# CS-E4600 — Lecture notes #1
# Data representations for data mining problems

Aristides Gionis

September 16, 2018

One of the first questions that come when dealing with data from real-world applications is the issue of data representation. Chosing which *model* or *abstraction* to use in order to represent the available data can have implications on how one thinks about the process of discovering knowledge from the data. To a large extent, the choice of data representation depends on the application domain. However, as often in computer science, it is useful to strip out details and opt for general abstractions for which it is easy to reason about and develop standard methodologies.

Some of the most commonly used data representations are the following

- relational data,
- real-valued data,
- 0–1 data,
- transactional data,
- sequential data,
- time-series,
- graphs (or networks).

As we will see, there are connections between the different data representations, and it is often possible to represent the same dataset using a different abstraction.

**Relational data.** The data is represented in a *relational table* $M$. The table $M$ consists of $n$ *rows* $R = \{r_1, \ldots, r_n\}$ and $m$ *columns* $C = \{c_1, \ldots, c_m\}$. Each row represents a data *object* (also referred to as *record*, *item*, *point*, *sample*, *instance*, *entity*, etc.), and each column represents an *attribute* (also referred to as *feature* or *variable*). The number of objects $n$ is the *size* of the data, while the number of attributes $m$ is the *dimensionality* of the data.

The $M_{ij}$ entry of the table $M$ denotes the value of the object $r_i$ with respect to the attribute $c_j$. We consider that the values of the entries of an attribute $c_j$ have a predefined *domain*. This domain can be

- categorical: the attribute takes values in a discrete set (e.g., {`'high'`, `'med'`, `'low'`} or {`'red'`, `'blue'`, `'green'`});
- numerical: the attribute takes numerical values (e.g., the annual income of a person in euros, or the acidity of a chemical solution);
- mixed: the attribute takes a mix of categorical and numerical values.

**Example.** The relational data representation is very generic can be used in many applications. For example, we can use the relational model to represent all employees in a company. Each row corresponds to an employee and the attributes may hold information regarding `Name`, `DoB`, `Degree`, `JobTitle`, `Location`, etc.

**Real-valued data.** A relational table $M$ where the domain of all attributes is numerical is a *real-valued* dataset. In this case, $M$ can be seen as a matrix and it is convenient to apply tools and methods from linear algebra, for example, *spectral analysis* or *principal component analysis* (PCA). Furthermore, each data object $r_i$, $i = 1\ldots,n$, can be considered as a vector in the $m$-dimensional Euclidean space $\mathbb{R}^m$. Following this view, we can use our geometric intuition and apply geometric techniques to analyze the data. For instance, it is common to define the similarity of two objects $r_i$ and $r_k$ using the Euclidean distance

$$d(r_i, r_k) = \left( \sum_{j=1}^{m} |M_{ij} - M_{kj}|^2 \right)^{\frac{1}{2}}.$$

**Example.** Real-valued representation is again very general and widely applicable. As one example, we may use it to keep information about user preferences regarding movies. Row $r_i$ corresponds to the $i$-th user, and column $c_j$ corresponds to the $j$-th movie. The entry $M_{ij}$ may take values in $\{0, 1, \ldots, 5\}$ indicating the extent at which user $r_i$ likes movie $c_j$ (1: does not like it at all; ...; 5: like it very much), while assuming that the value 0 indicates the fact that the user $r_i$ has not rated the movie $c_j$.

**Binary data.** A binary dataset (or 0–1 dataset) is a special case of a real-valued dataset, where the domain of all attributes is the set $\{0, 1\}$. According to the geometric intuition introduced above, each point $r_i$ is now a corner in the $m$-dimensional *hypercube* $\{0, 1\}^m$.

In many application scenarios, the number of 1's in the data is much smaller than the number of 0's. This asymmetry is known as *sparsity*, and we say that real datasets are *sparse*. To improve the efficiency of our methods, we often need to explicitly take into account the sparsity of the data. For example, for sparse data it is more space-efficient to store the data matrix $M$ using a sparse matrix representation. In this way, the space required to store the data is $\mathcal{O}(s)$, where $s$ is the number of 1's in the data. This should be contrasted with $\mathcal{O}(nm)$, the space required for a full matrix representation. In many cases, each data object has only a small (e.g., constant) number of 1's, so $s = \mathcal{O}(n)$.

**Example.** Consider a dataset recording the movies that different persons have *seen*. Row $r_i$ corresponds to the $i$-th person, and column $c_j$ corresponds to the $j$-th movie. The value of $M_{ij}$ is 1 if person $r_i$ has seen the movie $c_j$, and 0 otherwise. To get a sense of the sparsity of the data, consider that there are hundreds of thousands of movies (columns), while a typical person has only seen a few tens or hundreds of them. There may be a few persons who have seen many more movies than a typical person, e.g., film critics may have watched tens of thousands of movies. Such skewed distributions are common in real data, however, because there are only few rows with many 1's, the data remain sparse despite the skewness.

**Transactional data.** In transactional data we consider an underlying set of items $X = \{x_1, \ldots, x_m\}$. A transaction $s_i$ is then defined to be a subset of the underlying set of items, that is, $s_i \subseteq X$. The dataset $S$ is defined as a collection of $n$ transactions, that is, $S = \{s_1, \ldots, s_n\}$.

It should be immediate that the transactional data model is equivalent to the 0–1 data model: the transactions $\{s_1, \ldots, s_n\}$ correspond to the rows of a 0–1 matrix $M$, and the items $\{x_1, \ldots, x_m\}$

correspond to the columns of $M$. Furthermore, an entry $M_{ij}$ of the matrix $M$ is 1 if and only if, in the corresponding transactional dataset $S$ the item $x_j$ belongs in the transaction $s_i$.

As with 0–1 data, transactional data are used as a representation model in many applications. For a few examples, consider representing the set of movies that users have watched, the set of products that customers buy in a visit to their local supermarket, the set of urls that web users visit, the set of keywords that on-line advertisers are bidding, and so on.

**Time-series data.** A time-series $\mathbf{t} = \langle (t_1, v_1), \ldots, (t_m, v_m) \rangle$ is a sequence of pairs $(t_j, v_j)$, where $v_j$ is a numerical value and $t_j$ is a time-stamp.

We use the notation $\langle \ldots \rangle$ to emphasize the fact that the sequence is ordered, as opposed to the notation $\{\ldots\}$, which is used to denote unordered sets. The number of elements $m$ of the time-series is the *length* of the time-series. In practice the ordering is imposed by time, i.e., the time-series $\mathbf{t}$ represents measurements over time, and the value $v_j$ records a measurement at time $j$.

Time-series data are used to represent observations of a variable that changes over time, for instance, the price of a commodity in the market, the unemployment rate of a country, the $CO_2$ pollution level in a city, the average daily temperature in a geographic location, etc.

All previous examples are cases of *one-dimensional* time-series. In the general case, the observed variable can be multi-dimensional, giving rise to a *multi-dimensional* time-series, denoted by $\mathbf{t} = \langle (t_1, \vec{v}_1), \ldots, (t_m, \vec{v}_m) \rangle$. As an example, a multi-dimensional time-series can be used to represent the unemployment rate over time in all European countries.

Much of our discussion on time-series, e.g., all distance measures that we will focus in the next lectures, carries over in the multi-dimensional case easily, however, for simplicity we mostly consider one-dimensional series.

A time-series is *evenly-spaced* (or *equally-* or *regularly-spaced*) if the time stamps $t_j$ are evenly spaced in time. In such a case we can write $t_j = \delta j + \alpha$, where $\delta$ is a constant time interval that a new measurement is recorded. For evenly-spaced time-series we can omit the time-stamp information and simply represent them by $\mathbf{t} = \langle v_1, \ldots, v_m \rangle$. In most cases we work with evenly-spaced time-series. The reason is that time-series data are typically recorded by some automated process that includes a regular timer. For instance, imagine a sensor-based instrument for monitoring air quality, which records, say, the carbon-monoxide level in the ambient air every 30 secs.

Unless specified otherwise, in the rest of our discussion we consider evenly-spaced time-series.

Consider a set $T$ of $n$ time-series $\{\mathbf{t}_1, \ldots, \mathbf{t}_n\}$, and assume that all time-series in $T$ have the same length $m$. It is easy to see that the dataset $T$ can be organized as an $n \times m$ matrix $M$, with the $i$-th row corresponding to the time-series $\mathbf{t}_i$, and the $j$-th column corresponding to the $j$-th values of the set of time-series. Thus, under the assumption that all time-series have the same length, a time-series dataset can be represented as a real-valued matrix.

One should note, however, that due to the time ordering of the time-series, there is no exact equivalence between a set of time-series and a matrix: indeed, while the values of the time-series are ordered, there is no ordering imposed on the columns of a matrix. Nevertheless, sometimes it is convenient to represent a set of time-series as a matrix and take advantage of linear algebra and matrix-analysis techniques. There are also cases that such a matrix representation is not particularly useful. This happens, for instance, when the time-series have different length, or when we are interested in discovering patterns that relate different time periods of the time-series, e.g., compare two time-series after shifting the one by a certain time lag: there is not a natural matrix operator to support such a task.

**Graph data.** A graph $G = (V, E)$ is defined over a set of vertices $V$ (consider $V$ as a set of data objects), and a set of edges $E \subseteq V \times V$. The edges $E$ represent binary relations over $V$. One typical example is to use graphs to model *social networks*: in this case, the set of vertices $V$ represents a set of persons, and the set of edges $E$ captures a certain relation among the persons in $V$, e.g., friendship. An edge $e \in E$ is denoted by $e = (u, v)$, with $u, v \in V$, and we say that an edge is present between the vertices $u$ and $v$.

Note that a graph $G = (V, E)$ is equivalent to a 0–1 data matrix $M$, where both the rows and the columns of $M$ correspond to the set of vertices $V$, and the set of edges $E$ is related to the entries of $M$ such that $(u, v) \in E$ if and only if $M_{uv} = 1$.

Observe the following facts.

- An *undirected* graph $G$ corresponds to a *symmetric* matrix $M$.
- A non-square matrix $M$, whose rows $R$ and columns $C$ correspond to different objects (and not both to $V$), corresponds to a *bipartite* graph $G = (R, C, E)$.
- A real-valued matrix $M$ (and not necessarily 0–1) can be mapped to a *weighted* graph $G = (V, E)$, where the weight $w(u, v)$ of the edge $(u, v) \in E$ can be defined to be $M_{uv}$.