Data Eng. Notes

ibrahim.nasser@fau.de

Disclaimer

These are my personal notes and are not official course documents. They may contain inaccuracies or omissions, hence, they should not be considered as a substitute for official course materials or as comprehensive preparation for examinations.

Introduction and Basic Data Types

We call the data **Tabular** when there are no modelled dependencies between attributes, for example, demographic attributes such as age, gender, ZIP code, etc. (also called *Nondependency-Oriented Data*). Otherwise it is **Non-Tabular**, e.g. social networks, time series, etc.

Matrix Representation of Data

A set $X = \{X_i \mid i \in \{1 \dots n\}\}$, with n records (samples) is a d-dimensional dataset iff each sample X_i is a set of $\{x_j \mid j \in \{1 \dots d\}\}$ attributes (features). X is tabular if it is invariant w.r.t shuffling of samples and features. Each feature x_j has its own domain $\mathcal{D}_j$

Quantitative vs. Categorical: A variable x is quantitative (numeric) if its domain $\mathcal{D}_x$ is numeric. Otherwise, Categorical. *Examples (Q):* age, weight, height, BMI, Date of Birth. *Examples (C):* name, gender, country, ZIP Code, weather, ID, day.

Nominal vs. Ordinal: A categorical variable x is ordinal if its domain $\mathcal{D}_x$ has a natural ordering. Otherwise, Nominal. *Examples (N):* weather, name, gender, country, ZIP, ID, day *Examples (O):* heat level, textual gpa.

Finite vs. Infinite: A variable x has a finite domain iff $|\mathcal{D}_x| = N, N \in \mathbb{N}$. Otherwise, Infinite. *Examples (F):* age (years), country, ZIP, ID, gender, day. *Examples (I):* BMI, height, Date of Birth.

> Note
>
> All categorical variables have finite domains, not the other way around.

Discrete vs. Cont.: A Quantitative variable x is continuous iff $\forall z, y \in \mathcal{D}_x \exists w \in \mathcal{D}_x, z < w < y$. Otherwise, Discrete. *Examples (D):* age (years, months, days, hours, etc). *Examples (C):* age (unitless, number), Date of Birth (point in cont. time), BMI.

> Note
>
> By **rounding** quantitative data, we can transform cont. domains into discrete ones.

> Note
>
> Age is quantitative finite discrete if it is computed as whole years, months, days, hours. However, it is quantitative infinite continuous it is computed as precise value including fractions

> Note
>
> Date of Birth is quantitative infinite continuous since it is a point in a continuous endless time

Binary: We call a variable x binary iff $|\mathcal{D}_x| = 2$

Temporal: We call a variable x temporal iff $\mathcal{D}_x$ represents time points or intervals. *Examples:* day, month, Date of Birth

Encoding: Data Encoding refers to the technique of converting data into a form that allows it to be properly used by different systems.

Binning: Binning is an encoding technique that is a function $f : \mathcal{D} \to \{1 \dots K\}$

Example: Equal-Width Binning: Size (width) of each bin is calculated as $W = \frac{\text{Max}(x) - \text{Min}(x)}{K}$ where K is the number of bins.

One-Hot Encoding

To mitigate the problem of label encoding for nominal variables.
How? Create a fixed-size vector with size = |unique(x)|, where each position corresponds to a unique category value. Assign a `1` to the position representing the category and `0`s elsewhere.

Example: Suppose unique(x) = {Red, Green, Blue}

- Red → [1, 0, 0]
- Green → [0, 1, 0]
- Blue → [0, 0, 1]

> Note
>
> One-hot encoding avoids the problem of implying ordinal relationships. However, it increases dimensionality significantly, especially when the number of categories is large (curse of dimensionality).

Cyclic Encoding

Some categorical variables are *ordinal* and have a natural *cyclic* structure. A classic example is the months of the year:

$$\mathcal{D}_x = \{\text{Jan}, \text{Feb}, \ldots, \text{Dec}\}$$

This variable has both an order (Jan < Feb < ... < Dec) and a cyclic relationship (Dec is followed by Jan).

To encode this properly, we use the index i of each category in the ordered list, where $i = 1, 2, \ldots, k$, and k is the total number of categories.

Encoding Function:

$$\text{enc}(c_i) = (x_i, y_i)$$

$$x_i = \cos\left(\frac{2\pi(i-1)}{k}\right), \quad y_i = \sin\left(\frac{2\pi(i-1)}{k}\right)$$

This maps each category to a unique point on the unit circle, preserving both order and cyclicity.

> Note
>
> Cyclic encoding is useful when the first and last categories are conceptually adjacent (e.g., December and January). This is not possible with standard label or one-hot encoding.

Optional: Normalize to Unit Square

$$\text{enc}(c_i) = \left(\frac{x_i + 1}{2}, \frac{y_i + 1}{2}\right)$$

This scaled version maps points to the square $[0, 1] \times [0, 1]$, which can be useful when input normalization is required for machine learning models. Note that this transformation alters the original unit circle geometry.

> Note
>
> Use raw unit circle encoding when preserving angular distance is important. Use the normalized version when the model expects features in the range $[0, 1]$.

Non-Tabular Data

Such as Spatial data, images, time series, string, graphs.

A set $X = \{x_i \mid i \in \{1 \ldots n\}\}$ is a d-dimensional **spatial** dataset with n samples if each sample x_i contains a set of $\{x_j \mid j \in \{1 \ldots d\}\}$ features AND each data point x_{ij} is associated with a specific spatial location l.

A spatial location l can be a point $(l_x, l_y) \in \mathbb{R}^2$ (2D spatial data) or $(l_x, l_y, l_z) \in \mathbb{R}^3$ (3D spatial data), etc.

Tokenization (Character-Level)

Tokenization is the process of converting raw text into smaller units called tokens. In character-level tokenization, each unique character from the corpus is treated as a token.

Example: Consider the corpus consisting of a single sentence: `"hi ai"`

- Unique characters: {`h`, `i`, ` `, `a`}
- Assign token IDs: `h:0, i:1, :2, a:3`
- Tokenized sentence: `"hi ai"` → `[0, 1, 2, 3, 1]`

Each character in the sentence is replaced by its corresponding token ID.

Graphs

A graph is a mathematical structure used to model pairwise relations between objects.

- A graph G is defined as $G = (V, E)$, where:
  - V is a set of *vertices* (or *nodes*).
  - $E \subseteq V \times V$ is a set of *edges*.

Types of Graphs:

- **Undirected Graph:** An edge $(u, v) \in E$ implies a bidirectional connection:

  $$(u, v) \in E \Rightarrow (v, u) \in E$$

- **Directed Graph (Digraph):** Edges have direction:

  $$(u, v) \in E \not\Rightarrow (v, u) \in E$$

Graph Representations

Adjacency Matrix:

A $|V| \times |V|$ matrix A, where:

$$A[u][v] = \begin{cases} 1 & \text{if } (u, v) \in E \\ 0 & \text{otherwise} \end{cases}$$

- **Space consumption:** $\mathcal{O}(|V|^2)$
- **Edge access:** $\mathcal{O}(1)$
- **Neighbor iteration:** $\mathcal{O}(|V|)$

Adjacency List:

Each vertex $u \in V$ maintains a list of its neighbors.

- **Space consumption:** $\mathcal{O}(|V| + |E|)$
- **Edge access:** $\mathcal{O}(|V|)$ (worst-case search)
- **Neighbor iteration:** $\mathcal{O}(\deg(u))$, where $\deg(u)$ is the degree of vertex u

Weighted Graphs:

In some graphs, each edge $(u, v) \in E$ is associated with a numerical value called a *weight*, often representing cost, distance, capacity, etc.

- For weighted graphs, the edge set becomes:

  $$E \subseteq V \times V \times \mathbb{R}$$

  or we define a weight function:

  $$w : E \to \mathbb{R}$$

- In the adjacency matrix, $A[u][v]$ stores the weight instead of a binary 0 or 1.
- In the adjacency list, each neighbor can be stored along with its edge weight as a tuple: $(v, w(u, v))$.

Conceptual Modeling

ER Model: Entity-Relationship Model is a high-level, conceptual framework to describe entities, their attributes, and the relationships between them.

Entity: Basic concept of the Entity-Relationship (ER) model. It is an object in the real world. E.g. e1 (some employee).

Attribute: Entities have attributes that are the properties that describe them.

Entity Type: All entities that have the same entity type share the same attributes. E.g. EMPLOYEE (type), e1 (Entity).

Attribute Value: A particular entity has a specific value for each of its attributes.

Composite: An attribute is composite if it is described in terms of its smaller parts. E.g. Name, some databases consider name as a composite attribute consisting of two **atomic** attributes First Name and Last Name.

Atomic/Simple: Cannot be divided into smaller parts.

Note

These days, we store **date** as a single value attribute of the type *DATE*. Earlier, date was considered as a composite attribute consisting of atomic attributes *day, month, year*.

Derived: An attribute is derived if its value is calculated using other **stored** attributes, e.g. age.

Stored: An attribute is stored if it cannot be derived from other attributes.

Note

Age is both derived and atomic. DateOfBirth is both composite and stored.

Single-Valued: An attribute is single-valued if it can have only one value. E.g. DateOfBirth is single-valued composite. Biological sex is single-valued atomic.

Multi-Valued: An attribute is multivalued if it can have several values. E.g. college degrees is multivalued atomic and can have BSc, MSc, BEng, etc.

Note

Affiliation of an entity type RESEARCHER is multivalued (because one can have different affiliations) and composite because an affiliation could be represented as (Org. Name, Dept., Address, Role, Start Date, End Date).

Entity Set: Collection of entities of a particular entity type in a database in a given time point.

Entity Type	Blueprint/Description
Entity Set	Actual set of entities (entity instances) at a point in time

Note

In ancient logic and philosophy we refer to the definition or conceptual content of a term as an *intension*. However, the set of actual things that satisfy a concept is called *extension*. Hence, Entity Type is called intension, Entity Set is called extension.

Candidate Key (Key Attribute): A candidate key is an attribute (or set of attributes) that **uniquely** and **minimally** identifies each entity in an entity set. E.g. StudentID, studentEmail.

Primary Key: A primary key is the chosen candidate key that will be used to uniquely identify entities in the database.

Foreign Key: A foreign key is an attribute in one table/entity that references the primary key of another table/entity. It expresses a relationship between two entity sets.

Composite Primary Key: A composite primary key is a primary key that consists of two or more attributes combined together to uniquely identify a record in a table. Neither attribute alone is sufficient to guarantee uniqueness — but together, they do.

This is common in relationship tables, for example: enrollment relationship between students and courses (M:M):

StudentID	CourseID	Grade
101	CS101	A
101	MATH201	B
102	CS101	B+

Weak Entity Types: Entity types without key attributes.

Strong Entity Types: Entity types with key attributes.

Relations

If we want to model 1:M or M:1 relations, we use the idea of foreign key (modeling the relation with single value attribute). Examples:

PersonID	PersonName
1	Alice
2	Bob

categoryID	categoryName
1	Sport
2	Science

catID	catName	ownerID
1	Daisy	1
2	Smart	2
3	Sweet	1

articleID	title	categoryID
1	title	1
2	title	2
3	title	1

We model M:M Relations by creating an entity representing that relation (usually with composite primary key).

Relationship Type: The definition / template / blueprint of the relationship

Relationship Instance: A single actual link between entities.

Relationship Set: The collection of all relationship instances at a given time.

Participation: We say that entity types $E_1 \dots E_n$ participate in the **relationship type** R.

Relationship Degree: Number of participating entity types in the relation. E.g. consider a relation SUPPLY that models which suppliers supply which projects and what parts are supplied, the degree here is 3 due the three entity types (SUPPLIER, PROJECT, PART).

Role Name: The name describing the part an entity plays in a relationship.

Recursive Relationship: A relationship where the same entity type participates more than once with different roles. E.g. SUPERVISION.

Cardinality Ratio: Specifies the maximum number of entities of one type that can be associated with an entity of another type in a relationship. Examples (E_1 BINARY_RELATION E_2):

Let E_1 and E_2 be the sets of entities of type E_1 and E_2, respectively. Let $R \subseteq E_1 \times E_2$ be the binary relation between them: $R = \{(e_1, e_2) \mid e_1 \in E_1,\ e_2 \in E_2\}$

- **1:1** each entity of type E_1 can be related to at most one entity of type E_2 and vice versa

  $$\forall e_1 \in E_1, \quad |\{e_2 \in E_2 \mid (e_1, e_2) \in R\}| \leq 1$$

  $$\forall e_2 \in E_2, \quad |\{e_1 \in E_1 \mid (e_1, e_2) \in R\}| \leq 1$$

- **1:M** one entity of type E_1 can be related to many entities of type E_2. Each entity of type E_2 can be related to at most one entity of type E_1

  $$\forall e_1 \in E_1, \quad |\{e_2 \in E_2 \mid (e_1, e_2) \in R\}| \geq 0$$

  $$\forall e_2 \in E_2, \quad |\{e_1 \in E_1 \mid (e_1, e_2) \in R\}| \leq 1$$

- **M:1** Inverse of **1:M**

  $$\forall e_1 \in E_1, \quad |\{e_2 \in E_2 \mid (e_1, e_2) \in R\}| \leq 1$$

  $$\forall e_2 \in E_2, \quad |\{e_1 \in E_1 \mid (e_1, e_2) \in R\}| \geq 0$$

- **M:M** many entities of type E_1 can be related to many entities of type E_2 and vice versa

  $$\forall e_1 \in E_1, \quad |\{e_2 \in E_2 \mid (e_1, e_2) \in R\}| \geq 0$$

  $$\forall e_2 \in E_2, \quad |\{e_1 \in E_1 \mid (e_1, e_2) \in R\}| \geq 0$$

Total vs. Partial Participation

- **Total Participation of** E_1 in R
  $\forall e_1 \in E_1,\ \exists e_2 \in E_2 : (e_1, e_2) \in R$ (Every entity of E_1 is related to at least one entity of E_2.)
- **Partial Participation of** E_1 in R
  $\exists e_1 \in E_1 : \forall e_2 \in E_2, (e_1, e_2) \notin R$ (There exists an entity in E_1 that does not participate in R.)

Migrating attributes from relations to entities:

- 1:1 relationship types: Attributes can be migrated to either participating entity. (e.g. EMP MANAGES DEPT, start_date)
- 1:N or N:1 relationship types: Attributes should be migrated to the entity that participates at most once. (e.g. EMP WORKS_FOR DEPT, start_date)
- M:N relationship types: Attributes cannot be migrated to the participating entities and must remain on the relationship itself.

Identifying Relationships: a special relationship where a weak entity is identified by its relationship with a strong entity. The weak entity cannot exist without the strong entity, and the relationship plays a crucial role in providing the weak entity with a composite key.

Existence Dependency: A weak entity depends on the strong entity for its existence. It cannot exist without being related to a strong entity.

> **Note**
>
> A weak entity type has total participation in its identifying relationship. This means that every instance of the weak entity must be associated with at least one instance of the strong entity. If it doesn't, the weak entity doesn't exist ("existence dependency").

Example:

- Consider we have the following strong entities, customer, product.
- To manage orders, we have two entities, order, orderItem.
- Order is a strong entity since each order has its ID
- However, OrderItem entity can have OrderID, LineNumber, ProductID, quantity, price, discount, etc.
- In this case, OrderItem entity is a weak one, it cannot exist unless an order exists, therefore, the primary key is composite (OrderID, LineNumber)

Min-Max Modeling: Given an entity E participating in Relation R. If at least min and at most max instances of E must participate in R with $min >= 0, max >= 1, max >= min$, then we say E respects min-max constraint (min, max) w.r.t R.

ER Diagram