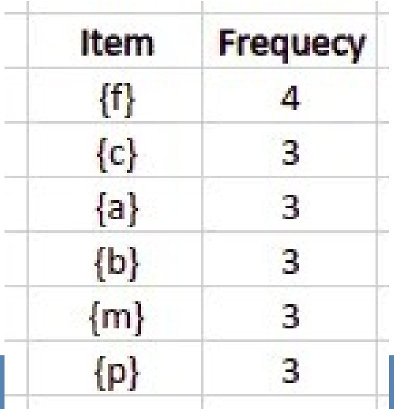
- Create a FP Tree of following dataset



ID	Items bought
100	{f, a, c, d, g, i, m, p}
200	{a, b, c, f, l, m, o}
300	{b, f, h, j, o}
400	{b, c, k, s, p}
500	{a, f, c, e, l, p, m, n}

FP-Tree (Example)

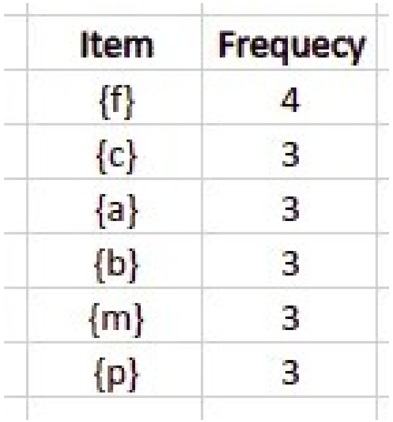
- Step 1: Item wise support count and eliminate the item that has support < min_support
  - Scan the dataset,
  - Create a frequency table containing each item from the database
  - Arrange them in descending order.
  - Filter items with a support value less than the minimum support
    - For example, let’s set up the minimum support value equal to 3. In that case, we will get the following frequency table:



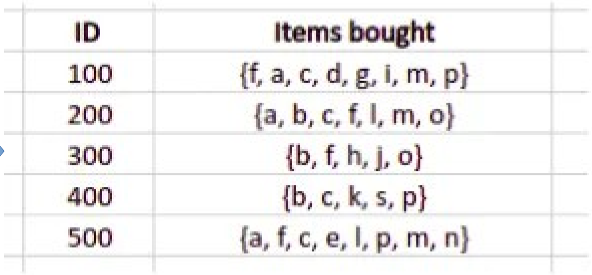
Item	Frequecy
{f}	4
{c}	3
{a}	3
{b}	3
{m}	3
{p}	3

FP-Tree (Example)

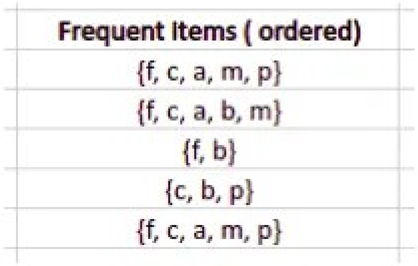
- Step 2: Rebuild dataset with items that created in step 1
  - Scan the database the second time and arrange elements based on the frequency table
  - Items with higher a frequency number will come first
    - if two items have the same frequency number they will be arranged in alphabetical order



Item	Frequecy
{f}	4
{c}	3
{a}	3
{b}	3
{m}	3
{p}	3



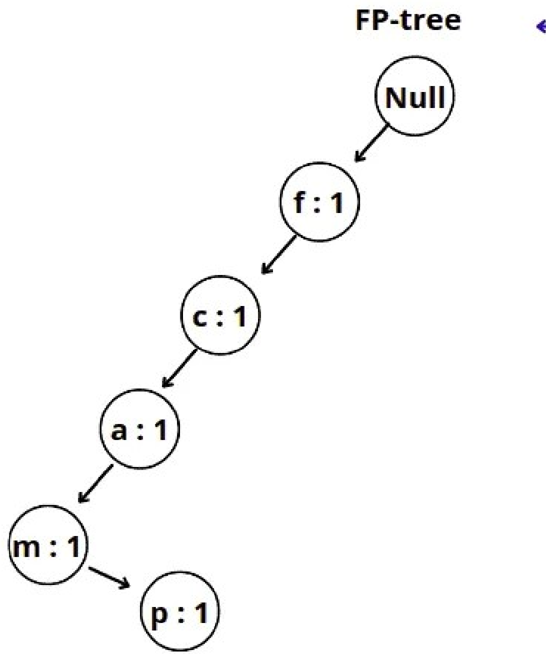
ID	Items bought
100	{f, a, c, d, g, i, m, p}
200	{a, b, c, f, l, m, o}
300	{b, f, h, j, o}
400	{b, c, k, s, p}
500	{a, f, c, e, l, p, m, n}

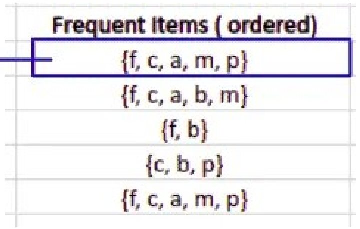


Frequent Items (ordered)
{f, c, a, m, p}
{f, c, a, b, m}
{f, b}
{c, b, p}
{f, c, a, m, p}

FP-Tree (Example)

- Step 3: Build Tree
  - We will create a tree based on the frequent items table of step 2
  - Scan through each dataset and build the tree => Parent Node = NULL
  - Add the first item

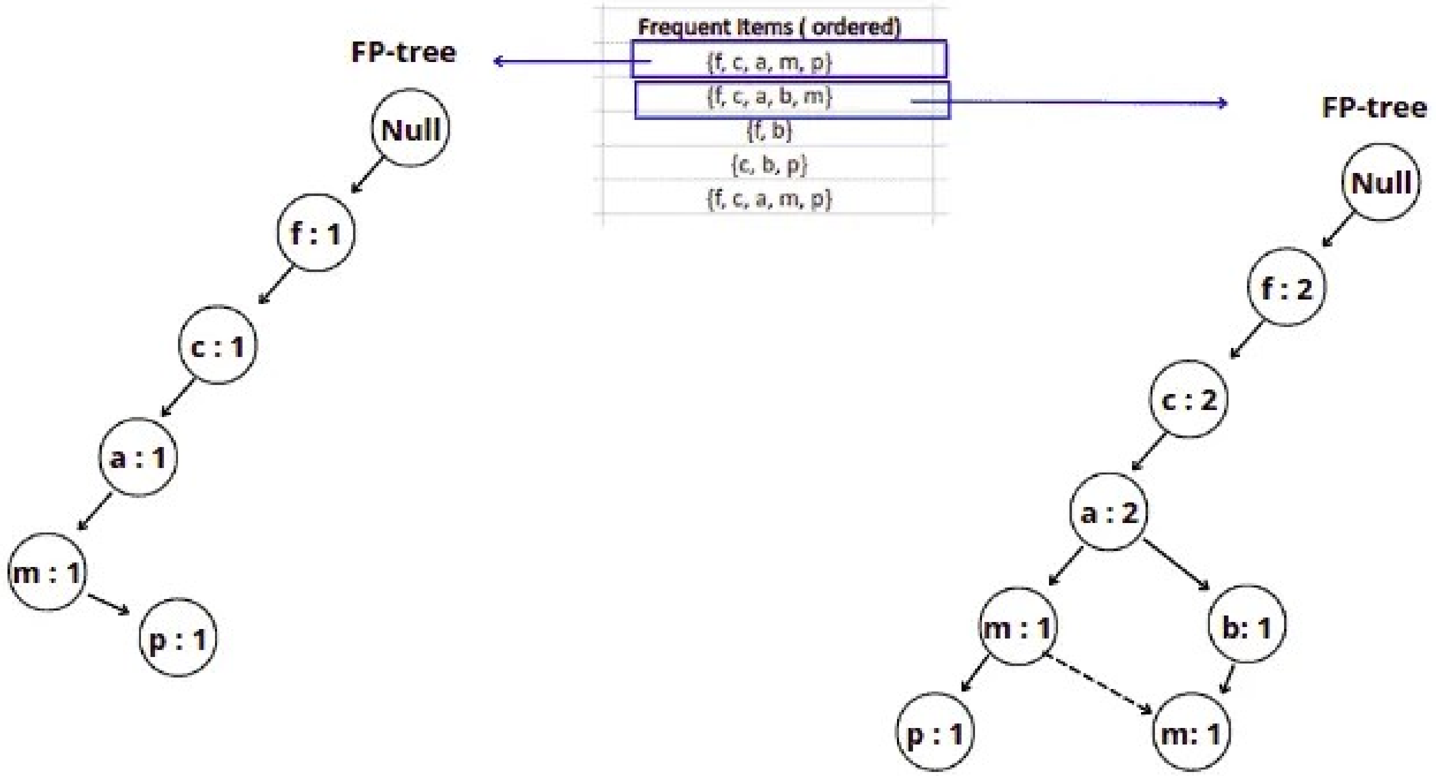




Frequent Items (ordered)
{f, c, a, m, p}
{f, c, a, b, m}
{f, b}
{c, b, p}
{f, c, a, m, p}

FP-Tree (Example)

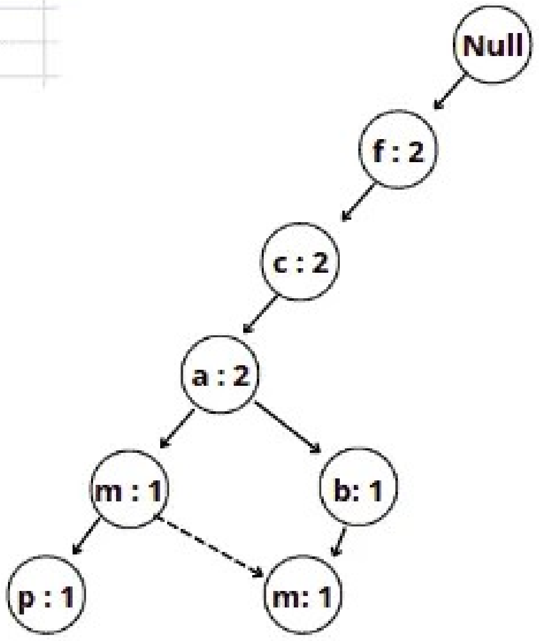
- Add 2nd Item of itemset



Frequent Items (ordered)
{f, c, a, m, p}
{f, c, a, b, m}
{f, b}
{c, b, p}
{f, c, a, m, p}

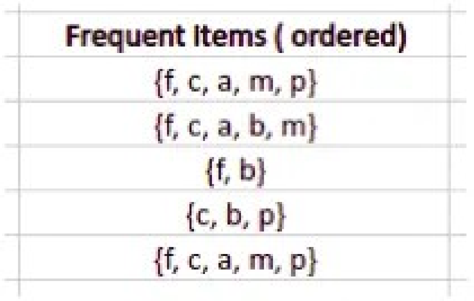
FP-Tree (Example)

- As we add the same element to the tree, we increment the support.
- But after item a we created a new node for item b because there was no item b in our initial tree after item a.
- And we have linked items m together because this is the same element located in different subtrees.

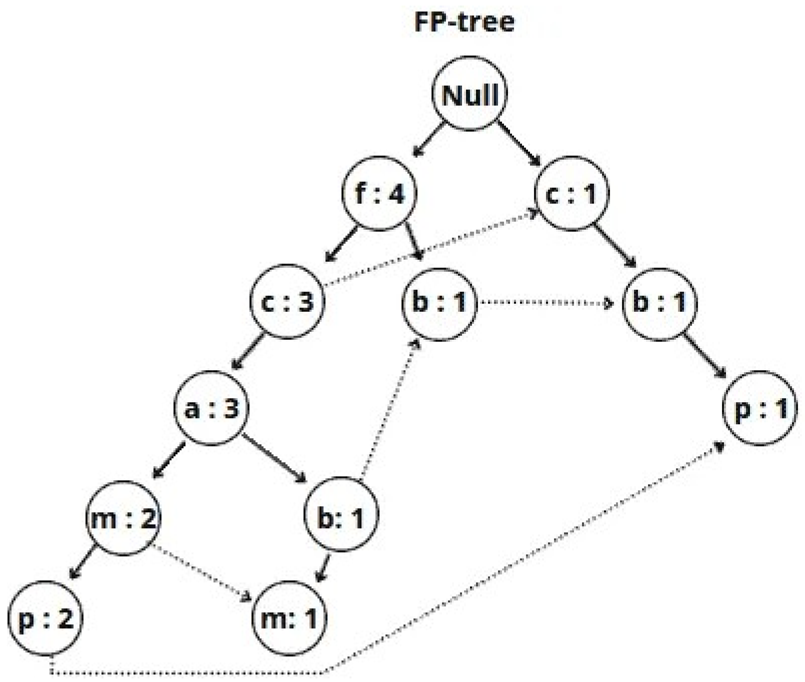


FP-Tree (Example)

- Add the all the dataset and populate the tree



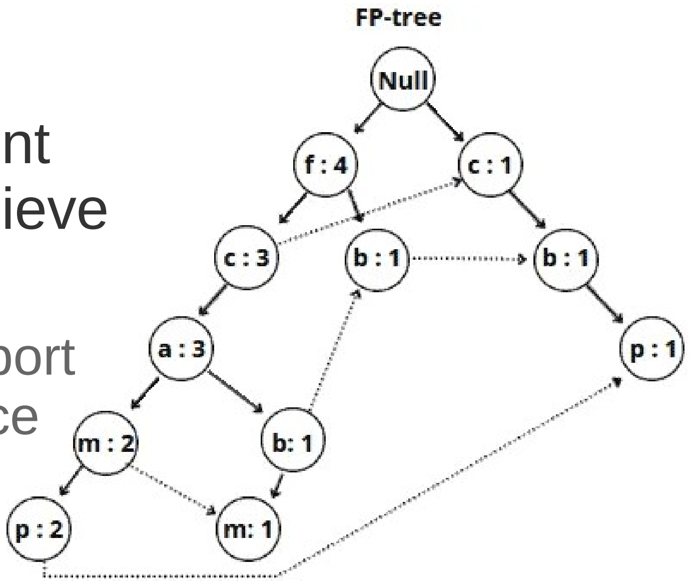
Frequent Items (ordered)
{f, c, a, m, p}
{f, c, a, b, m}
{f, b}
{c, b, p}
{f, c, a, m, p}



FP-Tree (Example)

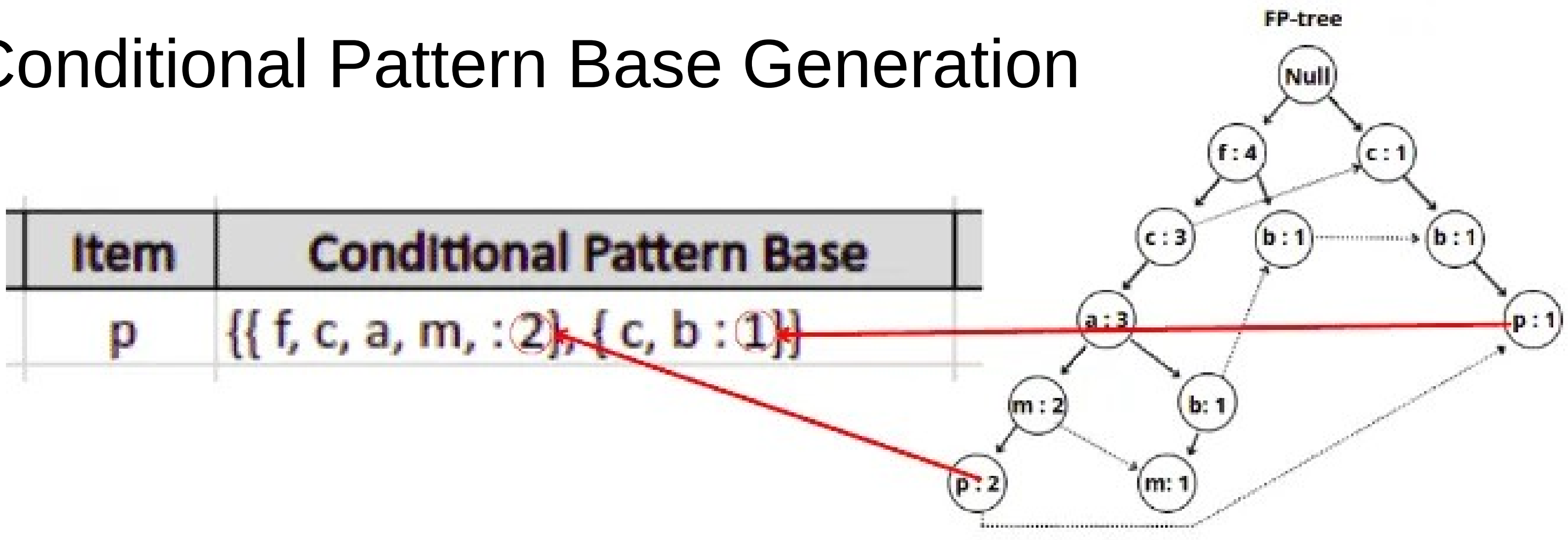
- Step 4: Build Association Rules
  - It will take the item with the minor support count and trace that item through the FP-tree to achieve that goal.
    - In our example, the item p has the lowest support count, and the FP-growth algorithm will produce the following paths:

      { {f, c, a, m, p : 2}, {c, b, p : 1} }.
    - Note: The item p is located in two different subtrees of the FP-tree, so the algorithm traced both paths and added the minimum support value for every path.



FP-Tree (Example)

- Conditional Pattern Base Generation

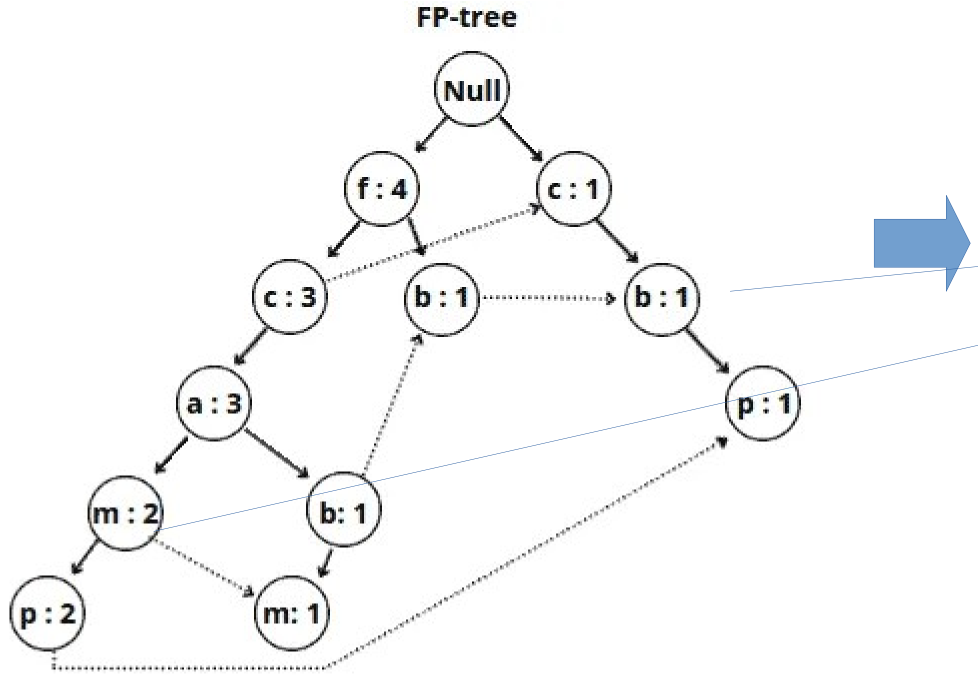


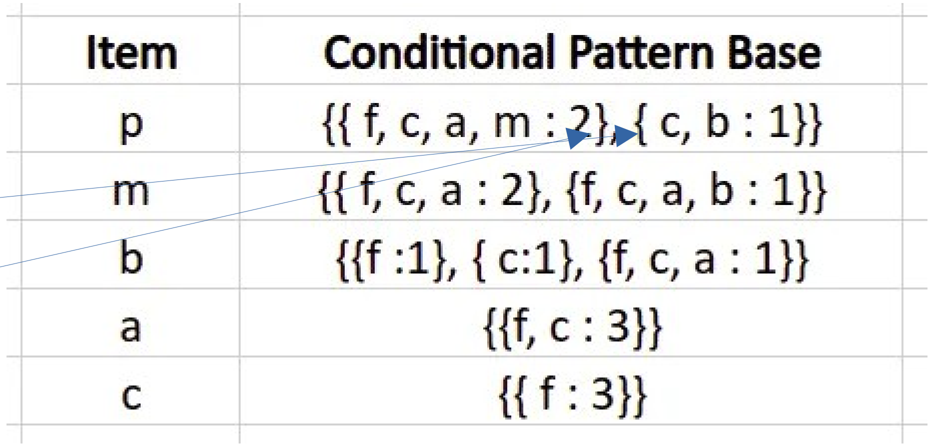
Item	Conditional Pattern Base
p	{{ f, c, a, m, :2}, { c, b :1}}

  – Similarly, the FP-growth will build the conditional pattern base table for all of the items from the FP-tree.

FP-Tree (Example)

- Conditional Pattern Base Generation
  - Start with leaf node and traverse upward (except those attached to NULL node)





Item	Conditional Pattern Base
p	{{ f, c, a, m : 2}, { c, b : 1}}
m	{{ f, c, a : 2}, {f, c, a, b : 1}}
b	{{f :1}, { c:1}, {f, c, a : 1}}
a	{{f, c : 3}}
c	{{ f : 3}}

FP-Tree (Example)

- Conditional FP Tree Generation

  Minimum support =3

  - get all items from the Conditional Pattern Base column that satisfy the minimum support requirement.
  - Let’s calculate elements’ occurrences for the p item:

    { f, c, a, m : 2 }, { c, b : 1 } - > { f: 2, c:3, a:2, m:2, b:1 }

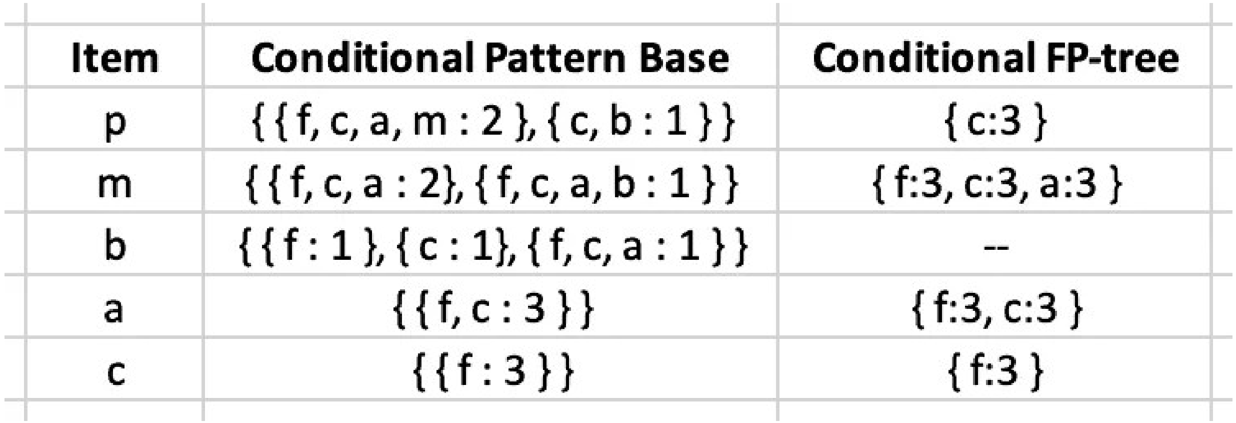
  - Only item c appears three times and satisfies the minimum support requirement.
  - That means the algorithm will remove all other items except c.

FP-Tree (Example)

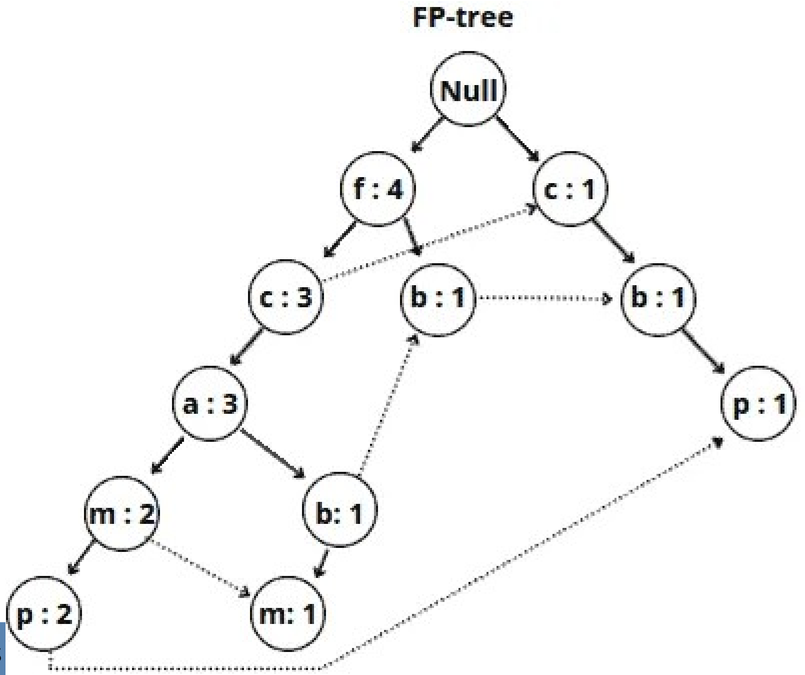
- Conditional FP Tree Generation

Minimum support =3

  - After removing items that do not meet the minimum support requirement, the algorithm will construct the following table:

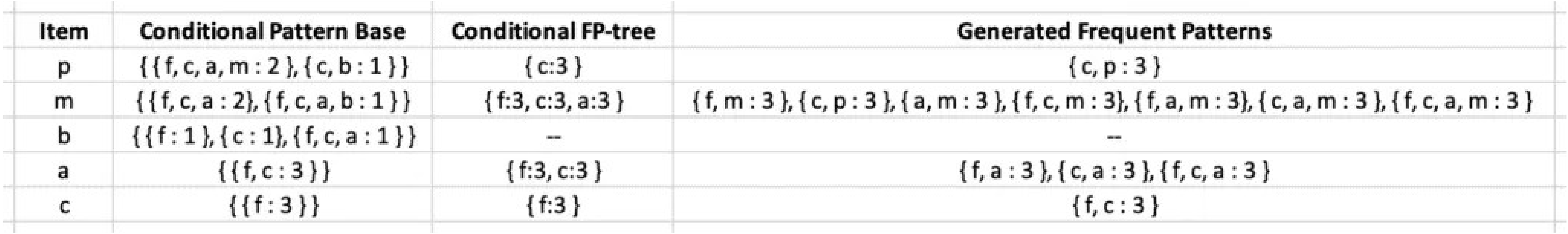


Item	Conditional Pattern Base	Conditional FP-tree
p	{ { f, c, a, m : 2 }, { c, b : 1 } }	{ c:3 }
m	{ { f, c, a : 2}, { f, c, a, b : 1 } }	{ f:3, c:3, a:3 }
b	{ { f : 1 }, { c : 1}, { f, c, a : 1 } }	--
a	{ { f, c : 3 } }	{ f:3, c:3 }
c	{ { f : 3 } }	{ f:3 }



FP-Tree (Example)

- Generate frequent patterns
  - Generate frequent patterns by pairing the items of the Conditional FP-tree column with the corresponding item from the Item column.
    - For example, for the first row
      - { c:3 } from the Conditional FP-tree column,
      - create its combination with the p element and add the support count value



Item	Conditional Pattern Base	Conditional FP-tree	Generated Frequent Patterns
p	{{f, c, a, m : 2 }, { c, b : 1 }}	{ c:3 }	{ c, p : 3 }
m	{{ f, c, a : 2}, { f, c, a, b : 1 }}	{ f:3, c:3, a:3 }	{ f, m : 3 }, { c, p : 3 }, { a, m : 3 }, { f, c, m : 3}, { f, a, m : 3}, { c, a, m : 3 }, { f, c, a, m : 3 }
b	{{ f : 1 }, { c : 1}, { f, c, a : 1 }}	--	--
a	{{ f, c : 3 }}	{ f:3, c:3 }	{ f, a : 3 }, { c, a : 3 }, { f, c, a : 3 }
c	{{ f : 3 }}	{ f:3 }	{ f, c : 3 }

FP-Tree (Example)

- Generate Association Rules

Confidence : 70%

  - Calculate the support and confidence for items in generated frequent pattern as done in Apriori Algorithm

Do it Yourself

- Explore of FP-growth algorithm using Python
  - mlextend library

```
from mlxtend.frequent_patterns import fpgrowth
from mlxtend.frequent_patterns import association_rules

res    = fpgrowth(dataset,min_support=0.05, use_colnames=True)
rules = association_rules(res, metric="lift", min_threshold=1)
```

- Reference: https://hands-on.cloud/implementation-of-fp-growth-algorithm-using-python/

Thank you

Apriori Algorithm

The Apriori Algorithm is an influential algorithm for mining frequent itemsets for boolean association rules.

Key Concepts :

Frequent Itemsets: The sets of item which has minimum support (denoted by Li for ith-Itemset).

Apriori Property: Any subset of frequent itemset must be frequent.

Join Operation: To find Lk , a set of candidate k-itemsets is generated by joining Lk-1 with itself.

Pseudo code

- Join Step: C_k is generated by joining L_{k-1} with itself
- Prune Step: Any (k-1)-itemset that is not frequent cannot be a subset of a frequent k-itemset
- Pseudo-code:

```
$C_k$: Candidate itemset of size k
$L_k$ : frequent itemset of size k

$L_1$ = {frequent items};
for ($k$ = 1; $L_k$ !=∅; $k$++) do begin
    $C_{k+1}$ = candidates generated from $L_k$;
    for each transaction $t$ in database do
        increment the count of all candidates in $C_{k+1}$
        that are contained in $t$
    $L_{k+1}$ = candidates in $C_{k+1}$ with min_support
end
return $\cup_k L_k$;
```

Advantages

1. Easy to understand algorithm
2. Join and Prune steps are easy to implement on large itemsets in large databases

Disadvantages

1. It requires high computation if the itemsets are very large and the minimum support is kept very low.
2. The entire database needs to be scanned.

Methods to Improve Apriori's Efficiency

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

<u>Example of Apriori:</u> Support threshold=50%, Confidence= 60%

Transaction	List of items
T1	I1,I2,I3
T2	I2,I3,I4
T3	I4,I5
T4	I1,I2,I4
T5	I1,I2,I3,I5
T6	I1,I2,I3,I4

Solution:

Support threshold=50% => 0.5*6= 3 => min_sup=3

1. Count Of Each Item

Item	Count
I1	4
I2	5
I3	4
I4	4
I5	2

2. Prune Step: TABLE -2 shows that I5 item does not meet min_sup=3, thus it is deleted, only I1, I2, I3, I4 meet min_sup count.

Item	Count
I1	4
I2	5
I3	4
I4	4

3. Join Step: Form 2-itemset. From **TABLE-1** find out the occurrences of 2-itemset.

Item	Count
I1,I2	4
I1,I3	3
I1,I4	2
I2,I3	4
I2,I4	3
I3,I4	2

4. Prune Step: TABLE -4 shows that item set {I1, I4} and {I3, I4} does not meet min_sup, thus it is deleted.

Item	Count
I1,I2	4
I1,I3	3
I2,I3	4
I2,I4	3

5. Join and Prune Step: Form 3-itemset. From the **TABLE- 1** find out occurrences of 3-itemset. From **TABLE-5**, find out the 2-itemset subsets which support min_sup.

We can see for itemset {I1, I2, I3} subsets, {I1, I2}, {I1, I3}, {I2, I3} are occurring in **TABLE-5** thus {I1, I2, I3} is frequent.

We can see for itemset {I1, I2, I4} subsets, {I1, I2}, {I1, I4}, {I2, I4}, {I1, I4} is not frequent, as it is not occurring in **TABLE-5** thus {I1, I2, I4} is not frequent, hence it is deleted.

Item
I1,I2,I3
I1,I2,I4
I1,I3,I4
I2,I3,I4

Only {I1, I2, I3} is frequent.

6. Generate Association Rules: From the frequent itemset discovered above the association could be:

{I1, I2} => {I3}

Confidence = support {I1, I2, I3} / support {I1, I2} = (3/ 4)* 100 = 75%

{I1, I3} => {I2}

Confidence = support {I1, I2, I3} / support {I1, I3} = (3/ 3)* 100 = 100%

{I2, I3} => {I1}

Confidence = support {I1, I2, I3} / support {I2, I3} = (3/ 4)* 100 = 75%

{I1} => {I2, I3}

Confidence = support {I1, I2, I3} / support {I1} = (3/ 4)* 100 = 75%

{I2} => {I1, I3}

Confidence = support {I1, I2, I3} / support {I2 = (3/ 5)* 100 = 60%

{I3} => {I1, I2}

Confidence = support {I1, I2, I3} / support {I3} = (3/ 4)* 100 = 75%

This shows that all the above association rules are strong if minimum confidence threshold is 60%.

Hash-based technique

(hashing itemsets into corresponding buckets)

A hash-based technique can be used to reduce the size of the candidate k-itemsets, Ck, for k > 1. For example, when scanning each transaction in the database to generate the frequent 1-itemsets, L1, we can generate all the 2-itemsets for each transaction, hash (i.e., map) them into the different buckets of a hash table structure, and increase the corresponding bucket counts (Figure 6.5). A 2-itemset with a corresponding bucket count in the hash table that is below the support threshold cannot be frequent and thus should be removed from the candidate set. Such a hash-based technique may substantially reduce the number of candidate k-itemsets examined (especially when k = 2).

Dynamic itemset counting

(adding candidate itemsets at different points during a scan)

A dynamic itemset counting technique was proposed in which the database is partitioned into blocks marked by start points. In this variation, new candidate itemsets can be added at any start point, unlike in Apriori, which determines new candidate itemsets only immediately before each complete database scan. The technique uses the count-so-far as the lower bound of the actual count. If the count-so-far passes the minimum support, the itemset is added into the frequent itemset collection and can be used to generate longer candidates. This leads to fewer database scans than with Apriori for finding all the frequent itemsets.

Reference Reading:

Book: Data Mining Concepts and Techniques - Morgan
Chapter 6 Mining Frequent Patterns, Associations, and Correlations

Classification

Unit 3: Classification

Concept, Evaluation, Rule Based Classifiers, Decision Tree, Baysian Classifier

–

School of Mathematical Sciences
Institute of Science and Technology (IoST), TU

Feb, 2025

Table of contents

Classification

What is classification?

Classification is a supervised learning task where a model assigns discrete labels (classes) to input data based on learned patterns from labeled training examples

Input Features (X)

Variables describing the data (e.g., pixel values, text embeddings)

Output Labels (y)

Categorical targets (e.g., spam/not spam, positive/negative)

Model

A function $f(X) \rightarrow y$ (e.g., logistic regression, decision trees)

Classification - Defination

Given a collection of records (training dataset), each records are characterized by a *tuple*(x, y), where x is the **attribute set** and y is the **class label**

x: attribute, predictor, independent variable, input
y: class, response, dependent variable, output

Task

Learn a model that maps each attribute set x into one of the predefined class labels y

Model

A classification model is an **abstract representation** of the relationship between the attribute set and the class label

Classification - Examples

Email message categorization

Dataset: Features extracted from email message header and content
Class Label: spam or non-spam

Identifying tumor cells

Dataset: Features extracted from x-rays or MRI scans
Class Label: malignant or benign cells

Cataloging galaxies

Dataset: Features extracted from telescope images
Class Label: Elliptical, spiral, or irregular-shaped galaxies

Classification Models - Predictive, Descriptive

Predictive Model

- Used to classify the previously unlabeled instances
- A good classification model must provide accurate predictions with a fast response time

Descriptive Model

- Used to identify the characteristics that distinguish instances from different classes
- This is particularly useful for critical applications, such as medical diagnosis, where it is insufficient to have a model that makes a prediction without justifying how it reaches such a decision

Classification Model (cont.)

Predictive or Descriptive?

Vertebrate 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	?

Predictive Model

Whole dataset can be used to predict the class label of the given vertebrate

Descriptive Model

It can be used as a descriptive model to help determine characteristics that define a vertebrate as a mammal

General Framework for Classification Model

Training Set

The model is created using a given a set of instances, known as the training set, which contains attribute values as well as class labels for each instance

Learning Algorithm

The systematic approach for learning a classification model given a training set is known as a learning algorithm.

- Rule Based Models
- Decision Tree based Models
- Probabilistics Models - Naive Bayes, Bayesian
- Generic Models - SVM, ANN
- Ensemble Classifier - Boosting, Bagging, Random Forests

General Framework for Classification Model (cont.)

Induction

The process of using a learning algorithm to build a classification model from the training data is known as induction

Classifier / Model

It is typically described in terms of a model

Deduction

This process of applying a classification model on unseen test instances to predict their class labels is known as deduction

Evaluation

Why Evaluate a Classification Model?

- To measure the model's performance on unseen data.
- To compare different models and select the best one.
- To understand the types of errors made by the model.
- To ensure reliability and robustness in real-world applications.

Confusion Matrix

A confusion matrix is a table used to evaluate the performance of a classification model in machine learning. It provides a detailed breakdown of the model's predictions compared to the actual labels, allowing for a clear understanding of where the model is performing well and where it is making errors.

For a binary classification problem, the confusion matrix is a 2x2 table with the following components:

	Predicted Positive	Predicted Negative
Actual Positive	TP	FN
Actual Negative	FP	TN

Confusion Matrix (cont.)

Benefits

- Provides a detailed error analysis
- Helps in computing various performance metrics

Weaknesses

- Does not account for class imbalance
- Can be misleading if only accuracy is considered

Accuracy (cont.)

Measures the proportion of correct predictions out of the total predictions

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- Measures overall correctness
- Can be misleading if classes are imbalanced

Example If TP = 50, TN = 40, FP = 5, FN = 5, then:

$$\text{Accuracy} = \frac{50 + 40}{50 + 40 + 5 + 5} = 90\%$$

Accuracy (cont.)

Consider a 2-class problem

- Number of Class 0 examples = 9990
- Number of Class 1 examples = 10

What is the accuracy if model predicts everything to be class 0

$$\textit{accuracy} = 9990/10000 = 99.9\%$$

Accuracy is misleading because model does not detect any class 1 example

Precision and Recall

Precision

Measures the proportion of correctly predicted positive instances out of all predicted positive instances.

$$\text{Precision} = \frac{TP}{TP + FP}$$

Recall (Sensitivity or True Positive Rate)

Measures the proportion of correctly predicted positive instances out of all actual positive instances.

$$\text{Recall} = \frac{TP}{TP + FN}$$

Precision and Recall

Interpretation

- Precision: How many predicted positives are actually correct?
- Recall: How many actual positives are correctly identified?

Example If TP = 50, FP = 10, FN = 20:

$$\text{Precision} = \frac{50}{50 + 10} = 0.83, \quad \text{Recall} = \frac{50}{50 + 20} = 0.71$$

F1-Score

The harmonic mean of precision and recall, providing a balance between the two.

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

- Harmonic mean of Precision and Recall.
- Balances both metrics, useful for imbalanced datasets.

Example

$$\text{F1-Score} = 2 \times \frac{0.83 \times 0.71}{0.83 + 0.71} = 0.77$$

Interpretation: Precision=0.83 and Recall = 0.71

Precision

- A precision of 0.83 means that 83% of the instances predicted as positive by the model are actually positive.
- This indicates that the model has a relatively low rate of false positives (FP). In other words, when the model predicts a positive outcome, it is correct 83% of the time.

Recall

- A recall of 0.71 means that the model correctly identifies 71% of all actual positive instances.
- This indicates that the model misses 29% of the actual positive instances (false negatives, FN). In other words, the model fails to capture some of the positive cases.

Trade-off Between Precision and Recall

Precision and recall often have an **inverse relationship**, meaning improving one may lead to a decrease in the other. This trade-off depends on the model's threshold for classifying positive instances:

- **High Precision, Lower Recall:** The model is conservative in predicting positives, resulting in fewer false positives but potentially missing some actual positives (higher false negatives).
- **High Recall, Lower Precision:** The model is more aggressive in predicting positives, capturing more actual positives but also increasing false positives.

Trade-off Between Precision and Recall (cont.)

The interpretation of precision and recall depends on the specific problem and the cost of errors:

High Precision is Important

- In scenarios where false positives are costly, high precision is desirable.
- **Example** Spam detection (incorrectly marking a non-spam email as spam is more costly than missing some spam emails).

Trade-off Between Precision and Recall (cont.)

The interpretation of precision and recall depends on the specific problem and the cost of errors:

High Recall is Important

- In scenarios where false negatives are costly, high recall is desirable.
- **Example** Medical diagnosis (missing a positive case, such as a disease, is more costly than false alarms).

Balanced Precision and Recall

In some cases, a balance between precision and recall is needed, which can be measured using the F1-score.
The F1-score for your model is:

$$F1{-}score = 2\times\frac{Precision \times Recall}{(Precision + Recall)} = 2\times(0.83\times0.71)/(0.83+0.71) = 0.76$$

This indicates a reasonably balanced performance.

Rule-Based Approach

Problem: Classifying Emails as Spam or Not Spam (Ham) Using Rule-Based Approach

We will define a simple rule-based classifier based on common spam email characteristics.

- Define the dataset
- Define the rules
- Implement the classifier
- Evaluate

Step 1: Define the Dataset

The dataset consists of email samples with features such as:

- Contains "buy now" (Boolean)
- Contains "free" (Boolean)
- Contains "limited offer" (Boolean)
- Contains "urgent" (Boolean)
- Number of words (Integer)
- Label: Spam or Ham (Non-Spam)

Step 2: Define Rule-Based Classifier

We'll classify an email as Spam if it matches any of these rules:

- If an email **contains** "buy now" AND "free", classify it as Spam.
- If an email **contains** "limited offer" AND "urgent", classify it as Spam.
- If an email **contains** any three of the spam words, classify it as Spam.
- If an email has **less than 40 words** AND contains any two spam words, classify it as Spam.
- Otherwise, classify it as Ham.

Step 3: Implementing the Rule-Based Classifier

It is mostly if else statements
Use any programming language to interprate those rules

Sample dataset

```
data = [
    {"buy_now": 1, "free": 1, "limited_offer": 0,
      "urgent": 1, "word_count": 50, "label": "Spam"},
    {"buy_now": 0, "free": 1, "limited_offer": 0,....
]
```

Classifier

```
def classify_email(email):
    spam_words = sum([email["buy_now"], email["free"],
         email["limited_offer"], email["urgent"]])

    if email["buy_now"] and email["free"]:
        return "Spam"
    if email["limited_offer"] and email["urgent"]:
        return "Spam"
    if spam_words >= 3:
        return "Spam"
    if email["word_count"] < 40 and spam_words >= 2:
        return "Spam"
    return "Ham"
```

Inference

```
import pandas as pd
df = pd.DataFrame(data)

df["predicted_label"] = df.apply(classify_email, axis=1)

# Evaluate
accuracy = (df["label"] == df["predicted_label"]).mean()
print("Rule-Based␣Classifier␣Accuracy:", accuracy)
print(df)
```

Step 4: Evaluate

The rule-based model is evaluated by checking how many labels it correctly predicts. Accuracy is computed as:

$$Accuracy = \frac{Correct\ Predictions}{Total\ Samples}$$

Example Problem

Classifying Patients as Diabetic or Non-Diabetic Using Rule-Based Approach
A patient is classified as Diabetic if they meet any of the following conditions:

- Blood Sugar Level 180 mg/dL = Diabetic
- BMI $\geq$ 30 AND Blood Sugar Level $\geq$ 140 mg/dL = Diabetic
- Age $\geq$ 50 AND Family History = 1 = Diabetic
- Age $\geq$ 40 AND Blood Sugar Level $\geq$ 150 mg/dL Diabetic
- Otherwise, classify as Non-Diabetic.

Example Problem

Classifier

```
def classify_patient(patient):
    if patient["blood_sugar"] >= 180:
        return "Diabetic"
    if patient["bmi"] >= 30 and patient["blood_sugar"] >= 140:
        return "Diabetic"
    if patient["age"] >= 50 and patient["family_history"] == 1:
        return "Diabetic"
    if patient["age"] >= 40 and patient["blood_sugar"] >= 150:
        return "Diabetic"
    return "Non-Diabetic"
```

Rule-Based Classification

Advantages of Rule-Based Classification

- Interpretable: The decision-making process is clear.
- Fast Execution: No need for training data.
- Works with Small Datasets: Unlike deep learning models, which require large labeled datasets

Disadvantages

- Limited Generalization: May not work well on unseen cases.
- Rule Engineering Required: Needs domain-specific rules.

Decison Tree

Decision Tree is a supervised learning algorithm used for classification tasks

It represents decisions and their possible consequences in a tree-like structure.

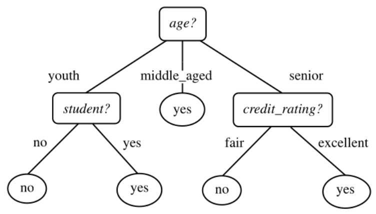
Each **internal node** represents a decision based on a feature, **branches** represent outcomes, and **leaves** represent final output

Classification by Decision Tree Induction

Class: **buys computer**

Each **internal (nonleaf) node** represents a test on an attribute.

Each leaf node represents a class (either buys *computer* = *yes* or *buyscomputer* = *no*)



Decision Tree (cont.)

How are decision trees used for classification?

Given a tuple, X, for which the associated class label is *unknown*, the attribute values of the tuple are tested against the decision tree.

A **path** is traced from the **root to a leaf node**, which holds the class prediction for that tuple.
Decision trees can easily be converted to classification rules.

Decision Tree (cont.)

Why are decision tree classifiers so popular?

- Easy to Understand and Interpret
- Handles numerical and categorical data
- Non-linear relationships - Unlike linear models, decision trees can capture complex, nonlinear patterns without needing feature engineering
- Minimal Assumptions - don't assume the data follows a specific distribution (e.g., normality), making them robust for messy, real-world datasets
- Foundation for Advanced Models - building blocks for powerful ensemble methods like Random Forests and Gradient Boosting (e.g., XGBoost), which amplify their strengths

Attribute Selection Measures

Decision Trees **divide data into subsets based on feature values**.

The best split is determined using a splitting criterion.

- Information gain
- Gain ratio
- Gini index

Entropy

Entropy (H) measures the uncertainty or impurity in a dataset.

- A dataset with **pure classes** (only one class) has **low entropy** (H = 0).
- A dataset with **equal distribution** of classes has **high entropy**.

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

where, p_i is the probability of class i in the dataset, m is distinct class

Entropy Interpretation

If all samples belong to the same class: $H = 0$ *(Pure dataset, no uncertainty, best case)*

If classes are evenly distributed (50-50 split in binary classification): $H = 1$ *(Maximum uncertainty, worst case)*

Entropy (cont.)

Dataset

Instances	Label
A	YES
B	YES
C	NO
D	NO

Class Probabilities

$p_{YES} = 2/4 = 0.5 \qquad p_{NO} = 2/4 = 0.5$

Entropy Calculation

$$H = -\sum p_i \log_2(p_i) = -(0.5 \times \log_2 0.5 + 0.5 \times log_2 0.5) = 1.0$$

Interpretation

- The dataset is highly impure
- We need to split the data to reduce entropy

Information Gain

Information Gain (IG) measures the reduction in entropy after a split. It helps decide which feature to split on.

$$IG = H(Parent) - \sum (\frac{|S_v|}{|S|} H(S_v))$$

Where,
$H(Parent)$ = Entropy of the original dataset
S_v = Subset after splitting by a feature
$|S_v|/|S|$ = Proportion fo subset in dataset

Use Case

A self-driving car needs to decide whether to turn left or right at an intersection based on the surrounding road conditions. The decision should consider various features such as

- traffic signals
- pedestrian movement
- oncoming traffic
- lane blockage

Use Case (cont.)- Do youtself

The dataset used for decision-making are as follows. Make a decision tree.

Traffic Signal	Pedestrian Nearby	Oncoming Traffic	Left Lane Blocked	Right Lane Blocked	Decision
Green	No	No	No	No	Left
Green	No	Yes	No	Yes	Left
Red	No	No	No	No	No Turn
Green	Yes	No	No	Yes	Left
Green	No	Yes	Yes	No	Right
Green	Yes	Yes	No	No	Right
Green	Yes	No	No	No	Left
Green	No	Yes	Yes	Yes	No Turn
Red	Yes	No	No	No	No Turn
Green	No	No	Yes	No	Right
Green	No	No	No	Yes	Left
Green	Yes	Yes	Yes	No	Right

Table: Decision Dataset for Turning Left or Right

Where Information Gain fails?

The information gain measure is biased toward tests with many outcomes. That is, it prefers to select attributes having a large number of values.

For example, consider an attribute that **acts as a unique identifier**, such as **product ID**. A split on product ID would result in a large number of partitions (as many as there are values), each one containing just one tuple.

Because **each partition is pure**, the information required to classify data set D based on this partitioning would be $IG_{productID}(D) = 0$.

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

Gain Ratio

Attributes with many distinct values (e.g., unique IDs) may have high Information Gain but are not meaningful for decision-making.

Gain Ratio **penalizes splits that produce many small subsets**, making it more balance

It applies a normalization to information gain using a "split information" value defined analogously with $IG(D)$ as

$$SplitInfo(SI) = -\sum \frac{|S_i|}{|S|}\log_2\left(\frac{|S_i|}{|S|}\right)$$

Where, S is total dataset and S_i represent the subset created by splitting on an attribute

Gain Ratio (cont.)

Split Information (SI) is a normalization factor that accounts for how broadly an attribute divides the dataset.

$$GainRatio = \frac{\text{Information Gain}}{\text{Split Information}}$$

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

C4.5 Algorithm selects the attribute with the highest Gain Ratio rather than just Information Gain.

Gini Index

The **Gini Index** (or **Gini Impurity**) measures how often a randomly chosen element would be incorrectly classified.

It is used in **CART (Classification and Regression Trees)** for decision tree splits.

$$Gini(S) = 1 - \sum p_i^2$$

Where, p_i is the probability of class i in the dataset S.

Gini Index(cont.)

Suppose we have a dataset with three classes:

- Left Turn: 5 samples
- Right Turn: 4 samples
- No Turn: 3 samples

$$Gini(S) = 1 - \left(\left(\frac{5}{12}\right)^2 + \left(\frac{4}{12}\right)^2 + \left(\frac{3}{12}\right)^2\right)$$
$$Gini(S) = 1 - (0.1736 + 0.1111 + 0.0625) = 0.6528$$

Gini Index(cont.)

Gini Index for Splitting

To determine the best split, we calculate the weighted Gini Index for each attribute and choose the one with the lowest value.

$$Gini_{split} = \sum \frac{|S_i|}{|S|} Gini(S_i)$$

Where, S_i are the subsets after splitting

Gini vs. Entropy

- **Gini Index:** Faster computation (no logarithms), used in CART.
- **Entropy (Information Gain):** Used in ID3, C4.5, requires logarithm calculation.
- **Key Difference:** Gini prefers balanced splits, while Information Gain may favor attributes with many unique values.

Decision Tree Induction

Algorithms

- Hunt's Algorithm
- ID3
- C4.5
- ID3
- SLIQ
- SPRINT

Read Yourself

- Go through the algorithms
- Note their weakness and strengths

Overfitting in Decision Tree

The problem of overfitting is considered when the algorithm continues to go deeper and deeper to reduce the training-set error but results with an increased test-set error.

So, accuracy of prediction for our model goes down.

It generally happens when we build many branches due to outliers and irregularities in data.

Tree Pruning

When a decision tree is built, many of the branches will reflect anomalies in the training data due to noise or outliers. Tree pruning methods address this problem of overfitting the data.

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

Tree Pruning (cont.)

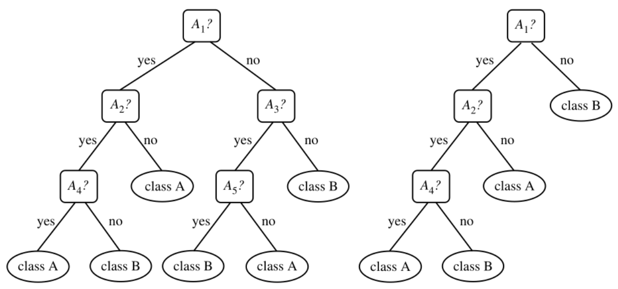


Figure: An unpruned decision tree and a pruned version of it

Tree Pruning (cont.)

How does tree pruning work?

There are two common approaches to tree pruning:

1. prepruning
2. postpruning

Prepruning

In the prepruning approach, a tree is "pruned" by halting its construction early (e.g., by deciding not to further split or partition the subset of training tuples at a given node)

Upon halting, the node becomes a leaf. The leaf may hold the most frequent class among the subset tuples or the probability distribution of those tuples.

If partitioning the tuples at a node would result in a split that falls below a prespecified threshold, then further partitioning of the given subset is halted. Choosing an appropriate threshold is challenging

- High thresholds could result in oversimplified trees
- Low thresholds could result in very little simplification.

Postpruning

The more common approach is postpruning, which removes subtrees from a "fully grown" tree.

A subtree at a given node is pruned by removing its branches and replacing it with a leaf. The leaf is labeled with the most frequent class among the subtree being replaced.

For example, notice the subtree at node "A3?" in the unpruned tree of previous figure. Suppose that the most common class within this subtree is class B. In the pruned version of the tree, the subtree in question is pruned by replacing it with the leaf "class B".

Other Pruning:

- Cost complexity - used by CART,
- Pessimistic pruning - used by C4.5

Bayesian classifiers

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

Studies comparing classification algorithms have found a simple Bayesian classifier known as the naive Bayesian classifier to be comparable in performance with decision tree and selected neural network classifiers.

Bayesian classifiers (cont.)

Bayesian classifiers have also exhibited high accuracy and speed when applied to large databases.

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

It is made to simplify the computations involved and, in this sense, is considered "naïve."

Bayes' Theorem

Let X be a data tuple. In Bayesian terms, X is considered **"evidence**. As usual, it is described by measurements made on a set of n attributes.

Let H be some hypothesis, such as that the data tuple X belongs to a specified class C.

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

Bayes' Theorem (cont.)

Posterior probability

$P(H|X)$ is a posteriori probability of H conditioned on X.

For example, suppose our world of data tuples is confined to customers described by the attributes *age* and *income*, respectively, and that X is a 35-year-old customer with an income of $40,000.

Suppose that H is the hypothesis that our customer will buy a computer. Then $P(H|X)$ reflects the probability that customer X will buy a computer given that we know the customer's age and income.

Bayes' Theorem (cont.)

Prior probability

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

The posterior probability, $P(H|X)$, is based on more information (e.g., customer information) than the prior probability, $P(H)$, which is **independent of X**.

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

Bayes' Theorem (cont.)

How are these probabilities estimated

$P(H)$, $P(X|H)$, and $P(X)$ may be estimated from the given data.

Bayes' theorem is useful in that it provides a way of calculating the posterior probability, $P(H|X)$, from $P(H)$, $P(X|H)$, and $P(X)$.

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

Naïve Bayesian Classification

Features

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

Classes

Suppose that there are m classes, $C_1, C_2, ..., C_m$

Naïve Bayesian Classification (cont.)

Given a tuple, X, the **classifier will predict that X belongs to the class having the highest posterior probability**, conditioned on X.

That is, the naïve Bayesian classifier predicts that tuple X belongs to the class C_i if and only if

$$P(C_i|X) > P(C_j|X) \qquad for 1 \leq j \leq m, j \neq i$$

Thus we maximize $P(C_i|X)$.

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

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

Naïve Bayesian Classification (cont.)

As $P(X)$ is constant for all classes, only $P(X|C_i)P(C_i)$ need be maximized.

If the class prior probabilities are not known, then it is commonly assumed that the classes are equally likely, that is,
$P(C_1) = P(C_2) = \cdots = P(C_m)$, and we would therefore maximize $P(X|C_i)$.
Otherwise, we maximize $P(X|C_i)P(C_i)$.

Note that the class prior probabilities may be estimated by

$$P(C_i) = \frac{|C_{i,D}|}{|D|}$$

where $|C_{i,D}|$ is the number of training tuples of class C_i in D

Naïve Bayesian Classification - naive assumption

Given data sets with many attributes, it would be extremely computationally expensive to compute $P(X|C_i)$. In order to reduce computation in evaluating $P(X|C_i)$, the **naive assumption of class conditional independence** is made.

This presumes that the values of the attributes are conditionally independent of one another, given the class label of the tuple (i.e., that there are no dependence relationships among the attributes). Thus,

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

We can easily estimate the probabilities $P(x_1|C_i), P(x_2|C_i), ..., P(x_n|C_i)$ from the training tuples. x_k refers to the value of attribute A_k for tuple X.

Naïve Bayesian Classification - data types

For each attribute, we look at whether the attribute is **categorical** or **continuous-valued**.
For instance, to compute $P(X|C_i)$, we consider the following:
If A_k is categorical, then $P(x_k|C_i)$ is the number of tuples of class C_i in D having the value x_k for A_k , divided by $|C_{i,D}|$ - the number of tuples of class C_i in D.
If A_k is continuous-valued, then we need to do a bit more work, but the calculation is pretty straightforward.
A continuous-valued attribute is typically **assumed to have a Gaussian distribution** with a mean μ and standard deviation σ, defined by

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

So that for classification,

$$P(x_k|C_i) = g(x_k, \mu_{Ci}, \sigma_{Ci})$$

Naïve Bayesian Classification (cont.)

A continuous-valued attribute is typically **assumed to have a Gaussian distribution** with a mean μ and standard deviation σ, defined by

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

So that,

$$P(x_k|C_i) = g(x_k, \mu_{Ci}, \sigma_{Ci})$$

We need to compute μ_{Ci} and σ_{Ci} , which are the mean (i.e., average) and standard deviation respectively, of the values of attribute A_k for training tuples of class C_i

Naïve Bayesian Classification (cont.)

In order to predict the class label of X, $P(X|C_i)P(C_i)$ is evaluated for each class C_i .

The classifier predicts that the class label of tuple X is the class C_i if and only if

$$P(X|C_i)P(C_i) > P(X|C_j)P(C_j) \qquad for 1 \le j \le m, j \ne i$$

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

Read yourself
Book: Example 6.4 (Page No: 313)

Classification - Neural Network

Unit 3: Classification

Artificial Neural Network

–

School of Mathematical Sciences
Institute of Science and Technology (IoST), TU

Feb, 2025

Table of contents

Neural Networks

What is neural network?

A neural network is a machine learning model inspired by the human brain, designed to recognize patterns and make predictions.

Two Views

- A way to **simulate biological learning** by simulating the nervous system.
- A way to increase the power of known models in machine learning by stacking them in careful ways as **computational graphs**
  - The number of nodes in the computational graph controls learning capacity with increasing data
  - The specific architecture of the computational graph incorporates domain-specific insights (e.g., images, speech).
  - The success of deep computational graphs has led to the coining of the term **"deep learning"**.

History of ANN

Groundwork

1943: *Warren McCulloch* and *Walter Pitts* proposed the first mathematical model of a neuron, the **"McCulloch-Pitts neuron"** which could perform simple logical operations.

Publication: A Logical Calculus of the ideas Imminent in Nervous Activity [1]

This laid the theoretical groundwork for artificial neural networks (ANNs).

Reference
- https://www.cse.chalmers.se/ coquand/AUTOMATA/mcp.pdf
- McCulloch-Pitts Neuron — Mankind's First Mathematical Model Of A Biological Neuron

History of ANN (cont.)

1958: *Frank Rosenblatt* introduced the **Perceptron**, a single-layer neural network capable of binary classification.

It was the first model implemented on a computer and sparked interest in machine learning. However, it could only solve linearly separable problems.

- Was implemented using a large piece of hardware.

- Generated great excitement but failed to live up to inflated expectation

AI Winter 1960s-1970s

- Minsky and Papert's Book "Perceptrons" (1969) showed that the perceptron only had limited expressive power
- Minsky is often blamed for setting back the field (fairly or unfairly)

Were Minsky/Papert justified in their pessimism?

History of ANN (cont.)

Did We Really Not Know How to Train Multiple Units?

It depends on who you ask

- AI researchers didn't know (and didn't believe it possible).
- Training computational graphs with dynamic programming had already been done in control theory (1960s)

History of ANN (cont.)

First attempt of backpropagation

Paul Werbos proposed backpropagation in his 1974 thesis (and was promptly ignored - formal publication was delayed)

Werbos (2006): "In the early 1970s, I did in fact visit Minsky at MIT. I proposed that we do a joint paper showing that MLPs can in fact overcome the earlier problems if (1) the neuron model is slightly modified to be differentiable; and (2) the training is done in a way that uses the reverse method, which we now call backpropagation in the ANN field. But Minsky was not interested. In fact, no one at MIT or Harvard or any place I could find was interested at the time."

History of ANN (cont.)

Seventies/Eighties - Era of logic

It was the era or work on logic and reasoning (discrete mathematics)

Work on continuous optimization had few believers

- Researchers like Hinton were certainly not from the main-stream
- This view has been completely reversed today
- The early favorites have little to show in spite of the effort

History of ANN (cont.)

Backpropagation: The Second Coming

Rumelhart, Hinton, and Williams wrote two papers on backpropagation in 1986 (independent from prior work)

1986: *David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams* popularized **backpropagation**, an algorithm for training multi-layer neural networks.

- This breakthrough allowed networks to learn complex, non-linear patterns, overcoming earlier limitations
- Networks with hidden layers (multilayer perceptrons, or MLPs) emerged, revitalizing interest in the field

History of ANN (cont.)

The Nineties

Acceptance of backpropagation encouraged more research in multilayer networks.

By the year 2000, most of the modern architectures had already been set up in some form.

- They just didn't work very well!
- The winter continued after a brief period of excitement

It was the era of the support vector machine.

The new view: SVM was the panacea (at least for supervised learning)

History of ANN (cont.)

The Deep Learning Revolution (2010s-Present)

2012: AlexNet, a deep convolutional neural network (CNN) developed by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, won the ImageNet competition by a wide margin. This demonstrated the power of deep learning, fueled by large datasets and GPU acceleration.

2010s-2020s: Innovations like Recurrent Neural Networks (RNNs), Generative Adversarial Networks (GANs), and Transformers (e.g., BERT, GPT) expanded neural networks into natural language processing (**NLP**), image generation, and beyond.

Today, neural networks underpin technologies like self-driving cars, virtual assistants, and advanced scientific research (e.g., protein folding with AlphaFold).

History of ANN (cont.)

2024 Nobel Prizes

The 2024 Nobel Prizes marked a significant moment for artificial intelligence (AI)

- Recognition of Foundational AI Contributions: The Nobel Prize in Physics was awarded to **John Hopfield and Geoffrey Hinton** for their pioneering work on artificial neural networks, which are the backbone of modern machine learning.
- AI as a Tool for Scientific Breakthroughs: The Nobel Prize in Chemistry went to **David Baker, Demis Hassabis, and John Jumper**, with **Hassabis and Jumper** recognized for developing AlphaFold, an AI system that solved the 50-year-old problem of protein structure prediction.

By awarding Nobels in Physics and Chemistry to AI-related work, the scientific community signaled that AI is **no longer just a computational tool** but a fundamental driver of progress across disciplines

How it All Started

The Biological Inspiration

Neural networks were originally designed to simulate the learning process in biological organisms.

The human nervous system contains cells called **neurons**.

The neurons are connected to one another with the use of synapses.

- The **strengths** of synaptic connections often **change in response to external stimuli**.
- This change causes learning in living organisms

Neural Network - The Biological Inspiration

Figure: (left) Biological Neuron, (right) ANN

- Neural networks contain computation units ⇒ Neurons
- The computational units are connected to one anotherthrough weights ⇒ Strengths of synaptic connections in biological organisms

Neural Network - Learning

Living organisms

- Synaptic weights change in response to external stimuli
- An unpleasant experience will change the synaptic weights of an organism, which will train the organism to behave differently

Artificial neural networks

- The weights are learned with the use of **training data**, which are input-output pairs (e.g., images and their labels)
- An error made in predicting the label of an image is the unpleasant "stimulus" that changes the weights of the neural network
- When trained over many images, the network learns to classify images correctly

Is ANN/DNN represent brain?

No one exactly knows how the brain works

The functions computed in a neural network are very different from those in the brain

Nevertheless, there are several examples, where the principles of neuroscience have been successfully applied in designing neural networks.

Convolutional neural networks are based on architectural principles drawn from the cat's visual cortex

Machine Learning versus Deep Learning

Machine Learning

For smaller data sets, traditional machine learning methods often provide slightly better performance
Traditional models often provide more choices, interpretable insights, and ways to handcraft features

Deep Learning

For larger data sets, deep learning methods tend to dominate

Perceptron

A Perceptron is one of the simplest types of artificial neural networks and serves as the foundational building block for more complex neural architectures.

A perceptron is a single-layer neural network that takes multiple binary or real-valued inputs, applies weights to them, sums them up, and passes the result through an activation function to produce an output.

Perceptron (cont.)

How do perceptron works?

A perceptron takes several binary inputs, $x_1, x_2, \ldots,$ and produces a single binary output:

In the example shown the perceptron has three **inputs**, x_1, x_2, x_3.
In general it could have more or fewer inputs.

Rosenblatt proposed a simple rule to compute the output.

He introduced **weights**, $w_1, w_2, ...,$ real numbers R expressing the importance of the respective inputs to the output.

Perceptron (cont.)

The neuron's output, 0 or 1, is determined by whether the weighted sum $\sum_j w_j x_j$ is less than or greater than some threshold value.

Just like the weights, the threshold is a real number which is a parameter of the neuron.

To put it in more precise algebraic terms:

$$output = \begin{cases} 1, & if \sum_j w_j x_j \leq threshold \\ 0, & if \sum_j w_j x_j > threshold \end{cases}$$

That's all there is to how a perceptron works!

Perceptron (cont.)

A way you can think about the perceptron is that it's a **device that makes decisions by weighing up evidence**.

Example

Suppose the weekend is coming up, and you've heard that there's going to be a cheese festival in your city. You like cheese, and are trying to decide whether or not to go to the festival. You might make your decision by weighing up three factors:

1. Is the weather good?
2. Does your boyfriend or girlfriend want to accompany you?
3. Is the festival near public transit? (You don't own a car).

Perceptron (cont.)

Example

We can represent these three factors by corresponding binary variables x_1, x_2 and x_3. For instance, we'd have

- $x_1 = 1$ if the weather is good, and $x_1 = 0$ if the weather is bad
- $x_2 = 1$ if your boyfriend or girlfriend wants to go
- $x_2 = 0$ if not. And similarly again for x_3 and public transit

How you decide?

Now, suppose you absolutely adore cheese so much so that you're happy to go to the festival even if your boyfriend or girlfriend is uninterested and the festival is hard to get to. But perhaps you really want check the weather, and there's no way you'd go to the festival if the weather is bad.

Perceptron (cont.)

You can use perceptrons to model this kind of decision-making.

- One way to do this is to choose a weight $w_1 = 6$ for the weather, and $w_2 = 2$ and $w_3 = 2$ for the other conditions.
- The larger value of w_1 indicates that the weather matters a lot to you, much more than whether your boyfriend or girlfriend joins you, or the nearness of public transit.

Finally, suppose you choose a **threshold of 5** for the perceptron.

With these choices, the perceptron implements the desired decision-making model, outputting 1 whenever the weather is good, and 0 whenever the weather is bad.

It makes no difference to the output whether your boyfriend or girlfriend wants to go, or whether public transit is nearby.

Perceptron (cont.)

Threshold = 3

The perceptron would decide that you should go to the festival whenever the weather was good or when both the festival was near public transit and your boyfriend or girlfriend was willing to join you.

In other words, it'd be a different model of decision-making.

$$output = \begin{cases} 1, & if \sum_j w_j x_j \leq threshold \\ 0, & if \sum_j w_j x_j > threshold \end{cases}$$

Dropping the threshold means you're more willing to go to the festival.

Multilayer Perceptron

The perceptron isn't a complete model of human decision-making!

But what the example illustrates is how a perceptron can weight up different kinds of evidence in order to make decisions.

And it should seem plausible that a complex network of perceptrons could make quite subtle decisions.

Multilayer Perceptron (cont.)

In this network, the first column of perceptrons – what we'll call the *first layer* of perceptrons – is making three very simple decisions, by weighing the input evidence.

What about the perceptrons in the second layer?
Each of those perceptrons is making a decision by weighing up the results from the first layer of decision-making.In this way a perceptron in the **second layer** can make a decision at a more complex and more **abstract level than perceptrons in the first layer**.

And **even more complex decisions can be made by the perceptron in the third layer**. In this way, a many-layer network of perceptrons can engage in sophisticated decision making.

Multilayer Perceptron (cont.)

Does perceptron has multiple output?

By defination, the perceptron has only one output. In must of the figures they look like they have multiple outputs.

In fact they are still single output.

The multiple output arrows are merely a useful way of indicating that the output from a perceptron is being used as input to several other perceptrons.

Generalizing the perceptron

Let's observe the output of the perceptron. Try to identify the mathematical limitation ...

$$output = \begin{cases} 1, & if \sum_j w_j x_j \leq threshold \\ 0, & if \sum_j w_j x_j > threshold \end{cases}$$

Can we improve this?

Generalizing the perceptron (cont.)

Let's simplify the way we describe perceptrons.
The condition $\sum_j w_j x_j > threshold$ threshold is difficult to use, and we can make two *notational changes* to simplify it.

- The first change is to write $\sum_j w_j x_j$ as a dot product, $w \cdot x = \sum_j w_j x_j$, where w and x are vectors whose components are the weights and inputs, respectively.
- The second change is to move the threshold to the other side of the inequality, and to replace it by what's known as the perceptron's *bias*, $b \equiv -threshold$.

Using the bias instead of the threshold, the perceptron rule can be rewritten:

$$\text{output} = \begin{cases} 0 & \text{if} \quad w \cdot x + b \leq 0 \\ 1 & \text{if} \quad w \cdot x + b > 0 \end{cases} \tag{2}$$

Generalizing the perceptron (cont.)

Bias

We can think of the bias as a measure of how easy it is to get the perceptron to output 1.

Or to put it in more biological terms, the bias is a measure of how easy it is to get the perceptron to *fire*.

- For a perceptron with a really **big bias**, it's extremely easy for the perceptron to output 1.
- But if the bias is **very negative**, then it's difficult for the perceptron to output 1.

Computational universality of perceptrons

Another way perceptrons can be used is to **compute the elementary logical functions** we usually think of as underlying computation, functions such as AND, OR, and NAND.

For example, suppose we have a perceptron with two inputs, each with weight -2, and an overall bias of 3. Here's our perceptron:

Then we see that input 00 produces output 1, since $(-2) \times +(-2) \times +3 = 3$ is positive.

Similar calculations show that the inputs 01 and 10 produce output 1. But the input 11 produces output 0, since $(-2) \times +(-2) \times +3 = -1$ is negative.

And so our perceptron implements a NAND gate!

Computational universality of perceptrons (cont.)

The computational universality of perceptrons is simultaneously reassuring and disappointing.

- It's **reassuring** because it tells us that networks of perceptrons can be as powerful as any other computing device.
- But it's also **disappointing**, because it makes it seem as though perceptrons are merely a new type of NAND gate. That's hardly big news!

Computational universality of perceptrons (cont.)

What make neural network different to conventional logic gates?

We can devise **learning algorithms** which can **automatically tune the weights and biases** of a network of artificial neurons.

This tuning happens in response to external stimuli, without direct intervention by a programmer.

These learning algorithms enable us to use artificial neurons in a way which is radically different to conventional logic gates.

Instead of explicitly laying out a circuit of NAND and other gates, our neural networks can simply learn to solve problems, sometimes problems where it would be extremely difficult to directly design a conventional circuit.

What is Learning?

Learning algorithms sound terrific. But how can we devise such algorithms for a neural network?

For example, the inputs to the network might be the raw pixel data from a scanned, handwritten image of a digit. And we'd like the network to learn weights and biases so that the output from the network correctly classifies the digit.
To see how learning might work, suppose we make a small change in some weight (or bias) in the network. What we'd like is for this small change in weight to cause only a small corresponding change in the output from the network. As we'll see in a moment, this property will make learning possible.

Learning (cont.)

If it were true that a small change in a weight (or bias) causes only a small change in output, then we could use this fact to modify the weights and biases to get our network to behave more in the manner we want.

For example, suppose the network was mistakenly classifying an image as an "8" when it should be a "9".
We could figure out how to make a small change in the weights and biases so the network gets a little closer to classifying the image as a "9".
And then we'd repeat this, changing the weights and biases over and over to produce better and better output. The network would be learning.

Is this possible with perceptron?

Learning (cont.)

A small change in the weights or bias of any single perceptron in the network can sometimes cause the output of that perceptron to completely flip, say from 0 to 1.

That flip may then cause the behaviour of the rest of the network to completely change in some very complicated way.

What is the solution?
We can overcome this problem by introducing a new type of artificial neuron called a **sigmoid neuron**. Sigmoid neurons are similar to perceptrons, but modified so that small changes in their weight and bias cause only a small change in their output. **That's the crucial fact which will allow a network of sigmoid neurons to learn**.

Sigmoid neurons

We'll depict sigmoid neurons in the same way we depicted perceptrons:

The sigmoid neuron has inputs, $x_1, x_2, \ldots$.
But instead of being just 0 or 1, these inputs can also take on any values between 0 and 1.
Also just like a perceptron, the sigmoid neuron has weights for each input, $w_1, w_2, \ldots$, and an overall bias, b. But the output is not 0 or 1. Instead, it's $\sigma(wx + b)$, where σ is called the sigmoid function

Sigmoid neuron (cont.)

$$\sigma(z) \equiv \frac{1}{1+e^{-z}}. \tag{3}$$

To put it all a little more explicitly, the output of a sigmoid neuron with inputs x_1,x_2,..., weights $w_1, w_2, \ldots$, and bias b is

$$\frac{1}{1+\exp\left(-\sum_j w_j x_j - b\right)}. \tag{4}$$

At first sight, sigmoid neurons appear very different to perceptrons.

In fact, there are many similarities between perceptrons and sigmoid neurons, and the algebraic form of the sigmoid function turns out to be more of a technical detail than a true barrier to understanding.

Sigmoid neuron(cont.)

When sigmoid act like perceptron?

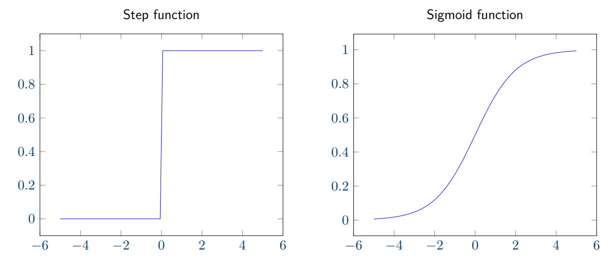
Suppose $z \equiv w \cdot x + b$ is a large positive number. Then $e^{-z} \approx 0$ and so $\sigma(z) \approx 1$.

Suppose on the other hand that $z = w \cdot x + b$ is very negative. Then $e^{-z} \to \infty$, and $\sigma(z) \approx 0$. So when $z = w \cdot x + b$ is very negative, the behaviour of a sigmoid neuron also closely approximates a perceptron.

It's only when $w \cdot x + b$ is of modest size that there's much deviation from the perceptron model.

Sigmoid vs. Perceptron

This shape is a smoothed out version of a step function:



Learning on Sigmoid

Why the smoothness of σ matters?

The smoothness of σ means that small changes Δw_j in the weights and Δb in the bias will produce a small change Δoutput in the output from the neuron.

How can we compute Δoutput?
Solution: Use calculus $\Rightarrow$ Δoutput is well approximated by

$$\Delta \text{output} \approx \sum_j \frac{\partial \text{output}}{\partial w_j} \Delta w_j + \frac{\partial \text{output}}{\partial b} \Delta b \qquad (5)$$

Where, the sum is over all the weights, w_j, and $\partial \text{output}/\partial w_j$ and $\partial \text{output}/\partial b$ denote partial derivatives of the output with respect to w_j and b, respectively.

Learning on Sigmoid

How should we interpret the output from a sigmoid neuron?
Suppose we want the output from the network to indicate either **"the input image is a 9"** or **"the input image is not a 9"**.
Obviously, it'd be easiest to do this if the output was a 0 or a 1, as in a perceptron.

But in practice we can **set up a convention** to deal with this
For example, by deciding to interpret any output of at least 0.5 as indicating a "9", and any output less than 0.5 as indicating "not a 9".

Multilayer Neural Network

Suppose we have the network:

The leftmost layer in this network is called the input layer, and the neurons within the layer are called *input neurons*.

The rightmost or *output* layer contains the *output neurons*, or, as in this case, a single output neuron.

The middle layer is called a *hidden layer*, since the neurons in this layer are neither inputs nor outputs.

The network above has just a single hidden layer, but some networks have multiple hidden layers.

Multilayer Neural Network

For example, the following four-layer network has two hidden layers:

Somewhat confusingly, and for historical reasons, such multiple layer networks are sometimes called *multilayer perceptrons* or *MLPs*, despite being made up of sigmoid neurons, not perceptrons.

Designing input and output layer

How to design input and output layer?
The design of the input and output layers in a network is often straightforward.

For example, suppose we're trying to determine whether a handwritten image depicts a "9" or not.

A natural way to design the network is to encode the intensities of the image pixels into the input neurons. If the image is a 64 by 64 greyscale image, then we'd have $4,096 = 64 \times 64$ input neurons, with the intensities scaled appropriately between 0 and 1.

The output layer will contain just a single neuron, with output values of less than 0.5 indicating "input image is not a 9", and values greater than 0.5 indicating "input image is a 9".

Design of Hidden Layers

While the design of the input and output layers of a neural network is often straightforward, there can be quite an art to the design of the hidden layers.

In particular, **it's not possible to sum up the design process for the hidden layers with a few simple rules of thumb**.

Instead, neural networks researchers have developed many design heuristics for the hidden layers, which help people get the behaviour they want out of their nets.

For example, such heuristics can be used to help determine how to trade off the number of hidden layers against the time required to train the network.

Feedforware Neural Network

Up to now, we've been discussing neural networks where the output from one layer is used as input to the next layer. Such networks are called ***feedforward* neural networks**.

This means there are no loops in the network – information is always fed forward, never fed back. If we did have loops, we'd end up with situations where the input to the σ function depended on the output. That'd be hard to make sense of, and so we don't allow such loops.

Recurrent Neural Network

There are models of artificial neural networks in which feedback loops are possible. These models are called **recurrent neural networks**.

The idea in these models is to have neurons which fire for some limited duration of time, before becoming quiescent. That firing can stimulate other neurons, which may fire a little while later, also for a limited duration.

That causes still more neurons to fire, and so over time we get a cascade of neurons firing.

Loops don't cause problems in such a model, since a neuron's output only affects its input at some later time, not instantaneously.

Recurrent Neural Network

Recurrent neural nets have been less influential than feedforward networks, in part because the learning algorithms for recurrent nets are (at least to date) less powerful.

But recurrent networks are still extremely interesting. They're much closer in spirit to how our brains work than feedforward networks. And it's possible that recurrent networks can solve important problems which can only be solved with great difficulty by feedforward networks.

A ANN for classification

How OCR works?

We can split the problem of recognizing handwritten digits into two sub-problems. - **segmentation, recognition**

For example, we'd like to break the image into six separate images.

$\Rightarrow$

We humans solve this *segmentation problem* with ease, but it's challenging for a computer program to correctly break up the image. Once the image has been segmented, the program then needs to classify each individual digit.

We'll focus on writing a program to solve the second problem, that is, classifying individual digits.

ANN for Digit Recognition

To recognize individual digits we will use a three-layer neural network:

ANN for Digit Recognition (cont.)

The **input layer** of the network contains neurons encoding the values of the input pixels.

Suppose, our training data for the network will consist of many 28 by 28 pixel images of scanned handwritten digits, and so the input layer contains $784 = 28 \times 28$ neurons.

For simplicity We've omitted most of the 784 input neurons in the diagram.

The input pixels are greyscale, with a value of 0.0 representing white, a value of 1.0 representing black, and in between values representing gradually darkening shades of grey.

ANN for Digit Recognition (cont.)

The second layer of the network is a **hidden layer**.

We denote the number of neurons in this hidden layer by n, and we can experiment with different values for n. The example shown illustrates a small hidden layer, containing just $n = 15$ neurons.

ANN for Digit Recognition (cont.)

The **output layer** of the network contains 10 neurons.

If the first neuron fires, i.e., has an output ≈ 1, then that will indicate that the network thinks the digit is a 0.

If the second neuron fires then that will indicate that the network thinks the digit is a 1. And so on.

A little more precisely, we number the output neurons from 0 through 9, and figure out which neuron has the highest activation value. If that neuron is, say, neuron number 6, then our network will guess that the input digit was a 6. And so on for the other output neurons.

ANN for Digit Recognition (cont.)

After all, the goal of the network is to tell us which digit $(0, 1, 2, \ldots, 9)$ corresponds to the input image. **Four neurons are enough to encode the answer**

A seemingly natural way of doing that is to use just 4 output neurons, treating each neuron as taking on a binary value, depending on whether the neuron's output is closer to 0 or to 1.

Four neurons are enough to encode the answer, since $2^4 = 16$ is more than the 10 possible values for the input digit.

Why should our network use 10 neurons instead? Isn't that inefficient?

Dataset

How can it learn to recognize digits?

The first thing we'll need is a data set to learn from – a so-called **training data set**.

We'll use the MNIST data set, which contains tens of thousands of scanned images of handwritten digits, together with their correct classifications. MNIST's name comes from the fact that it is a modified subset of two data sets collected by NIST, the United States' National Institute of Standards and Technology.

These are 28 by 28 greyscale images.

Dataset (cont.)

We'll use the notation x to denote a training input.

It'll be convenient to regard each training input x as a $28 \times 28 = 784$-dimensional vector. Each entry in the vector represents the grey value for a single pixel in the image.

We'll denote the corresponding desired output by $y = y(x)$, where y is a 10-dimensional vector.

For example, if a particular training image, x, depicts a 6, then $y(x) = (0, 0, 0, 0, 0, 0, 1, 0, 0, 0)^T$ is the desired output from the network.

Learning - Cost/Loss/Objective function

What we'd like is an algorithm which lets us find weights and biases so that the output from the network approximates $y(x)$ for all training inputs x.To quantify how well we're achieving this goal we define a cost function

$$C(w, b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2 \tag{1}$$

Where, w denotes the collection of all weights in the network, b all the biases, n is the total number of training inputs, a is the vector of outputs from the network when x is input, and the sum($\sum$) is over all training inputs x.C the *quadratic* cost function. It's also known as the *mean squared error*(MSE).

The output a depends on x, w and b, but to keep the notation simple we haven't explicitly indicated this dependence.

Learning - Cost function (cont.)

$C(w, b)$ is non-negative, since every term in the sum is non-negative.

$C(w, b)$ becomes small, i.e., $C(w, b) \approx 0$, precisely when $y(x)$ is approximately equal to the output, a, for all training inputs, x.

Generalizing, our training algorithm has done a good job if it can find weights and biases so that $C(w, b) \approx 0$.

By contrast, it's not doing so well when $C(w, b)$ is large – that would mean that $y(x)$ is not close to the output a for a large number of inputs.

Learning - Cost function (cont.)

Objective

So the aim of our training algorithm will be to minimize the cost $C(w, b)$ as a function of the weights and biases.

In other words, we want to find a set of weights and biases which make the cost as small as possible. We'll do that using an algorithm known as **gradient descent**.

Gradient Descent

Why introduce the quadratic cost? After all, aren't we primarily interested in the number of images correctly classified by the network? Why not try to maximize that number directly, rather than minimizing a proxy measure like the quadratic cost?

The problem with that is that the number of images correctly classified is not a smooth function of the weights and biases in the network. For the most part, making small changes to the weights and biases won't cause any change at all in the number of training images classified correctly.

That makes it difficult to figure out how to change the weights and biases to get improved performance.

If we instead use a smooth cost function like the quadratic cost it turns out to be easy to figure out how to make small changes in the weights and biases so as to get an improvement in the cost. That's why we focus first on minimizing the quadratic cost, and only after that will we examine the classification accuracy.

Gradient Descent (cont.)

Let's suppose we're trying to minimize $C(v)$. Where v could be any real-valued function of many variables, $v = v_1, v_2, \ldots$.

Note that we've replaced the w and b notation by v to emphasize that this could be any function – we're not specifically thinking in the neural networks context any more.

To minimize $C(v)$ it helps to imagine C as a function of just two variables, which we'll call v_1 and v_2. Our objective is to find where C achieves its global minimum.

For few variables the solution can be computed using derivtives. But it'll turn into a nightmare when we have many more variables.

Gradient Descent (cont.)

We'd randomly choose a starting point for an (imaginary) ball, and then simulate the motion of the ball as it rolled down to the bottom of the valley.

We could do this simulation simply by computing derivatives (and perhaps some second derivatives) of C – those derivatives would tell us everything we need to know about the local "shape" of the valley, and therefore how our ball should roll.

Its more like physics simulation of gravity....

Gradient Descent (cont.)

Let's think about what happens when we move the ball a small amount Δv_1 in the v_1 direction, and a small amount Δv_2 in the v_2 direction. Calculus tells us that C changes as follows:

$$\Delta C \approx \frac{\partial C}{\partial v_1} \Delta v_1 + \frac{\partial C}{\partial v_2} \Delta v_2 \tag{2}$$

We're going to find a way of choosing Δv_1 and Δv_2 so as to make ΔC negative; i.e., we'll choose them so the ball is rolling down into the valley.

Gradient Descent (cont.)

To figure out how to make such a choice it helps to define Δv to be the vector of changes in v, $\Delta v \equiv (\Delta v_1, \Delta v_2)^T$, where T is again the transpose operation, turning row vectors into column vectors.

We'll also define the *gradient* of C to be the vector of partial derivatives, $\left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2}\right)^T$. We denote the gradient vector by ∇C, i.e.:

$$\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2}\right)^T \tag{3}$$

∇C is a gradient vector.

Gradient Descent (cont.)

With these definitions, the expression (2) for ΔC can be rewritten as

$$\Delta C \approx \nabla C \cdot \Delta v \tag{4}$$

This equation helps explain why ∇C is called the gradient vector: ∇C relates changes in v to changes in C, just as we'd expect something called a gradient to do.

Gradient Descent (cont.)

But what's really exciting about the equation is that it lets us see how to choose Δv so as to make ΔC negative. In particular, suppose we choose

$$\Delta v = -\eta \nabla C, \tag{5}$$

where η is a small, positive parameter (known as the *learning rate*).

Then Equation (4) tells us that $\Delta C \approx -\eta \nabla C \cdot \nabla C = -\eta \|\nabla C\|^2$.

Because $\|\nabla C\|^2 \geq 0$, this guarantees that $\Delta C \leq 0$, i.e., C will always decrease, never increase, if we change v according to the prescription in (5).

Gradient Descent (cont.)

This is exactly the property we wanted! And so we'll take Equation (5) to define the "law of motion" for the ball in our gradient descent algorithm. That is, we'll use Equation (5) to compute a value for Δv, then move the ball's position v by that amount.

$$v \rightarrow v' = v - \eta \nabla C. \tag{6}$$

Then we'll use this update rule again, to make another move. If we keep doing this, over and over, we'll keep decreasing C until – we hope – we reach a global minimum.

Gradient Descent (cont.)

To make gradient descent work correctly, we need to choose the learning rate η to be small enough that Equation (4) is a good approximation.

If we don't, we might end up with $\Delta C > 0$, which obviously would not be good!

At the same time, we don't want η to be too small, since that will make the changes Δv tiny, and thus the gradient descent algorithm will work very slowly.

In practical implementations, η is often varied so that Equation (4) remains a good approximation, but the algorithm isn't too slow.

Gradient Descent (cont.)

We've explained gradient descent when C is a function of just two variables. But, in fact, everything works just as well even when C is a function of many more variables.

Suppose in particular that C is a function of m variables, $v_1, \ldots, v_m$. Then the change ΔC in C produced by a small change $\Delta v = (\Delta v_1, \ldots, \Delta v_m)^T$ is

$$\Delta C \approx \nabla C \cdot \Delta v \tag{7}$$

where the gradient ∇C is the vector

$$\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \ldots, \frac{\partial C}{\partial v_m}\right)^T \tag{8}$$

Gradient Descent (cont.)

Just as for the two variable case, we can choose

$$\Delta v = -\eta \nabla C \tag{9}$$

and we're guaranteed that our (approximate) expression (7) for ΔC will be negative.

This gives us a way of following the gradient to a minimum, even when C is a function of many variables, by repeatedly applying the update rule

$$v \rightarrow v' = v - \eta \nabla C \tag{10}$$

Learning with Gradient Descent

How can we apply gradient descent to learn in a neural network?

The idea is to use gradient descent to find the weights w_k and biases b_l which minimize the cost in Equation (1).

To see how this works, let's restate the gradient descent update rule, with the weights and biases replacing the variables v_j. In other words, our "position" now has components w_k and b_l, and the gradient vector ∇C has corresponding components $\partial C/\partial w_k$ and $\partial C/\partial b_l$.

Learning with Gradient Descent

Writing out the gradient descent update rule in terms of components, we have

$$w_k \to w'_k = w_k - \eta \frac{\partial C}{\partial w_k} \tag{11}$$

$$b_l \to b'_l = b_l - \eta \frac{\partial C}{\partial b_l} \tag{12}$$

Where, $C = \frac{1}{n}\sum_x C_x$ and $C_x \equiv \frac{\|y(x)-a\|^2}{2}$

By repeatedly applying this update rule we can "roll down the hill", and hopefully find a minimum of the cost function. In other words, this is a rule which can be used to learn in a neural network.

Limitation of Gradient Descent

Update Rules $b_l \to b'_l = b_l - \eta\frac{\partial C}{\partial b_l}$, $C = \frac{1}{n}\sum_x C_x$, and $C_x \equiv \frac{\|y(x)-a\|^2}{2}$

Can we see any issue with this update rule?

In practice, to compute the gradient ∇C we need to compute the gradients ∇C_x separately for each training input, x, and then average them, $\nabla C = \frac{1}{n}\sum_x \nabla C_x$.

Unfortunately, when the number of training inputs is very large this can take a long time, and learning thus occurs slowly.

Stochastic Gradient Descent

An idea called **stochastic gradient descent** can be used to speed up learning.

The idea is to estimate the gradient ∇C by computing ∇C_x for a small sample of randomly chosen training inputs.

By averaging over this small sample it turns out that we can quickly get a good estimate of the true gradient ∇C, and this helps speed up gradient descent, and thus learning.

Stochastic Gradient Descent (cont.)

To connect this explicitly to learning in neural networks, suppose w_k and b_l denote the weights and biases in our neural network.

Then stochastic gradient descent works by picking out a randomly chosen mini-batch of training inputs, and training with those,

$$w_k \to w'_k = w_k - \frac{\eta}{m}\sum_j \frac{\partial C_{X_j}}{\partial w_k} \tag{13}$$

$$b_l \to b'_l = b_l - \frac{\eta}{m}\sum_j \frac{\partial C_{X_j}}{\partial b_l} \tag{14}$$

Where the sums are over all the training examples X_j in the current mini-batch.

History

Learning representations by back-propagating errors:
https://www.cs.toronto.edu/ hinton/absps/naturebp.pdf

Clustering

Unit 4: Clustering

Advanced Data Mining

Rupak Raj Ghimire

School of Mathematical Sciences
Institute of Science and Technology (IoST), TU

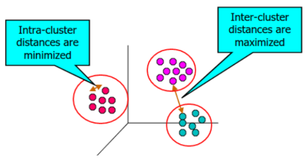
Feb, 2025

Table of contents

What is Cluster Analysis?

What is Cluster Analysis?

Finding groups of objects such that the objects in a group will be similar (or related) to one another and different from (or unrelated to) the objects in other groups


Intra-cluster distances are minimized
Inter-cluster distances are maximized

Cluster Analysis

Cluster Analysis

- A cluster of data objects can be treated collectively as **one group** and so may be considered as a form of data Compression
- The **process of grouping** a set of physical or abstract objects into classes of similar objects is called **clustering**
- A cluster is a **collection of data objects that are similar to one another** within the same cluster and are dissimilar to the objects in other clusters

Cluster vs. Classification

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

What is not Cluster Analysis?

Supervised classification

Have class label information

Simple segmentation

Dividing students into different registration groups alphabetically, by last name

Results of a query

Groupings are a result of an external specification

Applications of Cluster Analysis

- Land Use detection
- Crop / Forest Type identification
- Data segmentation
- Fault Isolation
- Outlier Detection

Challenges of Clustering

Scalability

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

Challenges of Clustering(cont.)

Ability to deal with different types of attributes

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

Challenges of Clustering(cont.)

Discovery of clusters with arbitrary shape

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

Challenges of Clustering(cont.)

Ability to deal with noisy data

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

Challenges of Clustering(cont.)

Incremental clustering and insensitivity to the order of input records

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

Challenges of Clustering(cont.)

High dimensionality

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

Challenges of Clustering(cont.)

Constraint-based clustering

- Real-world applications may need to perform clustering under various kinds of constraints. Suppose that your job is to choose the locations for a given number of new automatic banking machines (ATMs) in a city.
- To decide upon this, you may cluster households while considering constraints such as the city's rivers and highway networks, and the type and number of customers per cluster.
- A challenging task is to find groups of data with good clustering behavior that satisfy specified constraints.

Challenges of Clustering(cont.)

Interpretability and usability

- Users expect clustering results to be interpretable, comprehensible, and usable.
  - That is, clustering may need to be tied to specific semantic interpretations and applications.
- It is important to study how an application goal may influence the selection of clustering features and methods

Types of Data in Cluster Analysis

Suppose that a data set to be clustered contains n objects, which may represent persons, houses, documents, countries, and so on.

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

- Data Matrix: Object-by-variable-structure
- Dissimilarity Matrix: object-by-object structure

Data Matrix

This represents n objects, such as persons, with p variables (also called measurements or attributes), such as age, height, weight, gender, and so on.

The structure is in the form of a relational table, or n-by-p matrix (n objects × p variables):

$$\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{i1} & x_{i2} & \cdots & x_{ip} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix}$$

Dissimilarity Matrix

This stores a collection of proximities that are available for all pairs of n objects. It is often represented by an n-by-n table

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

where $d(i,j)$ is the measured difference or dissimilarity between objects i and j.

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

d(i, j) = d(j, i), and d(i, i) = 0

Dissimilarity Measure

Distance metrics **quantify** the similarity or dissimilarity between data points.

- Used in machine learning, clustering, nearest neighbor search, and other applications.
- Choosing the right distance metric depends on the data characteristics and problem requirements.

Some methods

- Euclidean Distance
- Manhattan (or City Block) Distance
- Minkowski Distance
- Weighted Euclidean distance

Euclidean Distance

Measures the straight-line distance between two points in an n-dimensional space.

Formula

$$d_E(\mathbf{x},\mathbf{y}) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Commonly used in clustering (e.g., k-means), classification (e.g., k-NN), and geometry.

Manhattan (City Block) Distance

Measures distance along axes at right angles, like city blocks.

Formula

$$d_M(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{n}|x_i - y_i|$$

Useful when movement is restricted to grid-like paths (e.g., robotics, urban planning).

Minkowski Distance

Generalization of both Euclidean distance and Manhattan distance.

Formula

$$d_M(\mathbf{x},\mathbf{y}) = \left(\sum_{i=1}^{n}|x_i - y_i|^p\right)^{\frac{1}{p}}$$

where, p is a positive integer. Such a distance is also called L_p norm, in some literature

Special cases

- $p = 1 \rightarrow$ Manhattan Distance (L_1 Norm)
- $p = 2 \rightarrow$ Euclidean Distance (L_2 Norm)

Provides flexibility in measuring distance based on parameter p.

Weighted Euclidean Distance

Assigns different **importance** to each feature dimension.

Formula

$$d_W(\mathbf{x},\mathbf{y}) = \sqrt{\sum_{i=1}^{n} w_i(x_i - y_i)^2}$$

Used when certain features have more significance than others (e.g., medical diagnosis, image processing)
Weighting can also be applied to the Manhattan and Minkowski distances

When to Use Which Distance?

- **Euclidean Distance** when features are equally important and the space is continuous.
- **Manhattan Distance** when movement is constrained (e.g., grid-based systems).
- **Minkowski Distance** when needing flexibility in distance measurement.
- **Weighted Euclidean Distance** when certain features matter more than others.

Interval-Scaled Variables

Interval-scaled variables are continuous measurements of a roughly linear scale

- Typical examples include weight and height, latitude and longitude coordinates (e.g., when clustering houses), and weather temperature
- The measurement unit used can affect the clustering analysis.
  - For example, changing measurement units from meters to inches for height, or from kilograms to pounds for weight, may lead to a very different clustering structure

Interval-Scaled Variables

To help avoid dependence on the choice of measurement units, the data should be standardized.

Standardizing measurements attempts to give all variables an equal weight.

Data Standardization

Data standardization is the process of transforming data to have a common scale, ensuring that **each feature contributes equally** to the clustering process.

It is crucial in clustering because algorithms like K-Means use distance-based similarity measures, which are **sensitive to feature magnitudes**.

Idea

Without standardization, features with **large numerical values dominate** the clustering process.

Standardization ensures fair influence by transforming the data into a comparable scale.

Z-Score Standardization

$$X' = \frac{X - \mu}{\sigma}$$

Example: For *Ages* = [25, 50, 40]

- Mean: $\mu = 38.33$
- Std Dev: $\sigma = 12.58$
- Standardized Age 25: $X' = \frac{25-38.33}{12.58} = -1.06$

Interpretation: Now, all feature values have mean = 0 and variance = 1, making them comparable.

Min-Max Scaling

$$X' = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

Example: For salaries [30,000, 90,000, 60,000]

- $X_{\min} = 30,000$, $X_{\max} = 90,000$
- Normalized 60,000: $X' = \frac{60{,}000-30{,}000}{90{,}000-30{,}000} = 0.5$

Robust Scaling (IQR)

$$X' = \frac{X - \text{Median}}{\text{IQR}}$$

- Uses median and interquartile range (IQR = Q3 - Q1).
- Resistant to outliers.

Unit Vector Scaling

Data is transformed into a unit vector, making it useful for text or high-dimensional clustering

$$X' = \frac{X}{||X||}, \quad ||X|| = \sqrt{X_1^2 + X_2^2 + ... + X_n^2}$$

- Ensures each feature vector has a unit length.
- Useful for cosine similarity-based clustering.

Example

For Age = 25, Salary = 30,000, calculate the norm:
$||X|| = \sqrt{(25)^2 + (30000)^2} = 30000.001$
Now, normalize:
$X'_{Age} = \frac{25}{30000.001} 0.00083$ $X'_{Salary} = \frac{30000}{30000.001} 0.99999$

Log Transformation

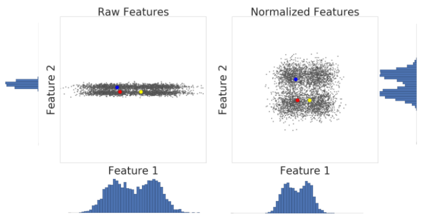
$$X' = \log(X + 1)$$

- Reduces skewness in data.
- Useful for data with large magnitude differences.

Suggested Reading

Book: Data Mining- Concepts and Techniques
Section **7.2 Types of Data in Cluster Analysis**

Clustering: Before and After data standardization



When to Use Which Scaling Method?

Method	When to Use?	Effect
Z-Score	When data follows a normal distribution	Mean = 0, Variance = 1
Min-Max	When we need data in a specific range [0,1]	Retains original feature distribution
Robust Scaling	When data has outliers (e.g., incomes)	Uses median instead of mean

Use Cases

- K-Means, PCA : Z-score Standardization
- Neural Networks : Min-Max Scaling
- Outlier-sensitive data : Robust Scaling (Median-based)

Clustering Methods

Many clustering algorithms exist in the literature.

It is difficult to provide a crisp categorization of clustering methods because these categories may overlap, so that a method may have features from several categories

- Partitioning Methods
- Hierarchical methods
- Density-based methods
- Grid-based methods
- Model-based methods

Partitioning Methods

Given a database of n objects or data tuples

- a partitioning method constructs ***k partitions*** of the data, where each partition represents a **cluster** and $k \leq n$.

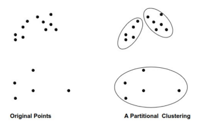
It **classifies** the data into **k groups**, which together satisfy the following requirements:

- Each group must contain at least one object, and
- Each object must belong to exactly one group

Partitioning Methods(cont.)

Given k, the number of partitions to construct, a partitioning method creates an **initial partition**ing.

It then uses an **iterative relocation technique** that attempts to improve the partitioning by moving objects from one group to another



Hierarchical methods

A hierarchical method **creates a hierarchical decomposition** of the given set of data objects.

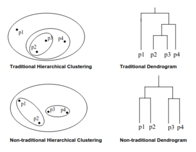
A hierarchical method can be classified as being either **agglomerative** or **divisive**, based on how the hierarchical decomposition is formed.

The **agglomerative** approach, also called the **bottom-up** approach.

Hierarchical methods(cont.)

How it works?

- Starts with all of the objects in the same cluster.
- In each successive iteration, a cluster is split up into smaller clusters, until eventually each object is in one cluster, or until a termination condition holds.


p1
p2
p3
p4
Traditional Hierarchical Clustering
Traditional Dendrogram
Non-traditional Hierarchical Clustering
Non-traditional Dendrogram

Density-based methods

- Distance based method unable to cluster the arbitrary shape of the cluster
- Other clustering methods have been developed based on the notion of density
- Their general idea is to continue growing the given cluster as long as the density (number of objects or data points) in the "neighborhood" exceeds some threshold;
  - that is, for each data point within a given cluster, the neighborhood of a given radius has to contain at least a minimum number of points.
- Such a method can be used to **filter out noise** (outliers) and **discover clusters of arbitrary shape**.

Grid-based methods

- Grid-based methods **quantize** the **object space** into a **finite number of cells** that form a **grid structure**.
- All of the clustering operations are performed on the grid structure (i.e., on the quantized space).
  - The main advantage of this approach is its **fast processing time**, which is typically independent of the number of data objects and dependent only on the number of cells in each dimension in the quantized space.

Centroid-Based Technique

The k-Means Method

The k-means algorithm takes the input parameter, k, and partitions a set of n objects into k clusters so that the resulting **intra-cluster similarity is low but the inter-cluster similarity is high**.

Cluster similarity is measured in regard to the mean value of the objects in a cluster, which can be viewed as the cluster's centroid or center of gravity.

Algorithm 1 K-Means Clustering

```
1: Input: Dataset $X = \{x_1, x_2, \ldots, x_n\}$, number of clusters $k$
2: Output: Cluster centers $C = \{c_1, c_2, \ldots, c_k\}$ and cluster assignments
3: Initialize $k$ cluster centroids $C = \{c_1, c_2, \ldots, c_k\}$ randomly from $X$
4: repeat
5:     Assign each point to the nearest centroid
6:     for each data point $x_i \in X$ do
7:         Compute the distance to each centroid: $d(x_i, c_j)$
8:         Assign $x_i$ to the closest cluster $c_j$
9:     end for
10:    Update centroids
11:    for each cluster $C_j$ do
12:        Compute new centroid as the mean of all assigned points: $c_j = \frac{1}{|C_j|}\sum_{x_i \in C_j} x_i$
13:    end for
14: until Centroids do not change (or a maximum number of iterations is reached)
15: Return final cluster centroids and assignments
```

K-Means Method(cont.)

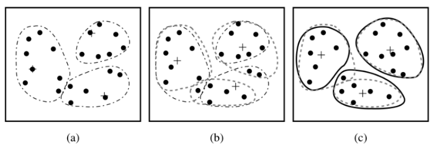


Figure: Building cluster : (a) →(b)→(c)

The k-Means Method(cont.)

Limitation of K-Means

- Can be applied only when the mean of a cluster is defined
- When data has categorical attributes, k means can not be applied. The necessity for users to specify k, the number of clusters, in advance can be seen as a disadvantage
- It is sensitive to noise and outlier data points because a small number of such data can substantially influence the mean value.

K-Modes Method

Another variant to k-means is the k-modes method

It extends the k-means paradigm to cluster categorical data by **replacing the means of clusters with modes**,

- Using new dissimilarity measures to deal with categorical objects and a frequency-based method to update modes of clusters.

The k-means and the k-modes methods can be integrated to cluster data with mixed numeric and categorical values.

EM algorithm

The EM (Expectation-Maximization) algorithm extends the k-means paradigm in a different way.

Whereas the k-means algorithm assigns each object to a cluster, in EM each object is assigned to each cluster according to a weight representing its probability of membership. In other words, there are no strict boundaries between clusters.

Therefore, new means are computed based on weighted measure

Hierarchical Clustering

Hierarchical clustering organizes data into a hierarchy of clusters, represented as a **tree-like structure** known as a **dendrogram**

This algorithm **builds a hierarchy** of clusters by **iteratively merging** or **splitting clusters** based on their similarity

Strength

Hierarchical clustering can discover clusters of arbitrary shapes and sizes, and it provides a visual representation of the hierarchical relationships between clusters

Weakness

Hierarchical clustering can be computationally expensive, especially for large datasets. It is also sensitive to the initial ordering of the data points and the choice of the distance metric

Hierarchical Clustering(cont.)

There are two main types of hierarchical clustering

Agglomerative Clustering

This is **bottom-up strategy** starts by **placing each object in its own cluster** and then **merges** these atomic clusters into **larger and larger clusters**, until all of the objects are in a single cluster or until certain termination conditions are satisfied.

Divisive Clustering

This is **top-down strategy** does the reverse of agglomerative hierarchical clustering by starting with all objects in one cluster. It **subdivides the cluster into smaller** and smaller pieces, until each object forms a cluster on its own or until it satisfies certain termination conditions, such as a desired number of clusters is obtained or the diameter of each cluster is within a certain threshold.

Agglomerative Vs. Divisive hierarchical clustering

- AGNES (AGglomerative NESting) an agglomerative hierarchical clustering method
- DIANA (DIvisive ANAlysis), a divisive hierarchical clustering method

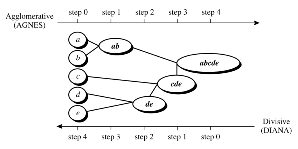


Figure: Agglomerative and divisive hierarchical clustering on data objects a, b, c, d, e

Dendrogram

A tree structure called a **dendrogram** is commonly used to represent the process of hierarchical clustering.

It shows how objects are grouped together step by step

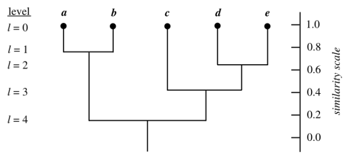


Figure: Dendrogram representation for hierarchical clustering of data objects a, b, c, d, e

Hierarchical Clustering - Types

Single link

When an algorithm uses the *minimum distance*, $d_{min}(C_i, C_j)$, to measure the distance between clusters, it is sometimes called a **nearest-neighbor clustering algorithm**. Moreover, if the clustering process is terminated when the distance between nearest clusters exceeds an arbitrary threshold, it is called a **single-linkage algorithm**.

Complete link

When an algorithm uses the **maximum distance**, $d_{max}(C_i, C_j)$, to measure the distance between clusters, it is sometimes called a **farthest-neighbor clustering algorithm**. If the clustering process is terminated when the maximum distance between nearest clusters exceeds an arbitrary threshold, it is called a **complete-linkage algorithm**.

Hierarchical Agglomerative Clustering(HAC)

Use the distance matrix in table below to perform single link and complete link hierarchical clustering. Show your results by drawing a dendogram

Dataset

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)

Prepare the distance matrix

$$\begin{bmatrix} 0.00 & & & & & \\ 0.23 & 0.00 & & & & \\ 0.22 & 0.14 & 0.00 & & & \\ 0.37 & 0.19 & 0.16 & 0.00 & & \\ 0.34 & 0.14 & 0.28 & 0.28 & 0.00 & \\ 0.24 & 0.24 & 0.11 & 0.22 & 0.39 & 0.00 \end{bmatrix}$$

Identify the min-value element

The minimum value element is $(p_3, p_6)$and value is 0.10

	p_1	p_2	p_3	p_4	p_5	p_6
p_1	0.00					
p_2	0.23	0.00				
p_3	0.22	0.14	0.00			
p_4	0.37	0.19	0.16	0.00		
p_5	0.34	0.14	0.28	0.28	0.00	
p_6	0.24	0.24	0.10	0.22	0.39	0.00

Recalculate or update the distance matrix for cluster

$MIN[dist((p_3, p_6), p_i)]$

Example : p_1

$dist[(p_3, p_6), p_1] = min(dist(p_3, p_1), dist(p_6, p_1))$

$= \text{min}(0.22, 0.23)$
$= 0.22$

	p_1	p_2	$p_{3,6}$	p_4	p_5
p_1	0.00				
p_2	0.23	0.00			
$p_{3,6}$	0.22	0.14	0.00		
p_4	0.37	0.19	0.16	0.00	
p_5	0.34	0.14	0.28	0.28	0.00

Repeat to get another pair

The minimum value element is (p_2, p_5) and value is 0.14

Repeat the process to get the order of the cluster

- 1st cluster (p_3, p_6)
- 2nd cluster (p_2, p_5)
- 3rd cluster (p_2, p_5, p_3, p_6)
- 4th cluster $(p_2, p_5, p_3, p_6, p_4)$
- 5th cluster $(p_2, p_5, p_3, p_6, p_4, p_1)$

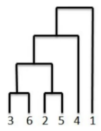


Figure: min version of HAC

for MAX version of HAC

update the distance with
$dist[(p_1, p_2), (p_3)] = max[dist(p_1, p_3), dist(p_2, p_3)]$

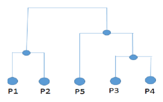


Figure: max version of HAC

Hierarchical Clustering(cont.)

Read the section 7.5 of book.

1. BIRCH: Balanced Iterative Reducing and Clustering Using Hierarchies
2. ROCK(RObust Clustering using linKs): A Hierarchical Clustering Algorithm forCategorical Attributes
3. Chameleon: A Hierarchical Clustering Algorithm Using Dynamic Modeling

Density-based clustering

To discover clusters with **arbitrary shape**, density-based clustering methods have been developed. These typically regard clusters as **dense regions of objects** in the data space that are separated by regions of low density (representing noise).

- DBSCAN: A Density-Based Clustering Method Based on Connected Regions with Sufficiently High Density
- OPTICS: Ordering Points to Identify the Clustering Structure
- DENCLUE: Clustering Based on Density Distribution Functions

DBSCAN

The algorithm grows regions with sufficiently high density into clusters and discovers clusters of arbitrary shape in spatial databases with noise.

It defines a cluster as a maximal set of density-connected points.

- The neighborhood within a radius ϵ of a given object is called the ϵ-**neighborhood** of the object.
- A point is a **core point** if it has more than a specified number of points (MinPts) within ϵ
- A **border point** has fewer than MinPts within , but is in the neighborhood of a core point
- A **noise point** is any point that is not a core point or a border point.
- Every object not contained in any cluster is considered to be noise.

Density-reachability and density connectivity

Given ϵ = Circle radious
MinPts = 3

- Of the labeled points, **m**, **p**, **o**, and **r** are core objects because each is in an ϵ-neighborhood containing at least three points.
- **q** is directly density-reachable from **m**. **m** is directly density-reachable from **p** and vice versa.
- **q** is (indirectly) density-reachable from **p** because **q** is directly density-reachable from **m** and **m** is directly density-reachable from **p**. However, **p** is not density-reachable from **q** because **q** is not a core object. Similarly, **r** and **s** are density-reachable from **o**, and **o** is density-reachable from **r**.
- **o**, **r**, and **s** are all density-connected.

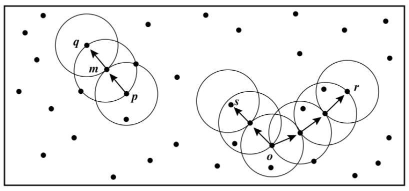


Figure: Density reachability and density connectivity in density-based clustering

Limitation of DBSCAN

DBSCAN can cluster objects given input parameters such as ϵ and *MinPts*, it still leaves the user with the **responsibility of selecting parameter** values that will lead to the discovery of acceptable clusters.

Actually, this is a problem associated with many other clustering algorithms. Such parameter settings are usually empirically set and difficult to determine, especially for real-world, high-dimensional data sets.

Most algorithms are very **sensitive** to such parameter values: **slightly different settings may lead to very different cluster**ings of the data.

Moreover, high-dimensional real data sets often have very **skewed distribution**s, such that their intrinsic clustering structure may not be characterized by global density parameters

OPTICS

Ordering Points to Identify the Clustering Structure
Rather than producing a data set clustering explicitly, **OPTICS** computes an augmented cluster ordering for automatic and interactive cluster analysis.

This ordering represents the density-based clustering structure of the data. It contains information that is equivalent to density-based clustering obtained from a **wide range of parameter settings**.

OPTICS(cont.)

The cluster ordering can be used to extract basic clustering information (such as cluster centers or arbitrary-shaped clusters) as well as provide the intrinsic clustering structure.

Therefore, in order to produce a set or ordering of **density-based** clusters, we can extend the DBSCAN algorithm to process a set of distance parameter values at the same time.

To construct the different clusterings simultaneously, the objects should be processed in a specific order. This order selects an object that is **density-reachable** with respect to the lowest ϵ-value so that clusters with higher density (lower ϵ) will be finished first.

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

OPTICS(cont.)

Suppose that $\epsilon = 6$ mm and MinPts $= 5$.

- The core-distance of p is the distance, ϵ' , between p and the fourth closest data object.
- The reachability-distance of q_1 with respect to p is the core-distance of p (i.e., $\epsilon' = 3$ mm) because this is greater than the Euclidean distance from p to q_1 .
- The reachability-distance of q_2 with respect to p is the Euclidean distance from p to q_2 because this is greater than the core-distance of p

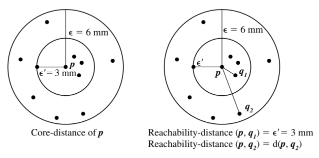


Figure: OPTICS terminology

OPTICS(cont.)

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

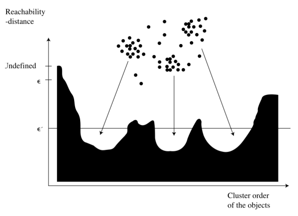


Figure: Cluster ordering in OPTICS

Grid-Based Methods

The grid-based clustering approach uses a **multiresolution grid data structure**.

It **quantizes** the object space into a **finite number of cells** that form a grid structure on which all of the operations for clustering are performed.

The main advantage of the approach is its **fast processing time**, which is typically **independent of the number of data objects**,yet **dependent on only the number of cells** in each dimension in the quantized space.

Grid-Based Methods(cont.)

STING

explores statistical information stored in the grid cells

WaveCluster

clusters objects using a **wavelet transform** method

CLIQUE

represents a **grid and density-based approach** for clustering in high-dimensional data space

STING: STatistical INformation Grid

STING is a grid-based multiresolution clustering technique in which the spatial area is divided into rectangular cells.

There are usually several levels of such rectangular cells corresponding to different levels of resolution, and **these cells form a hierarchical structure**: each cell at a high level is partitioned to form a number of cells at the next lower level.

STING: STatistical INformation Grid

Statistical parameters of higher-level cells can easily be computed from the parameters of the lower-level cells.
These parameters include the following:

- the attribute-independent parameter: count
- the attribute-dependent parameters: mean, stdev (standard deviation), min (minimum), max (maximum)
- the type of distribution that the attribute value in the cell follows: normal, uniform, exponential, or none (if the distribution is unknown)

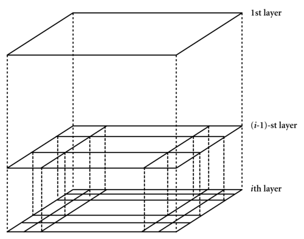


Figure: A hierarchical structure for STING clustering

Model-Based Clustering Methods

Model-based clustering methods attempt to optimize the **fit between the given data and some mathematical model**.

Such methods are often based on the assumption that the data are generated by a mixture of underlying probability distributions

- Expectation-Maximization
- Neural network approach

Expectation-Maximization

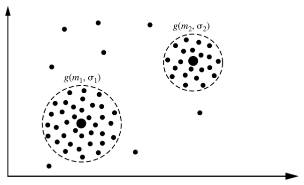
EM starts with an **initial estimate or "guess"** of the parameters of the mixture model (collectively referred to as the parameter vector).

It iteratively **rescores** the objects against the mixture density produced by the parameter vector. The rescored objects are then used to update the parameter estimates.

Each object is assigned a probability that it would possess a certain set of attribute values given that it was a member of a given cluster.

Expectation-Maximization(cont.)

Each cluster can be represented by a probability distribution, centered at a mean, and with a standard deviation.
Here, we have two clusters, corresponding to the Gaussian distributions $g(m_1, \sigma_1)$ and $g(m_2, \sigma_2)$, respectively, where the dashed circles represent the first standard deviation of the distributions.



Expectation-Maximization(cont.)

Step 1: Make an initial guess of the parameter vector

This involves randomly selecting k objects to represent the cluster means or centers (as in k-means partitioning), as well as making guesses for the additional parameters

Step 2: Iteratively refine the parameters (or clusters)

based on the following two steps:

1. Expectation Step
2. Maximization Step

1. Expectation Step

Assign each object x_i to cluster C_k with the probability

$$P(x_i \epsilon C_k) = p(C_k|x_i) = \frac{p(C_k)p(x_i|C_k)}{p(x_i)}$$

where $p(x_i|C_k) = N(m_k, E_k(x_i))$ follows the normal (i.e., Gaussian) distribution around mean, m_k , with expectation, E_k.

In other words, this step calculates the probability of cluster membership of object x_i , for each of the clusters.

These probabilities are the "expected" cluster memberships for object x_i

2. Maximization Step

Use the probability estimates from above to re-estimate (or refine) the model parameters.

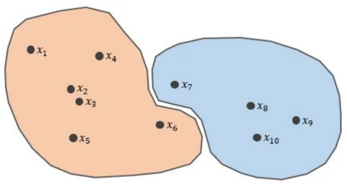
$$m_k = \frac{1}{n} \sum_{i=1}^{n} \frac{x_i P(x_i \epsilon C_k)}{\sum_j P(x_i \epsilon C_j)}$$

This step is the "maximization" of the likelihood of the distributions given the data.

Hard partition

where the data points are strictly allocated as a member of one cluster and are not a member of another cluster, assuming that the number of clusters is known.

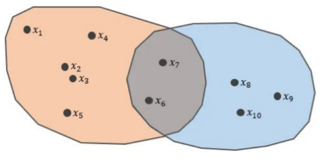
The k-means is one of the algorithms that use a hard partition.



Soft Partition

Every data point is given a probability of closeness $[0, 1]$ for existing clusters, assuming that the number of clusters is known.

One of the algorithms that use fuzzy partition is Fuzzy c-means.



Fuzzy Clustering

With fuzzy clustering, a data point can belong to multiple clusters to varying degrees, providing a richer representation of complex relationships

The key idea behind fuzzy clustering lies in the concept of **membership functions**.

Instead of assigning binary membership values (0 or 1) as in traditional clustering, fuzzy clustering employs membership values ranging from 0 to 1.
These values represent the **degree of belongingness** of each data point to each cluster.

A higher membership value indicates a **stronger association** with a particular cluster, while a lower value signifies a **weaker association**

Fuzzy c-means (FCM)

FCM is the most well-known and widely used fuzzy clustering technique.

It is an iterative algorithm that minimizes the sum of the weighted squared distances between each data point and the centers of the clusters.

The degree of membership of each data point to each cluster is calculated using a membership function, which assigns a probability value between 0 and 1 for each cluster

Fuzzy Clustering - Applications

- **Image Segmentation**: granular delineation of image regions, enabling better object recognition and scene understanding
- **Pattern Recognition**: character recognition or speech recognition
- **Customer Segmentation**:considering the degree of membership to different segments, businesses can tailor personalized marketing strategies
- **Document clustering**: cluster documents based on their content, such as keywords, topics, and themes

Self study

- Graph based Clustering
- Connected Components Clustering (CCC)
- highly connected subgraphs clustering

Evaluation of Clustering

Evaluating the effectiveness of the clustering results, known as clustering evaluation or validation

It can be used to determine which clustering algorithm is best suited for a particular dataset and task, and to tune the hyperparameters of these algorithms

Challenges

Since clustering is an unsupervised learning method, there are no ground truth labels against which the clustering results can be compared.

Determining the correct number of clusters or the best clustering is often a subjective decision, even for domain experts. What one considers as a meaningful cluster, another might dismiss as coincidental.

Evaluation of Clustering

Challenges

In many real-world datasets, the boundaries between clusters are not clear-enough
Some data points might sit at the boundary of two clusters and could be reasonably assigned to both.

Different applications might prioritize different aspects of clustering
For example, in one application, it might be essential to have tight, well-separated clusters, while in another, capturing the overall data structure might be more important.

Types of Evaluation

Two types of clustering evaluation measures (or metrics) Internal measures do not require any ground truth to assess the quality of clusters. They are based solely on the data and the clustering results. External measures compare the clustering results to ground truth labels.

Types of Evaluation(cont.)

Internal Evaluation Measures

Most internal validation measures are based on the following two criteria:

Compactness measures how closely related objects in the same cluster are.
Compactness can be measured in different ways, such as by using the variance of the points within each cluster, or computing the average pairwise distance between them.

Separation measures how distinct or well-separated a cluster is from other clusters.
Examples for measures of separation include pairwise distances between cluster centers or pairwise minimum distances between objects in different

Types of Evaluation(cont.)

External Evaluation Measures

compare the clustering results to ground truth labels

Silhouette Index

The **silhouette index** (or score) **measures the degree of separation** between clusters by comparing each object's similarity to its own cluster against its similarity to objects in other clusters

It measures how similar a point is to its own cluster (cohesion) compared to other clusters (separation).

- Close to 1: Well-clustered
- Close to 0: Overlapping clusters
- Negative: Misclassified points

Silhouette Coefficient

- For each point x_i in a cluster:
  1. Compute the average intra-cluster distance ($a(x_i)$):

$$a(x_i) = \frac{1}{|C| - 1} \sum_{x_j \in C_i, j \neq i} d(x_i, x_j)$$

We can interpret $a(x_i)$ as a measure of how well point x_i is matched to its own cluster (the smaller the value, the better the match).

Note that $a(x_i)$ is not clearly defined for clusters with size 1, in which case we set $s(x_i) = 0$.

Silhouette Coefficient (cont.)

2. Compute the nearest-cluster average distance ($b(x_i)$):

$$b(x_i) = \min_{C' \neq C} \frac{1}{|C'|} \sum_{j \in C'} d(i, j)$$

$b(x_i)$ is the average distance between x_i and the points in its neighboring cluster, i.e., the cluster whose points have the smallest average distance to xi

3. Compute the silhouette coefficient $S(x_i)$:

$$S(x_i) = \frac{b(x_i - a(x_i)}{\max(a(x_i), b(x_i))}$$

The silhouette coefficient ranges from **-1** to **+1**, where a high value indicates that the point is well matched to its own cluster and poorly matched to neighboring clusters

Silhouette Index(cont.)

Based on the silhouette coefficients of the samples, we now define the silhouette index (SI) as the average of the coefficients over all the data points:

$$SI = \frac{1}{n} \sum_{i=1}^{n} S(x_i)$$

Where, n is the total number of data points

The silhouette index provides an overall measure for the quality of the clustering:

1. An index close to 1 means that the clusters are compact and well separated.
2. An index around 0 indicates overlapping clusters.
3. An index close to -1 means the clustering has either too many or too few clusters.

Silhouette Index(cont.)

Importance

- Provides an intuitive way to assess clustering performance.
- Helps in choosing the optimal number of clusters.
- Can be used to compare different clustering algorithms.

Limitations

- Computationally expensive for large datasets.
- May not work well for non-spherical clusters.
- Sensitive to noise and outliers.

SI for estimating number of clusters

- The red dashed line represents the average silhouette coefficient for each clustering scenario.
- Since $K = 3$ has a higher average silhouette score than $K = 4$, it suggests that $K = 3$ might be a better choice for clustering in this case
- However, the final decision should also consider domain knowledge and the structure of the data

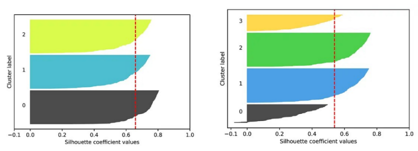


Figure: SI for (left) K=3 and (right) k=4

Calinski-Harabasz Index (CHI)

The Calinski-Harabasz index (CHI) measures the ratio between the between-cluster separation and the within-cluster dispersion

$$CHI = \frac{BCSS/(k-1)}{WCSS/(n-k)}$$

Where,

- k is the number of clusters
- n is the total number of data points
- BCSS (Between-Cluster Sum of Squares)
- WCSS (Within-Cluster Sum of Squares)

Pros: Performs well in identifying the optimal number of clusters
Cons: Assumes spherical clusters

Calinski-Harabasz Index(cont.)

BCSS (Between-Cluster Sum of Squares)

It is the weighted sum of squared Euclidean distances between each cluster centroid (mean) and the overall data centroid (mean)

$$BCSS = \sum_{i=1}^{k} n_i ||c_i - c||^2$$

Where n_i is the number of data points in cluster i, c_i is the centroid (mean) of cluster i, and c is the overall centroid(mean) of all data points

BCSS measures how well the clusters are separated from each other (the higher the better)

Calinski-Harabasz Index(cont.)

WCSS (Within-Cluster Sum of Squares)

It is the sum of squared Euclidean distances between the data points and their respective cluster centroid:

$$WCSS = \sum_{i=1}^{k} \sum_{x \in C_i} ||x - c_i||^2$$

WCSS measures the compactness or cohesiveness of the clusters (the smaller the better).

Minimizing WCSS (also known as the inertia) is the objective of centroid-based clustering such as k-means.

Calinski-Harabasz Index(cont.)

Advantages

- Simple to calculate and computationally efficient
- Easy to interpret. Higher values generally indicate better clustering
- Like silhouette score, it can be used to find the optimal number of clusters.

Other Evaluation Scores

Read yourself

- Davies-Bouldin Index - measures the average similarity between each cluster and its most similar cluster
- Dunn Index - Measures the ratio of the minimum inter-cluster distance to the maximum intra-cluster distance. Higher values indicate better clustering.

What to choose?

- Silhouette Index or Dunn Index - If clusters are well-separated and compact
- Davies-Bouldin Index or CH Index - If clusters have varying densities

External Evaluation Measures

External evaluation measures are used when the true labels of the data points are known.

These measures compare the the results of the clustering algorithm against the ground truth labels.

Contingency Matrix

Similar to confusion matrices in classification problems, a contingency matrix (or table) describes the relationship between the ground truth labels and the cluster labels

The rows of the matrix represent the ground-truth classes and its column represent the clusters. Each cell in the matrix, denoted by nij, contains the count of the data points that have a class label i and were assigned to cluster j

Various external evaluation metrics, such as adjusted Rand index and Fowlkes-Mallows index use the contingency matrix as the basis for their calculation

Random Index

The Rand Index (RI), named after William Rand, measures the similarity between the cluster assignments and the true class labels by making pairwise comparisons.

It is calculated as the ratio of the number of agreements between the cluster assignments and the class labels to the total number of pairs of data points:

$$RI = \frac{a+b}{\binom{n}{2}}$$

Where,

- a is the number of pairs of points that have the same class label and also belong to the same cluster.

Random Index (cont.)

- b is the number of pairs of points that have different class labels and belong to different clusters.
- n is the total number of points.

The RI ranges from 0 to 1, where 1 indicates that the cluster assignments and the class labels are exactly the same

Fowlkes-Mallows Index (FMI)

The Fowlkes-Mallows Index (FMI) is defined as the geometric mean of the pairwise precision (the accuracy of grouped pairs of points) and recall (the completeness of correctly grouping pairs that belong together):

$$FMI = \frac{TP}{\sqrt{(TP+FP)(TP+FN)}}$$

where:

- TP (True Positive) is the number of pairs of points that have the same class label and belong to the same cluster.
- FP (False Positive) is the number of pairs of points that have different class labels but are assigned to the same cluster.
- FN (False Negative) is the number of pairs of points that have the same class labels but are assigned to different clusters

Fowlkes-Mallows Index (FMI) (cont.)

The FMI score ranges from 0 to 1, where 0 indicates no correlation between the clustering results and the true labels, and 1 represents a perfect correlation.

Other Evaluation Scores

Read yourself

- Adjusted Rand index (ARI)
- Homogeneity
- Completeness
- V-measure

Parameter Estimation

The number of clusters (KK) affects clustering quality.

- Too few clusters may merge distinct groups
- Too many may overfit noise

Common Methods

- Elbow Method
- Silhouette Score
- Davies-Bouldin Index
- Dunn Index

Elbow Method

A technique used to determine the optimal number of clusters (K) in clustering.

Based on the Sum of Squared Errors (SSE) or Within-Cluster Sum of Squares (WCSS).

The goal is to find a balance between too few and too many clusters.

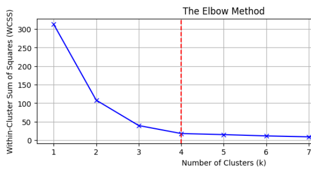
Working

- Compute clustering for different values of K (e.g., 1 to 10)
- Calculate the WCSS (Within-Cluster Sum of Squares) for each K
- Plot KK vs. WCSS
- Identify the "elbow point" where WCSS starts decreasing at a slower rate

Elbow Method (cont.)

WCSS measures how well the data points are clustered around their respective centroids. It is defined as the sum of the squared distances between each point and its cluster centroid:

$$WCSS = \sum_{i=1}^{k} \sum_{x \in C_i} ||x - c_i||^2$$



Anomaly / Outlier Detection

Unit 5: Outlier Detection

–

School of Mathematical Sciences
Institute of Science and Technology (IoST), TU

March, 2025

Table of contents

Anomaly/Outlier Detection

Similar terms

- Anomaly
- Deviation
- Extreme Value
- Noise
- Novelty : something new, off-track, unusual
- Fluctuation
- Discrepancy

Introduction

What are anomalies/outliers?

The set of data points that are considerably different than the remainder of the data

An outlier is a data point that significantly differs from other data points in a dataset.

Outliers are different from the noise data

- Noise is random error or variance in a measured variable
- Noise should be removed before outlier detection

Introduction (cont.)

How outliers are formed?

Outliers can occur due to various reasons, such as errors in data collection, measurement variability, or genuine rare occurrences in the data.

Identifying and handling outliers is important in data analysis and statistics because they can distort statistical analyses and machine learning models.

Characteristics of Outliers

Unusual Value

Outliers are data points that are significantly different from the majority of other data points in the dataset.

They may be unusually high or low compared to the rest of the data.

Impact on Summary Statistics

Outliers can significantly affect summary statistics such as the mean and standard deviation.

For example, a single extremely high or low value can skew the mean and increase the variance.

Characteristics of Outliers (cont.)

Visual Detection

Outliers can often be detected visually in plots such as histograms, box plots, or scatter plots.

They may appear as points that are far from the main cluster of data points.

Quantitative Detection

Outliers can also be detected quantitatively using statistical methods such as the z-score, which measures how many standard deviations a data point is away from the mean.

Characteristics of Outliers (cont.)

Impact on Models

Outliers can have a significant impact on statistical analyses and machine learning models. They can lead to biased estimates, reduced model performance, or incorrect conclusions if not properly handled.

Sources of Outliers

Data Entry Errors

Outliers may occur due to human errors during data entry, recording, or transcription

For example, typos, misreadings, or mistakes in data collection can lead to erroneous data points that deviate significantly from the true values

Sources of Outliers (cont.)

Sampling Error

Outliers may arise from sampling errors when the sample size is small or not representative of the population

Sampling variability, sampling bias, or outliers in the population itself can lead to outliers in the sample data

Sources of Outliers (cont.)

Natural Variability

In some cases, outliers may occur naturally due to inherent variability or heterogeneity in the data-generating process

Natural processes, such as biological variation, environmental factors, or random fluctuations, can produce outliers that deviate from the central tendency of the data

Sources of Outliers (cont.)

Measurement Variability

Variability in measurement instruments or techniques can lead to outliers in the data

Measurement errors, calibration issues, or sensor malfunctions may result in data points that are inconsistent or inaccurate compared to the majority of observations

Sources of Outliers (cont.)

Measurement Units

Outliers may arise from inconsistencies in measurement units or scales used to record data

Incompatible units, unit conversions, or mixing of different measurement scales can result in outliers that do not conform to the expected distribution of values

Sources of Outliers (cont.)

Data Collection Methods

Outliers may be introduced by limitations or biases in data collection methods

Biased sampling, non-random selection of subjects, or incomplete data collection can lead to outliers that do not accurately represent the underlying population

Sources of Outliers (cont.)

Genuine Extreme Values

Outliers may represent genuine extreme values or rare events in the data

Unusual phenomena, extreme weather events, or rare occurrences may produce outliers that are legitimate observations but deviate from the typical patterns in the data

Sources of Outliers (cont.)

Errors in Data Processing

Outliers can result from errors or anomalies introduced during data processing or manipulation

Mistakes in data cleaning, transformation, or aggregation processes may lead to outliers that distort the analysis or modeling results

Sources of Outliers (cont.)

Data Corruption or Tampering

Outliers may be introduced deliberately or accidentally through data corruption or tampering

Malicious attacks, data breaches, or data manipulation can alter the integrity of the data and introduce outliers that compromise its quality and reliability

Types of Outliers

Three kinds

- Global
- Contextual
- Collective outliers

Global Outlier (Point Anomaly)

Object is Global Outlier (O_g) if it significantly deviates from the rest of the data set

Ex. Intrusion detection in computer networks

Issue

- Find an appropriate measurement of deviation

Contextual Outlier (Conditional Outlier)

Object is Contextual Outlier O_c if it deviates significantly based on a selected context

Ex. 25 Degree C in Kathmandu:
Outlier?$\rightarrow$ depending on summer or winter

Contextual Outlier (Conditional Outlier)

Attributes of data objects should be divided into two groups

- **Contextual attributes:** defines the context, e.g., time location
- **Behavioral attributes:** characteristics of the object, used in outlier evaluation, e.g., temperature

Can be viewed as a generalization of local outliers whose density significantly deviates from its local area

Issue

- How to define or formulate meaningful context?

Collective Outliers

A subset of data objects collectively deviate significantly from the whole data set, even if the individual data objects may not be outliers

Applications: E.g., intrusion detection:

- When a number of computers keep sending denial-of-service packages to each other

Collective Outliers

Detection of collective outliers

- Consider not only behavior of individual objects, but also that of groups of objects
- Need to have the background knowledge on the relationship among data objects, such as a distance or similarity measure on objects

Outlier Detection Problem

Variants of Anomaly/Outlier Detection Problems

- Given a database D, find all the data points $x \in D$ with anomaly scores greater than some threshold t
- Given a database D, find all the data points $x \in D$ having the $top - n$ largest anomaly scores $f(x)$
- Given a database D, containing mostly normal (but unlabeled) data points, and a test point x, compute the anomaly score of x with respect to D

Applications

The outlier detection techniques are applicable in various fields

- Credit card fraud detection
- Telecommunication fraud detection
- Network intrusion detection
- Fault detection
- ...

Challenges

How many outliers are there in the data?

Method is unsupervised

Validation can be quite challenging (just like for clustering)

Anomaly Detection Schemes

General Steps

1. Build a profile of the normal behavior

Profile can be patterns or summary statistics for the overall population

2. Use the normal profile to detect anomalies

Anomalies are observations whose characteristics differs significantly from the normal profile

Types of Anomaly Detection Schemes

Based on user-labeled data

- Supervised Methods, Semi-Supervised

Based on assumptions about normal data and outliers

- Unsupervised Method
- Graphical and Statistical Based
- Distance Based
- Model Based

Supervised Method

Modeling outlier detection as a classification problem

Samples examined by domain experts used for training testing

Methods for Learning a classifier for outlier detection effectively:

- Model normal objects report those not matching the model as outliers, or
- Model outliers and treat those not matching the model as normal

Supervised Method (cont.)

Challenges

- Imbalanced classes
  - i.e., outliers are rare
  - Boost the outlier class and make up some artificial outliers
- Catch as many outliers as possible - i.e., recall is more important than accuracy (i.e., not mislabeling normal objects as outliers)

Unsupervised Method

Assume the normal objects are somewhat "clustered" into multiple groups, each having some distinct features

An outlier is expected to be far away from any groups of normal objects

Weakness

Cannot detect collective outlier effectively

Normal objects may not share any strong patterns, but the collective outliers may share high similarity in a small area

Semi-Supervised Methods

Situation

In many applications, the number of labeled data is often small: Labels could be on outliers only, normal objects only, or both

Semi-supervised outlier detection: Regarded as applications of semi-supervised learning

Semi-Supervised Methods

If some labeled normal objects are available

- Use the labeled examples and the proximate unlabeled objects to train a model for normal objects
- Those not fitting the model of normal objects are detected as outliers

If only some labeled outliers are available, a small number of labeled outliers many not cover the possible outliers well

- To improve the quality of outlier detection, one can get help from models for normal objects learned from unsupervised methods

Graphical Approaches

Boxplot (1-D), Scatter plot (2-D), Spin plot (3-D)

Limitations

Time consuming

Subjective

Interquartile Range (IQR)

The interquartile range (IQR) is a measure of statistical dispersion by dividing a data set into quartiles and is also called as Mid-spread or H-spread.

It shows how the data is spread about the median.

The data sorted in ascending order and then divided into quartiles.

IQR is calculated as the difference between the 75th and 25th percentiles.

This method is also called **Extreme Value analysis**

IQR

IQR = Q3 - Q1

Upper Limit = Q3 + 1.5 IQR

Lower Limit = Q1 - 1.5 IQR

Statistical Approaches

Assume a parametric model describing the distribution of the data (e.g., normal distribution)

Apply a statistical test that depends on

- Data distribution
- Parameter of distribution (e.g., mean, variance)
- Number of expected outliers (confidence limit)

Limitation of Statistical Approaches

Most of the tests are for a single attribute

In many cases, data distribution may not be known

For high dimensional data, it may be difficult to estimate the true distribution

Distance-based Approaches

Data is represented as a vector of features

Three major approaches

- Nearest-neighbor based
- Density based
- Clustering based

Nearest-Neighbor Based Approach

Compute the distance between every pair of data points

There are various ways to define outliers:

- Data points for which there are fewer than p neighboring points within a distance D
- The top n data points whose distance to the k_{th} nearest neighbor is greatest
- The top n data points whose average distance to the k nearest neighbors is greatest

Density Based: LOF Approach

For each point, compute the density of its local neighborhood

Compute local outlier factor (LOF) of a sample p as the average of the ratios of the density of sample p and the density of its nearest neighbors

Outliers are points with largest LOF value

In the NN approach, p_2 is not considered as outlier, while LOF approach find both p_1 and p_2 as outliers

Clustering based Approach

Basic idea: Cluster the data into groups of different density

Choose points in small cluster as candidate outliers

Compute the distance between candidate points and non-candidate clusters.
If candidate points are far from all other non-candidate points, they are outliers