Data Eng. Notes

ibrahim.nasser@fau.de

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 $X = \{X_i \mid i \in \{1 \ldots n\}\}$, with n records (samples) is a d-dimensional dataset iff each sample X_i is a set of $\{x_j \mid j \in \{1 \ldots d\}\}$ 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 $f : \mathcal{D} \to \{1 \ldots K\}$

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 = $|\text{unique}(x)|$, where each position corresponds to a unique category value. Assign a `1` to the position representing the category and `0`s elsewhere.

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

- 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:

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

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: $(v, w(u, v))$.