

# CS5228 Knowledge Discovery and Data Mining

AY2024/25 Sem2 By Zhao Peiduo

## Lecture 1

### Introduction

**Common Data Mining Tasks** Data mining encompasses various techniques to analyze and extract patterns from large datasets. Some of the most common data mining tasks include:

- **Association Rules:** This method analyzes *transactional data*, where a transaction is a data record consisting of a set of items from a fixed collection. The goal is to identify *association rules* that predict the occurrence of items based on the presence of other items in the dataset.
- **Clustering:** Clustering involves grouping data points based on a well-defined notion of similarity. The objective is to form *clusters*, ensuring that data points within the same cluster have high intra-cluster similarity while minimizing inter-cluster similarity with other clusters.
- **Classification:** This method uses datasets with multiple attributes to determine the *categorical value* of an attribute as a function of other attribute values. Popular classification techniques include K-Nearest Neighbor, Decision Trees, and Linear Classification.
- **Regression:** Similar to classification, regression also works with datasets having multiple attributes, but it predicts *numerical values* of an attribute as a function of other attributes. Common regression methods include K-Nearest Neighbor, Regression Trees, and Linear Regression.
- **Graph Mining:** This technique analyzes data represented as a graph,  $G = (V, E)$ , where  $V$  represents data points (vertices) and  $E$  represents relationships between them (edges). Typical patterns derived from graph mining include identifying communities of nodes and detecting important nodes within the network.
- **Recommender Systems:** Recommender systems work with *user-rated items* (such as movie ratings) to predict missing values and recommend items based on similarities. They exploit item features and user similarities to enhance recommendations.

#### Types of Attributes

- **Categorical (Qualitative):**
  - **Nominal:**
    - \* Values are only labels.
    - \* Operations:  $=$ ,  $\neq$
    - \* Examples: sex (m/f), eye color, zip code.
  - **Ordinal:**
    - \* Values are labels with a meaningful order.
    - \* Operations:  $=$ ,  $\neq$ ,  $<$ ,  $>$
    - \* Examples: street numbers, education level.
- **Numerical (Quantitative):**
  - **Interval:**
    - \* Values are measurements with a meaningful distance.
    - \* Operations:  $=$ ,  $\neq$ ,  $<$ ,  $>$ ,  $+$ ,  $-$
    - \* Examples: body temperature in °C, calendar dates.
  - **Ratio:**
    - \* Values are measurements with a meaningful ratio.
    - \* Operations:  $=$ ,  $\neq$ ,  $<$ ,  $>$ ,  $+$ ,  $-$ ,  $\times$ ,  $\div$
    - \* Examples: age, weight, income, blood pressure.

#### Types of Data

Data can be classified into three main types based on structure and organization:

- **(Well-)Structured Data:**
  - Highly organized: adheres to a predefined data model.
  - Each object has the same fixed set of attributes.
  - Easy to search, aggregate, manipulate, and analyze.
  - Examples: relational databases, spreadsheets.
- **Semi-Structured Data:**
  - No rigid data model: mix of structured and unstructured data.
  - Data exchange formats: XML, JSON, CSV.
  - Tagged unstructured data (e.g., photo with date/time, location, exposure, resolution, flash, etc.).
- **Unstructured Data:**
  - No fixed data model.
  - Requires more advanced data analysis techniques.
  - Examples: images, videos, audio, text, social media.

#### Data Quality

- **Noise:** Data can be defined as: true signal + **noise**. Sources of noise include:
  - Sensor readings from faulty devices (e.g., intrinsic noise or external influences).
  - Errors during data entry (by humans or machines).
  - Errors during data transmission.
  - Inconsistencies in data formats (e.g., ISO time vs. Unix time, DD/MM/YYYY vs. MM/DD/YYYY).
  - Inconsistencies in conventions (e.g., meters vs. miles, meters vs. centimeters).
- **Outliers:** An outlier is a data point with attribute values considerably different from other points. Outliers can be classified into:
  - **Outliers as noise:**
    - \* They negatively interfere with data analysis.
    - \* Removal of outliers or using robust methods is recommended.
  - **Outliers as targets:**
    - \* The goal is to detect rare or anomalous events such as credit card fraud detection and intrusion detection in security systems.
- **Missing Values**
  - Common causes of missing values:
    - \* Attribute values not collected (e.g., broken sensor, person refused to report age).
    - \* Attributes not applicable in all cases (e.g., no income data for children).
  - Handling missing values:
    - \* Remove data points with missing values.
    - \* Remove attributes with missing values (if not essential).
    - \* Try to fill in missing values (e.g., using average temperature from nearby sensors).
- **Duplicates**
  - Duplicates refer to data points representing the same object/entity.
    - \* **Exact duplicates:** Data points have identical attribute values.
    - \* **Near duplicates:** Data points slightly differ in their attribute values (e.g., same person with phone numbers in different formats).
  - Duplicate elimination:
    - \* Relatively easy for exact duplicates.
    - \* Challenging for near duplicates.

#### Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) is an essential step in data analysis to identify potential issues such as noise, outliers, missing values, and class distribution imbalances.

- **Identifying Noise**
  - Using histograms to inspect the distribution of data values.
- **Identifying Noise / Outliers**
  - Using box plots to inspect the distribution of attribute values.
    - \* Make outliers explicit.
  - Using scatter plots to inspect correlations.
    - \* Not always feasible in practice.
    - \* Requires good understanding of data.
- **Handling Missing Values**
  - Example: Default value (0) if people did not disclose weight.
    - \* Can negatively affect simple analysis such as calculating means/averages.
- **Distribution of Class Labels**
  - Classification tasks generally benefit from balanced datasets.
    - \* Balanced = all classes are (almost) equally represented.
  - Distribution of classes also affects the evaluation of found patterns.

#### Data Preprocessing

- **Main Purposes**
  - Improve data quality ("Garbage in, garbage out!").
  - Generate valid input for data mining algorithms.
  - Remove complexity from data to ease analysis.
- **Core Preprocessing Tasks**
  - Data cleaning
  - Data reduction
  - Data transformation
  - Data discretization
- **Data Cleaning**
  - Remove or fill missing values.
  - Identify and remove outliers (if outliers are not the goal of the analysis).
  - Identify and remove/merge duplicates.
  - Correct errors and inconsistencies (e.g., convert inches to centimeters).
  - *Non-trivial tasks that are typically very application-specific.*
- **Data Reduction**
  - **Reducing the number of data points**
    - \* Sampling — selecting a subset of data points (random or stratified sampling).
    - \* Used for preliminary analysis or large datasets.
  - **Reducing the number of attributes**
    - \* Removing irrelevant attributes (e.g., IDs, sensitive attributes).
    - \* Dimensionality reduction (PCA, LDA, t-SNE).
  - **Reducing the number of attribute values**
    - \* Aggregation or generalization.
    - \* Binning with smoothing.
- **Data Transformation**
  - Some data reduction techniques also transform data (e.g., dimensionality reduction, aggregation, binning).
  - Attribute construction:
    - \* Add or replace attributes inferred from existing attributes.
    - \* Example: weight, volume  $\rightarrow$  density.
  - Normalization:
    - \* Scaling attribute values to a specified range (e.g., [0,1]).
    - \* Standardization: scaling using mean and standard deviation.
- **Data Discretization**
  - Converting continuous attributes into ordinal attributes.
  - Some algorithms accept only categorical attributes.
  - Convert a regression task to a classification task.
- **One-Hot Encoding**
  - Converting categorical attributes into numerical attributes.
  - Transform categorical attributes into binary attributes (0/1).
  - Allows the application of numerical methods on categorical attributes.

## Lecture 2

### Clustering

#### Goal of Clustering

- Separate **unlabeled** data into groups of **similar** objects/points
- Maximize **intra-cluster** similarity
- Minimize **inter-cluster** similarity

#### Meso-level perspective on data

- **micro:** individual data points
- **meso:** clusters
- **macro:** whole dataset

#### Ingredients for Clustering

- **Representation of objects, e.g.:**
  - (Multidimensional) point coordinates  $x, y$
  - Sets  $A, B$  (e.g., items in a transaction)
  - Vectors  $u, v$  (e.g., TF-IDF)
- **Similarity Measure, e.g.:**
  - Euclidean Distance:  $dist_{\text{euclidean}}(x, y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$
  - Jaccard Similarity:  $sim_{\text{jaccard}}(A, B) = \frac{|A \cap B|}{|A \cup B|}$
  - Cosine Similarity:  $sim_{\text{cosine}}(u, v) = \frac{u \cdot v}{\|u\| \|v\|}$
- **Clustering Algorithm**
  - Process that determines if an object belongs to a cluster

#### Types of Clusters

- **Well-separated vs. Center-based**
  - Well-separated: Any object in a cluster is closer to every other object in the cluster than to any point outside the cluster.
  - Center-based: Any object in a cluster is closer to the "center" of the cluster than to the center of any other cluster. The cluster center is commonly called a **centroid**.
- **Contiguity-based vs. Density-based**
  - Contiguity-based: Two objects are in the same cluster if they are more similar than a specified threshold, ensuring each object is more similar to some object in the cluster than to any point in a different cluster.
  - Density-based: A cluster is defined as a dense region of objects surrounded by regions of lower density, which allows better handling of noise.
- **Partitional vs. Hierarchical**
  - Partitional: The data is divided into non-overlapping subsets (i.e., clusters), where each object belongs to exactly one cluster or no cluster at all.
  - Hierarchical: Clusters can be nested, and a point can belong to different clusters depending on the hierarchy level.
- **Exclusive vs. Non-exclusive / Overlapping**
  - Exclusive: Each object belongs to exactly one cluster.
  - Non-exclusive / Overlapping: An object can belong to more than one cluster at a time. Fuzzy clustering assigns each object to all clusters with a certain probability.
- **Complete vs. Partial**
  - Complete: Every object is assigned to at least one cluster, ensuring full coverage of data points.
  - Partial: An object might not belong to any cluster, allowing for the presence of noise and outliers.

## K-Means

- **Basic Characteristics**
  - Clusters are centroid-based.
  - Clustering is partitional, exclusive, and complete.
- **Inputs (for d-dimensional Euclidean space)**
  - Data points:  $(x_1, x_2, \dots, x_N)$ ,  $x_i \in \mathbb{R}^d$
  - Number of clusters:  $K \rightarrow C_1, C_2, \dots, C_K$  (cluster centers).
- **Optimization Objective**
  - Minimize Sum of Squared Error (SSE):  $SSE = \sum_{i=1}^K \sum_{x \in C_i} \|x - c_i\|^2$
  - Finding the optimal solution is NP-hard:  $O(N^{Kd+1})$ .

#### K-Means — How to Define the Centroid of a Cluster?

##### Simple case in Euclidean space

The centroid is derived by minimizing SSE, which turns out to be the mean of all points in that cluster. Proof:

$$\begin{aligned} SSE &= \sum_{i=1}^K \sum_{x \in C_i} \|x - c_i\|^2 \\ \frac{\delta}{\delta c_k} SSE &= \frac{\delta}{\delta c_k} \sum_{i=1}^K \sum_{x \in C_i} \|x - c_i\|^2 \\ &= \sum_{x \in C_k} \frac{\delta}{\delta c_k} \|x - c_k\|^2 \\ &= \sum_{x \in C_k} 2(x - c_k) \end{aligned}$$
$$\begin{aligned} \sum_{x \in C_k} 2(x - c_k) &= 0 \\ \sum_{x \in C_k} x - \sum_{x \in C_k} c_k &= 0 \\ m_k c_k &= \sum_{x \in C_k} x \\ c_k &= \frac{1}{m_k} \sum_{x \in C_k} x \end{aligned}$$

#### K-Means — Basic Algorithm (Lloyd's Algorithm)

- **Initialization**
  - Select  $K$  points as initial centroids:  $C_1, C_2, \dots, C_K$ .
- **Repeat**
  - **Assignment:** Assign each point to the nearest cluster (i.e., centroid).
  - **Update:** Move each centroid to the average of its assigned points.
- **Stopping Criterion**
  - Repeat until no change in assignments.

#### Handling Empty Clusters

- **Artificially fill empty clusters after assignment step**
  - Replace empty cluster with the point that contributes most to SSE.
  - Replace empty cluster with a point from the cluster with the highest SSE.

- **Postprocessing**
  - Split "loose" clusters with very high SSE.
- **Modification of Lloyd's Algorithm**
  - K-Means variants aim to address the initial centroids issue.

#### K-Means — Limitations and Potential Workarounds

- **Susceptibility to "Natural" Clusters**
  - K-Means struggles with:
    - \* Non-spherical clusters.
    - \* Clusters of different sizes.
    - \* Clusters of different densities.
  - **Potential Workaround: Choose a Larger Value for K**
    - \* Split natural clusters into multiple "well-behaved" subclusters.
    - \* Apply suitable postprocessing steps to merge subclusters.
- **Sensitivity to Initial Centroid Selection**
  - Different initializations of centroids may yield:
    - \* Different clusterings with varying SSEs (leading to local optima).
    - \* Empty clusters if a centroid is "blocked off" by other centroids.
  - **Potential Workaround: Improve Centroid Initialization**
    - \* **Artificially Fill Empty Clusters After Assignment**
      - Replace empty cluster with the point that contributes most to SSE.
      - Replace empty cluster with a point from the cluster with the highest SSE.
    - \* **Postprocessing**
      - Split "loose" clusters with very high SSE.
    - \* **Modification of Lloyd's Algorithm (K-Means Variants)**
      - Use improved centroid initialization techniques.

#### K-Means Variants

- **K-Means++**
  - Only changes the initialization of centroids.
  - Goal: Spread out centroids for better performance with theoretical guarantees.
  - **Initialization Process:**
    - \* Pick a random point as the first centroid  $C_1$ .
    - \* Repeat:
      - For each point  $x$ , calculate distance  $d_x$  to the nearest existing centroid.
      - Pick a random point for the next centroid with probability proportional to  $d_x^2$ .
    - \* Until  $K$  centroids have been picked.
- **X-Means**
  - Automatic method to choose  $K$ .
  - Iteratively applies K-Means with  $K = 2$  to refine clustering.
  - **Example Scoring Functions:**
    - \* Bayesian Information Criterion (BIC).
    - \* Akaike Information Criterion (AIC).
    - \* Minimum Description Length (MDL).
- **K-Medoids**
  - **Restriction:** Centroids are chosen from the data points.
    - \* Does not require calculation of averages.
    - \* Uses only a notion of distance or similarity.
    - \* More robust to noise and outliers.
  - **Main Issue: Performance**
    - \* More expensive update step.
    - \* Swap medoid with each point in the cluster and calculate change in cost (e.g., SSE).
    - \* Choose the point as the new medoid that minimizes cost after swapping.

## DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

#### Basic Characteristics

- **Clusters:** Density-based
- **Clustering:** Partitional, Exclusive, Partial

#### Inputs (for d-dimensional Euclidean space)

- $\mathbf{x} = (x_1, x_2, \dots, x_N)$ ,  $x_i \in \mathbb{R}^d$
- $\epsilon$  (Epsilon): Radius defining a point's neighborhood
- $MinPts$ : Minimum number of points in a neighborhood

$$density = \frac{mass}{volume} = \frac{MinPts}{\epsilon^d}$$

#### Types of Points in DBSCAN

- **Core points**
  - Points with at least  $MinPts$  neighbors within radius  $\epsilon$
  - Form the **interior** of a cluster
- **Border points**
  - Non-core points with at least one core point in their neighborhood
  - Form the **border** of a cluster
- **Outliers / Noise**
  - All other points
  - Default node type

## Algorithm 1 DBSCAN Algorithm

```
1: Input: Dataset  $D$ , radius  $\epsilon$ , minimum points  $MinPts$ 
2: Output: Clusters and outliers
3: for each point  $P$  in  $D$  do
4:   if  $P$  is already visited then
5:     Continue
6:   end if
7:   Mark  $P$  as visited
8:   Retrieve all neighbors  $N$  of  $P$  within radius  $\epsilon$ 
9:   if  $|N| < MinPts$  then
10:    Mark  $P$  as noise
11:  else
12:    Create a new cluster
13:    Expand cluster by adding density-reachable points
14:  end if
15: end for
```

#### Characteristics of DBSCAN

- **DBSCAN always converges**
  - Every data point is explored in either Phase 1 or Phase 2
  - A data point does not change its type (exception: noise  $\rightarrow$  border)
- **DBSCAN is not completely deterministic**
  - Phase 1 introduces randomness
  - Border points may be reachable from core points of different clusters
  - Noise and core points are deterministic

#### DBSCAN — Limitations

- DBSCAN cannot handle different densities.
- DBSCAN is generally very sensitive to parameters. Choosing  $\epsilon$  and  $MinPts$  requires a good understanding of data and context.

#### DBSCAN — How to Choose Parameter Values?

- Informed by results of EDA, e.g.:
  - Distribution of all pairwise distances.
  - First insights into suitable values for  $\epsilon$ .
- Density of data points has intuitive semantic meaning, e.g.:
  - Geographic distance between bars in a city.
  - Task: Find areas (clusters) with more than 10 bars within 500m.

## Hierarchical Clustering

### Basic characteristics

- Clusters: depends...
- Clustering: hierarchical, complete, exclusive (at each level!)

### No parameterization (in principle)

- In practice, typically the number of clusters is specified (similar to K-means)
- Different choices of measures to calculate distances between clusters

### Dendrograms

- A dendrogram is a visualization of hierarchical relationships
- Binary tree showing how clusters are hierarchically merged/split
- Each node represents a cluster
- Each leaf is a singleton cluster
- Height reflects distance between clusters

### Two Main Types of Hierarchical Clustering

- **Agglomerative (bottom-up)**
  - Start with each point being its own cluster
  - Merge the closest pair of clusters at each step
  - Stop when only one cluster remains
  - Example: **AGNES (Agglomerative Nesting)**
- **Divisive (top-down)**
  - Start with all points in a single cluster
  - Recursively split clusters at each step
  - Stop when each cluster contains a single point
  - Example: **DIANA (Divisive Analysis)**

## AGNES Algorithm

### Algorithm 2 AGNES Algorithm

```

1: Input: Dataset  $D$ 
2: Output: Hierarchical clusters
3: for each point  $P$  in  $D$  do
4:   Assign  $P$  to its own cluster
5: end for
6: while more than one cluster exists do
7:   Merge the two closest clusters into one
8: end while

```

### AGNES Algorithm Linkages

- **Single Linkage Clustering**
  - Distance between clusters = **minimum distance** between two points from each cluster:  
 $d_{\text{single}}(C_i, C_j) = \min_{p \in C_i, q \in C_j} d(p, q)$
  - **Strength:** Can handle non-globular shapes.
  - **Weakness:** Very susceptible to noise (*Chaining Effect*).
- **Complete Linkage Clustering**
  - Distance between clusters = **maximum distance** between two points from each cluster:  
 $d_{\text{complete}}(C_i, C_j) = \max_{p \in C_i, q \in C_j} d(p, q)$
  - **Strength:** Less susceptible to noise or outliers.
  - **Weakness:** Bias towards globular clusters, breaks large clusters.
- **Average Linkage Clustering**
  - Distance between clusters = **average distance** between two points from each cluster:  
 $d_{\text{average}}(C_i, C_j) = \text{avg}_{p \in C_i, q \in C_j} d(p, q)$
- **Centroid Linkage**
  - Distance between clusters = distance between centroids:  
 $d_{\text{centroid}}(C_i, C_j) = d(m_i, m_j)$
- **Ward Linkage**
  - Variance-based merging criterion:

$$d_{\text{ward}}(C_i, C_j) = \sum_{k \in C_i \cup C_j} \|x_k - m_{ij}\|^2 - \sum_{k \in C_i} \|x_k - m_i\|^2 - \sum_{k \in C_j} \|x_k - m_j\|^2$$

### AGNES Complexity Analysis

- **Space Complexity:**  $O(N^2)$  (storing distance matrix)
- **Time Complexity:**
  - Baseline:  $O(N^3)$  - (N-1) steps, each step  $O(N^2)$
  - Using heap/priority queue:  $O(N^2 \log N)$
  - Single Linkage special optimization:  $O(N^2)$

## DIANA Algorithm

- **Top-Down Hierarchical Clustering**
  - Start with all points in a single cluster
  - Recursively split clusters until each point is its own cluster
- **Challenge:**  $2^n$  ways to split a cluster with  $n$  points
  - Heuristics required to restrict the search space
  - Generally slower and less common than AGNES
- **Cases where DIANA is preferable:**
  - When no complete clustering is needed (early stopping)
  - When splitting can use global knowledge

## Cluster Evaluation

### Cluster Evaluation Problems

- Challenges in Visual Inspection: Just eyeballing the clustering is rarely possible.
- Algorithmic Bias: Clustering algorithms will always find some clusters.

### Purpose of Cluster Evaluation

- Comparing the results of different clustering algorithms.
- Comparing the results of a clustering algorithm with different parameters.
- Minimizing the effects of noise on the clustering.

### Two Main Approaches of Clustering Evaluation

- **External quality measures:** Evaluate clustering against a ground truth (if available).
- **Internal quality measures:** Evaluate clustering from the data itself.

### External Quality Measures

- **Ground Truth: Labeled Data**
  - Labels indicate that two points "belong together."
  - If clustering reflects this, it is considered good clustering.
- **Cluster Purity**
  - Measures how pure clusters are concerning the most common label.
  - Formula:  $P = \frac{1}{N} \sum_{c \in C} \max_{l \in L} |c \cap l|$
  - Example Calculation:  $P = \frac{1}{8}(3 + 4) = 0.875$
  - **Limitation:** Purity does not penalize having many clusters.
  - $P = 1$  is easy to achieve if each cluster contains only a single point.
- **Information Retrieval Metrics**
  - **True Positives (TP):** Same cluster, same label.
  - **True Negatives (TN):** Different clusters, different labels.
  - **False Positives (FP):** Same cluster, different labels.
  - **False Negatives (FN):** Different clusters, same label.
- **Rand Index (RI)**
  - Reflects clustering accuracy.
  - Formula:  $RI = \frac{TP+TN}{TP+FP+FN+TN}$
  - Example:  $RI_{\text{example}} = 0.82$
- **Precision, Recall, and F1-Score**
  - Precision:  $P = \frac{TP}{TP+FP}$
  - Recall:  $R = \frac{TP}{TP+FN}$
  - F1-Score:  $F1 = \frac{2 \cdot P \cdot R}{P+R}$
  - Example values:  $P_{\text{example}} = 0.75$ ,  $R_{\text{example}} = 0.69$ ,  $F1_{\text{example}} = 0.72$

### Internal Quality Measures

- **Sum of Squared Errors (SSE)**
  - SSE is often used to determine the number of clusters.
  - Formula:  $SSE = \sum_{i=1}^k \sum_{x \in C_i} \|x - \mu_i\|^2$
  - **Limitations:**
    - \* Does not penalize having a large number of clusters.
    - \* SSE decreases as the number of clusters increases.
    - \* Applicable beyond K-Means but less intuitive for non-globular clusters.
  - SSE can also be used for complex data but inherently favors globular clusters.
- **Silhouette Coefficient**
  - A good clustering has:
    - \* High inter-cluster distances.
    - \* Low intra-cluster distances.
  - Defined for each data point  $x$ :
    - \* **Cohesion**  $a(x)$ : average distance to points in the same cluster.
    - \* **Separation**  $b(x)$ : minimum average distance to points in a different cluster.
    - \* **Silhouette Score:**

$$s(x) = \frac{b(x) - a(x)}{\max\{a(x), b(x)\}}, \quad \text{if } |C_x| > 1$$
  - \* If  $|C_x| = 1$ , then  $s(x) = 0$ .
  - The overall \*\*Silhouette Coefficient\*\* for the clustering:

$$SC = \frac{1}{N} \sum_{i=1}^N s(x_i)$$

- Interpretation:  $-1 \leq s(x) \leq 1$  where \*\*negative values indicate poor clustering\*\* and \*\*values closer to 1 indicate well-separated clusters\*\*.

### Pragmatic Considerations

- The choice of the "best" clustering is often pragmatic:
  - Fixed number of clusters (problematic for DBSCAN).
  - Parameters defined by tasks (e.g., "areas with more than 5 McDonalds within 500m").
  - Maximum, minimum, or average size of clusters.
  - Focus on individual clusters instead of the whole clustering (e.g., largest/smallest cluster).
  - Setting  $k$  too high and merging later if needed.

## Association Rule Mining

### Basic Setup

- **Input database:**
  - Set of **transactions**
  - Transaction = set of **items**
- **Output: Association Rules**
  - Rules predicting the occurrence of some items based on the occurrence of other items.

Example Association Rules

- Antecedent → Consequent
- {item<sub>2</sub>, item<sub>3</sub>} → {item<sub>5</sub>}
- {item<sub>1</sub>} → {item<sub>3</sub>}

Problem Statement

- Association rules are not "hard"rules
  - Example: {cereal} → {milk} does not mean customers always buy milk with cereal.
  - Each possible combination (e.g., {yogurt, bread} → {milk}) is a potential rule.
- Given  $d$  unique items, the number of possible rules is:  $3^d - 2^{d+1} + 1$
- Example: If  $d = 6$ , then there are 602 possible rules.

Applications of Association Rule Mining

- Market Basket Analysis: Understanding customer shopping behavior.
- Medical Data Analysis: Diagnosis Support Systems and ADR discovery (adverse drug reaction)
- Census Data Analysis: Getting insights into a population.
- Behavior Data Analysis: User preferences & linkings.

Association Rule Definitions

- **Itemset**: A subset of items.
- **K-itemset**: An itemset containing  $k$  items.
- **Support Count (SC)**: Number of transactions containing an itemset.
- **Support (S)**: Fraction of transactions containing an itemset.
- **Frequent Itemset**: An itemset with a support greater or equal to a minimum threshold  $minsup$ .
- **Association Rule**: An implication expression  $X \rightarrow Y$ , where  $X$  and  $Y$  are itemsets.
- **Support of an Association Rule**: Fraction of transactions containing all items in an association rule  $X \rightarrow Y$ :

$$S(X \rightarrow Y) = \frac{SC(X \cup Y)}{N} = S(X \cup Y)$$

- **Confidence of an Association Rule**: Probability of  $Y$  given  $X$ :

$$C(X \rightarrow Y) = \frac{S(X \rightarrow Y)}{S(X)} = \frac{S(X \cup Y)}{S(X)}$$

Support-Confidence Combinations

- **Low Support, Low Confidence**: Items in  $(X \cup Y)$  do not frequently appear together. Even when items in  $X$  appear together, they often do so without items in  $Y$ .
- **High Support, Low Confidence**: Items in  $(X \cup Y)$  frequently appear together, but if items in  $X$  appear together, they often do so without items in  $Y$ .
- **Low Support, High Confidence**: Items in  $(X \cup Y)$  do not frequently appear together, but when items in  $X$  appear together, they often do so with items in  $Y$ .
- **High Support, High Confidence**: Items in  $(X \cup Y)$  frequently appear together, and if items in  $X$  appear together, they often do so with items in  $Y$ .

Brute Force Algorithm

- **Finding Association Rules**: Given a set of transactions, find all association rules  $X \rightarrow Y$  with:
  - Support  $S(X \rightarrow Y) \geq minsup$
  - Confidence  $C(X \rightarrow Y) \geq minconf$
- **Brute Force Algorithm**:
  - List all possible association rules  $X \rightarrow Y$ .
  - Calculate support  $S(X \rightarrow Y)$  and confidence  $C(X \rightarrow Y)$  for each rule.
  - Drop rules where  $S(X \rightarrow Y) < minsup$  and  $C(X \rightarrow Y) < minconf$ .

Algorithm 3 Brute Force Association Rule Mining

- 1: **Input**: Transaction database  $D$ , minimum support  $minsup$ , minimum confidence  $minconf$   
2: **Output**: Set of association rules  $R$   
3: Initialize an empty set  $F$  for frequent itemsets  
4: Initialize an empty set  $R$  for valid rules

▷ Frequent Itemset Generation

- 5: **for** each transaction  $t$  in  $D$  **do**  
6:   **for**  $k = 1$  to  $|t|$  **do**  
7:     Generate all possible  $k$ -itemsets  
8:     **for** each itemset  $I$  in  $k$ -itemsets **do**  
9:       Count occurrences of  $I$   
10:     **end for**  
11:   **end for**  
12: **end for**  
13: Retain only itemsets with support  $\geq minsup$

▷ Rule Generation

- 14: **for** each frequent itemset **do**  
15:   Generate all possible rules  $X \rightarrow Y$   
16:   Compute  $S(X \rightarrow Y)$  and  $C(X \rightarrow Y)$   
17:   **if**  $S(X \rightarrow Y) \geq minsup$  and  $C(X \rightarrow Y) \geq minconf$  **then**  
18:     Add  $X \rightarrow Y$  to  $R$   
19:   **end if**  
20: **end for**  
21: **return**  $R$

Computational Complexity:

- Given  $d$  unique items, the number of possible rules is:

$$3^d - 2^{d+1} + 1 \in O(3^d)$$

- Let  $w$  be the maximum number of items in a transaction within the database:

$$O(N \cdot (3^w - 2^{w+1} + 1))$$

where typically  $w \ll d$ .

- Frequent itemset generation complexity:

$$O(N \cdot 2^w)$$

Observation: Decoupling Support and Confidence:

- A rule  $X \rightarrow Y$  has sufficient support only if  $X \cup Y$  is a frequent itemset.
- No need to calculate confidence of rules where  $X \cup Y$  is not a frequent itemset.

Apriori Algorithm

- **Apriori Principle (Anti-Monotonicity)**:
  - If an itemset  $X$  is "not frequent", then any superset  $Y \supset X$  is "also not frequent".
  - If an itemset  $Y$  is "frequent", then all its subsets  $X \subset Y$  must also be "frequent".
- **Apriori Algorithm**:
  - Iteratively generate candidate frequent itemsets.
  - Use previously found frequent  $(k - 1)$ -itemsets to "prune" candidate  $k$ -itemsets.
  - Filter frequent itemsets based on the minimum support threshold.

Algorithm 4 Apriori Algorithm for Frequent Itemset Mining

- 1: **Input**: Transaction database  $D$ , minimum support  $minsup$   
2: **Output**: Frequent itemsets  $F$   
3: Initialize  $F_1$  with frequent 1-itemsets

▷ Frequent Itemset Generation

- 4: **for**  $k = 2$  to max itemset length **do**  
5:   Generate candidate  $k$ -itemsets  $L_k$  from  $F_{k-1}$   
6:   Prune infrequent itemsets from  $L_k$   
7:   **for** each transaction  $t$  in  $D$  **do**  
8:     **for** each candidate itemset  $I$  in  $L_k$  **do**  
9:       Count occurrences of  $I$   
10:     **end for**  
11:   **end for**  
12:   Retain only itemsets with support  $\geq minsup$  as  $F_k$   
13:   **if**  $F_k$  is empty **then**  
14:     **break**  
15:   **end if**  
16: **end for**

▷ Rule Generation

- 17: **for** each frequent itemset **do**  
18:   Generate all possible rules  $X \rightarrow Y$   
19:   Compute  $S(X \rightarrow Y)$  and  $C(X \rightarrow Y)$   
20:   **if**  $C(X \rightarrow Y) \geq minconf$  **then**  
21:     Add  $X \rightarrow Y$  to  $R$   
22:   **end if**  
23: **end for**  
24: **return**  $F$

- **Rule Generation**:
  - Generate association rules  $X \rightarrow Y$  from frequent itemsets.
  - Compute confidence:

$$C(X \rightarrow Y) = \frac{S(X \cup Y)}{S(X)}$$

- Retain rules where  $C(X \rightarrow Y) \geq minconf$ .
- **Computational Complexity**:
  - Generates at most  $2^d - 1$  itemsets for  $d$  unique items.
  - Reduces complexity by "pruning" infrequent itemsets early.
  - Worst-case complexity:

$$O(N \cdot 2^w)$$

where  $N$  is the number of transactions and  $w$  is the largest transaction size.

- **Advantages**:
  - Reduces the number of candidate itemsets compared to brute-force.
  - Efficient for "dense" datasets with many frequent itemsets.
- **Limitations**:
  - "Computationally expensive" when itemsets are very large.
  - "Many database scans required", increasing execution time.