

Freie Universität Berlin



Freie Universität Berlin  
Erasmus Program

10 ECTS

## Machine learning for data science

**Professor:**  
Prof. G. Montavon

**Autor:**  
Filippo Ghirardini

---

Winter Semester 2024-2025

# Contents

<b>1</b>	<b>Data science</b>	<b>3</b>
1.1	Comparison . . . . .	3
1.1.1	Hypothesis-Driven . . . . .	3
1.1.2	Data Science . . . . .	3
1.2	History . . . . .	3
1.3	Sources of data . . . . .	3
1.4	Usages . . . . .	4
1.5	Examples . . . . .	4
1.5.1	T-SNE analysis of TCGA high-dimensional omics data . . . . .	4
1.5.2	Planetary science . . . . .	4
1.5.3	Digital humanities . . . . .	4
1.6	Comparisons . . . . .	4
1.6.1	Statistics . . . . .	4
1.6.2	Decision making . . . . .	4
<b>2</b>	<b>Data</b>	<b>5</b>
2.1	Structures . . . . .	5
2.1.1	Classic . . . . .	5
2.1.2	Images, text, sound . . . . .	5
2.1.3	Network . . . . .	5
2.1.4	Relational databases . . . . .	5
2.1.5	Fusion of datasets . . . . .	5
2.1.6	Unstructured data . . . . .	5
2.1.7	Large datasets . . . . .	5
2.1.8	Streaming data . . . . .	5
2.1.9	Data subject to regulations . . . . .	6
2.2	Preprocessing . . . . .	6
2.2.1	Missing values . . . . .	6
<b>3</b>	<b>Visualization</b>	<b>7</b>
3.1	Classical . . . . .	7
3.1.1	Array plot . . . . .	7
3.1.2	Histograms . . . . .	8
3.1.3	Scatter plots . . . . .	8
3.1.4	Graph visualization . . . . .	8
3.2	Low dimensional embedding . . . . .	9
3.2.1	MDS . . . . .	10
3.2.2	T-SNE . . . . .	11
3.2.3	Comparison . . . . .	12
3.2.4	Networks . . . . .	12
3.2.5	Stringing . . . . .	13
3.2.6	Limits . . . . .	13
<b>4</b>	<b>Data dispersion</b>	<b>14</b>
4.1	Lagrange multipliers . . . . .	14
4.2	PCA . . . . .	14
4.2.1	Formulations . . . . .	15
4.2.2	Solution . . . . .	16
4.2.3	Problems . . . . .	16
4.2.4	Multiple components . . . . .	17
4.2.5	Biplot . . . . .	17
4.2.6	Explained variance . . . . .	17
4.2.7	Scree plot . . . . .	18
4.2.8	Applications . . . . .	19
4.2.9	Improvements . . . . .	19

# Machine learning for data science

Author: Ghirardini Filippo

Winter Semester 2024-2025

---

# 1 Data science

**Definition 1.0.1** (Study of data). *Data science is a study of data. It includes:*

- *Processing data*
- *Representation of data (compressed, understandable)*
- *Value extraction from data*
- *Knowledge extraction from data*

**Definition 1.0.2** (Data-Driven science). *Data science as a tool scientific discovery, complement to hypothesis-drive science.*

## 1.1 Comparison

### 1.1.1 Hypothesis-Driven

In the **hypothesis-driven** experimentation, we first form an *hypothesis*, then we perform an *experiment* and in the end we *check* the results.

### 1.1.2 Data Science

**Data-Driven** science instead starts with taking *existing data*, performing an *analysis* on them and *observe* the result.

## 1.2 History

In the past, experimentation and data collection existed with separate purposes. When the **cost** of collection and accessing the data dropped dramatically (thanks to the internet and all the sensors), they started being implemented in science, since on the other hand the cost of an experiment was the same. Furthermore, computers are now able to process large amount of data.

**Observation 1.2.1.** The first and most important thing in data science is collecting the data and only later working on it.

The first example of data science can be found in a map made by John Snow in 1854 showing the clusters of cholera cases in London. With the map he noticed that there was a center place with a fountain/well and understood that the disease was transmitted via water and not air.

## 1.3 Sources of data

There are multiple sources of data:

- Physics equations: data generated artificially via simulation running multiple times with different parameters
- User content
- User activity: for example public GPS traces of OpenStreetMap, that enables to gain insights on mobility patterns (e.g. relation between location, mode of transportation, date and user).
- Historical books or artifacts: for example *De Sphaera Corpus*, the corpus of multiple editions of the cited textbook made available in digital form. It enables similar dataset-wide analysis, relating content, year and city of publications.
- Earth data
- Biomedical data: for example the National Cancer Institute in the USA and the UK Biobank data

*Note 1.3.0.1.* Data is often not intended for scientific use. It is often a by-product of an activity with a different purpose (e.g. clinical practice, accounting).

## 1.4 Usages

The outcome of a data science analysis can:

- be **insightful** and of interest by itself (and be published)
- inform on further experiments or investigations to be conducted in order to **verify a hypothesis**
- inform on whether it is **feasible** to build a system that accurately predicts the data

## 1.5 Examples

### 1.5.1 T-SNE analysis of TCGA high-dimensional omics data

We start by defining:

- $x_i$  is the representation of  $i$  in the original measurement space (e.g.  $x_i \in \mathbf{R}^{1000}$ )
- $d_{ij}$  is the distance between instance  $i$  and  $j$  in measurement space

$$d_{ij} = \|x_i - x_j\|$$

- $y_i$  is the representation of instance  $i$  in low-dimensional space, usually  $\mathbf{R}^2$
- 

### 1.5.2 Planetary science

Can internal properties of a planet be predicted from a few observables? Yes, starting from predetermined parameters we then run tons of simulations via convection models and check the observables. In the end we try to build a model that can predicts them. This is an example of a **correlational** research.

### 1.5.3 Digital humanities

The idea was to extract correlations between visual patterns and categorization of images.

## 1.6 Comparisons

### 1.6.1 Statistics

**Statistics** makes particular assumptions about the nature of the data whereas **data science** addresses the data as it occurs in real-world applications. Data science also addresses the technical challenge of processing the data.

### 1.6.2 Decision making

**Data science** focuses on extracting insightful structures and relations from the data and presenting them in an interpretable manner, while **Decision Making** focuses on learning good autonomous decisions so that repetitive tasks can be performed without human intervention.

## 2 Data

### 2.1 Structures

#### 2.1.1 Classic

A classical dataset consists of a collection of  $N$  **instances** (data points, examples) where each instance can be represented as a vector of  $d$  **features** (attributes, measurements).

They can be stored in a two-dimensional array structure of  $N \times d$  and they usually come with **metadata** that explains the data.

#### 2.1.2 Images, text, sound

Sometimes the data is **not tabular** but, for example, is an image, text or a sound. In these cases the most common approach is to provide the dataset as a folder that contains the file. Sub folders may be used to organize the data according to metadata.

#### 2.1.3 Network

In this case the data consists of a network of  $N$  instances with connections (directed or not, weighed or not) between pairs of related instances. It can be represented as an **adjacency matrix** of size  $N \times N$ . Since it is typically sparse (one node connected to few), it may be better to use a **sparse representation** such as a table of size  $\#edges \times 2$  storing the links.

#### 2.1.4 Relational databases

It's a collection of tables of two different types:

- The first one has a row for each entity and a column for each attribute
- The second one stores relations between instances of two different tables

The analysis may proceed either by:

- Focusing on data from a single table
- Joining tables
- Operating on the relational structure using advanced techniques

#### 2.1.5 Fusion of datasets

**Aggregation** of multiple small datasets can enable the learning of more general and accurate models. For them to be valuable, data coming from the multiple sources needs to be **homogenized**. Furthermore, **implicit information** in the original datasets needs to be included in the aggregated one, ideally as additional features or metadata.

#### 2.1.6 Unstructured data

Data may have a level of heterogeneity such that there is no obvious data model that can be used. In that case, the data model must be rebuilt from scratch using expert knowledge from the field.

#### 2.1.7 Large datasets

Datasets whose size is too large to be processed with classical techniques (e.g. high throughput devices like FMRI or complex simulations). In this situation advanced approaches are needed like data parallelism and model synchronization between multiple machines.

#### 2.1.8 Streaming data

In this case data arrives continuously at a high rate. Insights need to be delivered in a timely fashion since there is no time to collect a full batch before the analysis.

### 2.1.9 Data subject to regulations

User data or medical data is subject to regulations for privacy reasons that determine who can access the data. There could be the need for a two-level data analysis: the first one may be performed only on non sensitive data.

## 2.2 Preprocessing

Tabular data can be converted into an array via

---

```
numpy.getfromtxt
```

---

while non numerical may be discarded or converted to a numerical value.

**Images** can be loaded in python via **PIL** or **cv2**. Otherwise, one can use raw pixel values to compute low level features or feed the image to a pretrained neural network feature extractor.

**Sound** data are usually converted in spectrograms showing the frequency information at coarser time steps.

**Text** data can be converted to numbers with encodings. You can also remove non important words such as "the" or "and".

### 2.2.1 Missing values

When there are missing values (e.g. faulty sensor) we can:

- Replace missing values with standard ones
- Replace with the most likely given the others
- Encode each value as a two-dimensional vector, e.g.

$$x \mapsto (x, I\{\text{missing}\}) \quad x \mapsto (x, 1 - x) \cdot (1 - I\{\text{missing}\})$$

### 3 Visualization

Visualization is a key component of data analysis since it can provide insights on its own. It relies on the fact that a human can easily recognize patterns. Usually they are 2D with color.

#### 3.1 Classical

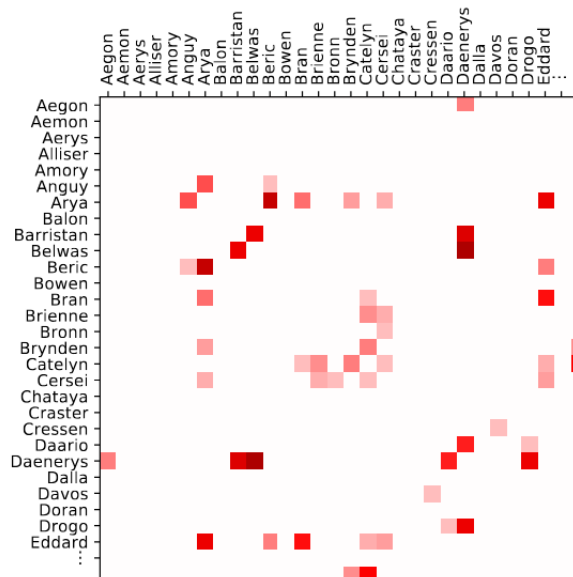
There are basically four categories of classical visualization, depending on the type of data:

- Array plot
- Scatter plot
- Histograms
- Graphs

##### 3.1.1 Array plot

Usually a two dimensional space organized as a **grid**. Each row represents an instance and the columns represent numerical features. Each element in the array is then **colored** according to the feature value of the given instance and a color map. Usually there is also a color scale.

**Example 3.1.1** (Game of Thrones). In this example we see the adjacency matrix for the relationship between all the characters in Game of Thrones.



The strengths of this type of visualization are that a lot of **qualitative information** can be gathered and it's really helpful for spotting **missing values** or **lack of normalization** before a more advanced data analysis.

At the same time, it gets quickly **overwhelming** with large datasets and it's **not very precise** about the values and their distribution.



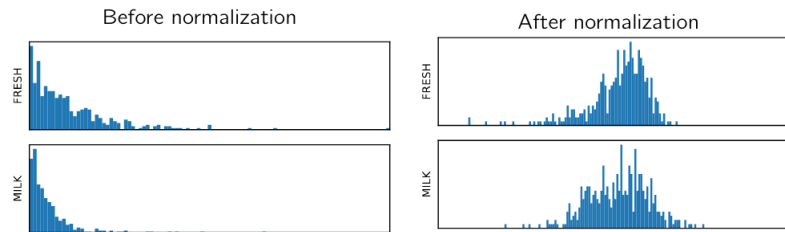
### 3.1.2 Histograms

This type of visualization focuses on just one feature so that more information can be extracted. The values are indicated by the position on the  $x$ -axis while the number of instances for each feature by the position on the  $y$ -axis.

**Example 3.1.2** (UCI Wholesale). In this example we have one Histogram for each product. Since there are different levels of spending, the result shows mainly the large spenders. To avoid this we apply the following **preprocessing**:

$$z = \log(1 + x)$$

and **standardization** (subtract the mean and divide by standard deviation).

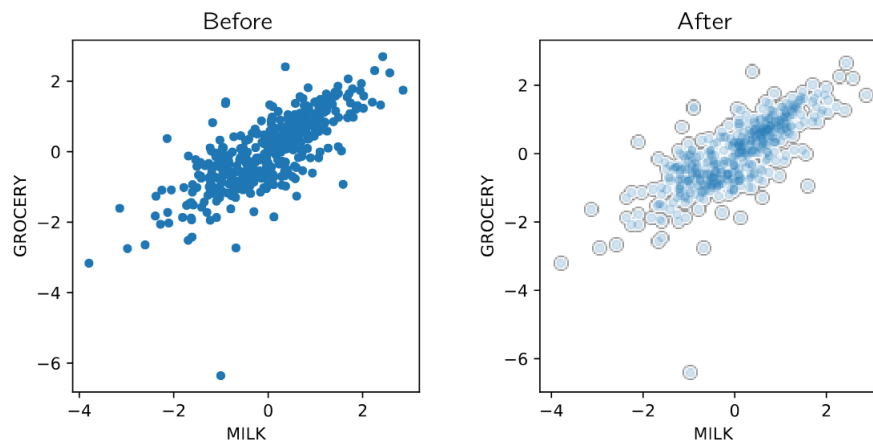


Histograms are very **precise** in characterizing the distribution of individual features but they do not highlight correlations between them. Also, not practical with a high-dimensional dataset since we would need an Histogram for every feature.

### 3.1.3 Scatter plots

Scatter plots consider two features (on the axis) at the same time to find correlations. To improve them we can use **transparency** and build a black **outline** to better find outliers.

**Example 3.1.3.** A scatter plot with the same dataset of the last example:



### 3.1.4 Graph visualization

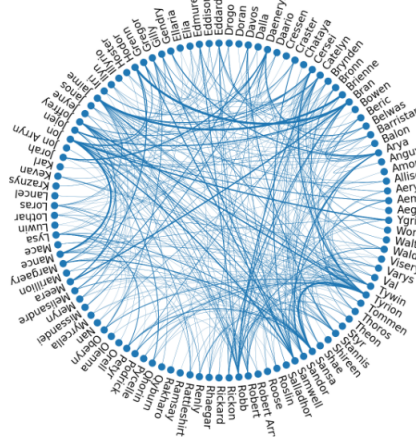
To visualize a graph, we arrange the nodes in a two-dimensional layout (e.g. a ring) and we draw lines between the nodes connected by an edge. If the edge strength is a real value, draw the line only if above a certain threshold.

It works well with a **sparse** graph.

**Example 3.1.4.** A **chord plot**, where the line used for each pair of nodes  $(x_i, x_j)$  is the following parameterized curve

$$x(t) = x_i \cdot p^t + x_j \cdot (1 - t)^p \quad (1)$$

which is a line when  $p = 1$ , otherwise it's a curve attracted to the center, making connections between nearby nodes more visible.



### 3.2 Low dimensional embedding

Low-dimensional embedding aims to construct a *scatter plot* where the axes do not carry specific meaning but it's the **distance** that represent distances and similarities in the original input space.

**Gradient descent** Assuming you want to find the minimum of a function  $J(\theta)$ , follow:

1. Initialize  $\theta$  with a random value
2. Apply repeatedly

$$\theta \leftarrow \theta - \gamma \nabla_{\theta} J(\theta) \quad (2)$$

Where  $\gamma$  is the **learning rate** set appropriately to avoid slowing down the convergence or create instability.

3. Return  $\theta$

*Note 3.2.0.1.* **Initialization** is important to reach a good minimum. Gradient descent can be made to converge quickly by following the direction of the **average** of past gradients.

**Chain rule** Suppose that a **parameter of interest**  $\theta_q$  (one element of the vector) is linked to  $z$  through a mapping

$$\theta_q \longrightarrow a \longrightarrow b \longrightarrow z$$

then the derivative of the parameter of interest is the product of the local derivatives along the path

$$\frac{\delta z}{\delta \theta_q} = \frac{\delta z}{\delta b} \cdot \frac{\delta b}{\delta a} \cdot \frac{\delta a}{\delta \theta_q}$$

In practice the parameter of interest could be linked to the output through multiple paths. The rule is therefore extended by enumerating all the paths:

$$\frac{\delta z}{\delta \theta_q} = \sum_i \sum_j \frac{\delta z}{\delta b_j} \cdot \frac{\delta b_j}{\delta a_i} \cdot \frac{\delta a_i}{\delta \theta_q}$$

### 3.2.1 MDS

Multi Dimensional Scaling (MDS) is a low-dimensional embedding technique that generates for each instance a **vector** in low dimensions. These vectors are optimized so that the distances between points in low-dimensional space replicate the true distances between corresponding instances.

It follows this notation:

- $d_{ij}$ : **true distance** between the two instances, usually the Euclidean distance in the input space

$$d_{ij} = \|x_i - x_j\|$$

- $y_i$ : representation of the **instance** in low dimensional space (usually two dimensions)

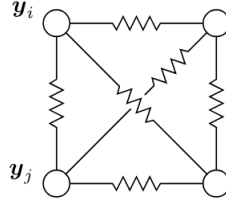
- $\hat{d}_{ij}$ : **distance** between the two instances in the low-dimensional space

$$\hat{d}_{ij} = \|y_i - y_j\|$$

**Metric MDS** It finds the embedding that minimizes the discrepancy between  $d_{ij}$  and  $\hat{d}_{ij}$  by minimizing

$$\text{stress}(y_1, \dots, y_N) = \left( \frac{\sum_{i < j} (d_{ij} - \hat{d}_{ij})^2}{\sum_{i < j} d_{ij}^2} \right)^{\frac{1}{2}} \quad (3)$$

The idea is similar to connecting the data points with strings, each one with a specific length in a relaxed state, which is determined by the input space. The Metric MDS solution is given by the relaxed state of the spring system.



The algorithm starts with a random initialization of the points  $y_1, \dots, y_N$  and iteratively updates them to minimize the *stress* function. The optimization procedure is usually done multiple times with different seeds.

Another technique is with the **gradient descent**, considering a slightly different stress function

$$\text{stress}(y) = C \cdot \left( \sum_{i < j} (\Delta_{ij} - d_{ij})^2 \right)^{\frac{1}{2}} \quad \Delta_{ij} = \|y_i - y_j\| \quad (4)$$

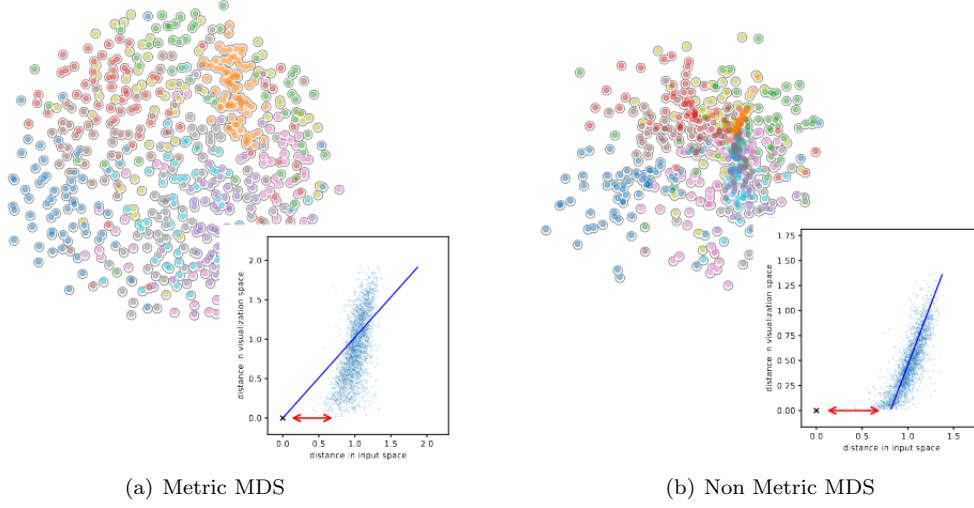
and with the gradient given by

$$\frac{\delta_{\text{stress}}}{\delta_{y_k}} = C \cdot \sum_{i < j} \frac{\Delta_{ij}}{\|\Delta\|} \cdot \frac{y_i - y_j}{\|y_i - y_j\|} \cdot (\delta_{ik} - \delta_{jk}) \quad \Delta = (\Delta_{ij})_{i < j} \quad (5)$$

**Non Metric MDS** Extends Metric MDS with the application of a fine transformation  $f$  (learned from data) to the distances. This allows for handling mismatches, enabling points that are relatively close in space to be very close in embedded space and points that are relatively far in space to be very far in embedded space.

$$\text{stress}(y_1, \dots, y_N) = \left( \frac{\sum_{i < j} (d_{ij} - f(\hat{d}_{ij}))^2}{\sum_{i < j} d_{ij}^2} \right)^{\frac{1}{2}} \quad (6)$$

**Comparison** Non Metric MDS matches more faithfully true distances between data points (up to an affine transformation) and appears crowded near the center of the visualization.



### 3.2.2 T-SNE

This method ensures to preserve **similarities**, promoting a correct representation of small distances and tolerating large errors for large distances.

It aims to learn embedding coordinates such that similarities between points matches similarities in embedded space. The collection of pairwise **similarities** are viewed as two probabilities distribution defined over the pair of points:  $p$  in the *input space* and  $q$  in the *embedded space*.

The embedding  $\mathbf{y} = (\mathbf{y}_1, \dots, \mathbf{y}_N)$  is such that minimizes the KL divergence

$$D_{KL}(p||q(\mathbf{y})) = \sum_{ij} p_{ij} \log \frac{p_{ij}}{q_{ij}(\mathbf{y})} \quad (7)$$

This function is minimized through **gradient descent**:

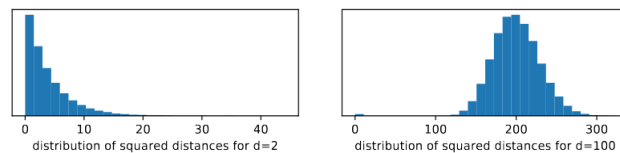
$$\mathbf{y} \leftarrow \mathbf{y} - \gamma \cdot \nabla D_{KL}(p||q(\mathbf{y}))$$

**Modeling** Similarities are modeled using different functions for the input and the embedded space because of the **concentration of distances**.

**Definition 3.2.1** (Concentration of distances). *The squared Euclidean distance between two points is given as a sum over individual dimensions:*

$$\|\mathbf{x} - \mathbf{x}'\|^2 = \sum_{t=1}^d (x_t - x'_t)^2 \quad (8)$$

*Assuming dimensions-wise squared distances are iid. (e.g. random data), the expected square distance grows linearly with  $d$  but the standard deviation grows with  $\sqrt{d}$  (cf. law of large numbers).*



Similarities  $p_{ij}$  in the **input space** are modeled with the **Gaussian** function

$$p_{ij} = \frac{1}{Z} \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right) \quad (9)$$

with  $Z$  being a normalizing term that ensures  $\sum_{ij} p_{ij} = 1$ . The scale  $\sigma$  is determined by another parameter called **perplexity**.

*Note 3.2.2.1.* In practice, to better represent low connectivity regions, one first computes conditional distributions  $p_{i|-j}$  and  $p_{j|i}$  and then sets  $p_{ij} \propto p_{i|-j} + p_{j|i}$ .

Similarities  $q_{ij}$  in the **embedded space** are modeled with the **t-Student** function

$$q_{ij}(\mathbf{y}) = \frac{1}{Z} \cdot \frac{1}{1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2} \quad (10)$$

with  $Z$  set such that  $\sum_{ij} q_{ij} = 1$ .

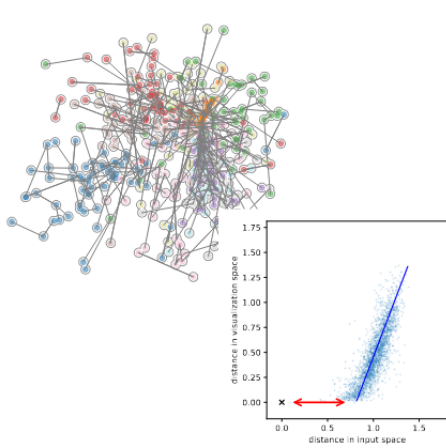
**Early exaggeration** It's the initial phase of T-SNE where the gradient of the objective

$$\nabla D_{KL}(p||q(\mathbf{y})) = 4 \cdot \sum_j (p_{ij} - q_{ij}) \cdot \frac{\mathbf{y}_i - \mathbf{y}_j}{1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2} \quad (11)$$

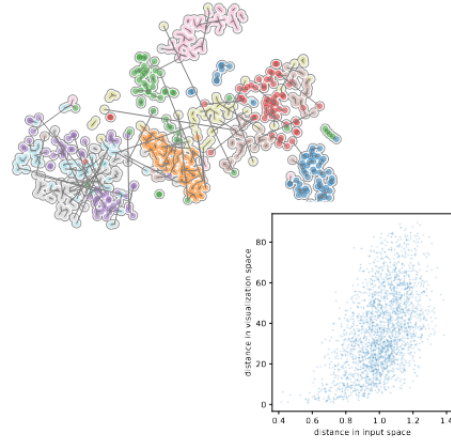
is modified by multiplying the values  $p_{ij}$  by a factor  $\alpha$  in the gradient computations. The optimization loses its probabilistic phase but it helps to escape the local optima and build compact clusters in embedded space representing the cluster structure of the input data.

### 3.2.3 Comparison

T-SNE better resolves the local structures of the data and identifies **clusters**, while MDS better preserves overall distances.



(c) MDS



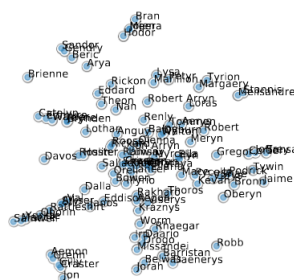
(d) T-SNE

### 3.2.4 Networks

Usually embedding techniques are used for tabular data but they might also be used for **network data** (e.g. adjacency matrix). There are different approaches:

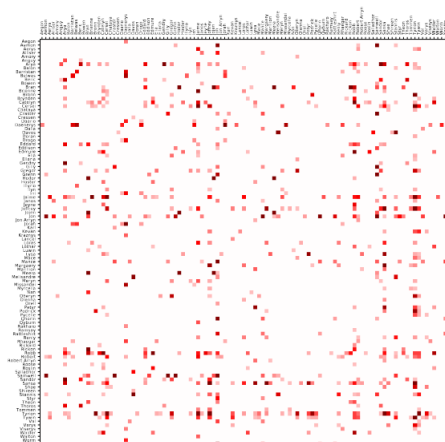
- Use an embedding techniques that natively works with adjacencies
- Consider the adjacency matrix as a data matrix  $X \leftarrow A$
- Cholesky decomposition  $A = LL^T$  and then  $X \leftarrow L$

**Example 3.2.1.** Embedding of the Game of Thrones relationships.

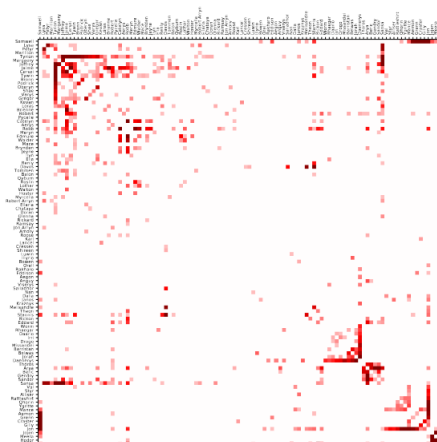


### 3.2.5 Stringing

Visualizing the data as an array can be overwhelming when the number of instances and dimensions is large. We can apply **T-SNE** to the dataset and observe that the embedding coordinates define an **ordering of instances**. We can then redraw the matrix according to it, getting an easier to interpret visualization.



(e) Before



(f) After

### 3.2.6 Limits

Low-embedding techniques have two main problems:

- They could find structures that in reality do not exist
- They can distort the geometry of the input data

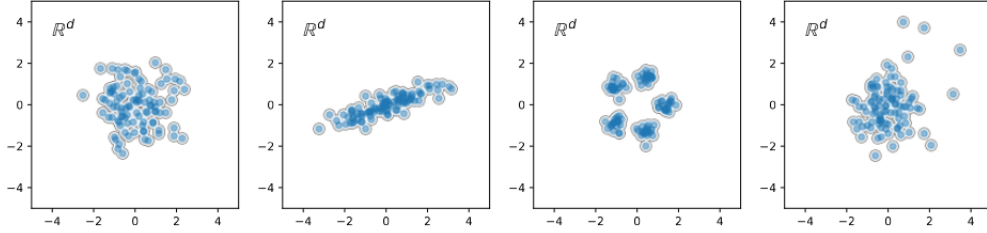
## 4 Data dispersion

Dispersion is an important property of the data that indicates how much **variation** there is in some dataset  $D = (\mathbf{x}_1, \dots, \mathbf{x}_N)$ . There are various possible measures:

- **Number of distinct data points**
- **Radius of minimal enclosing sphere**
- **Average square Euclidean distance** from the dataset mean  $\mathbf{m}$

$$s(X) = \frac{1}{N} \sum_{i=1}^N \|\mathbf{x}_i - \mathbf{m}\|^2 \quad (12)$$

Often it's important also to describe the **structure** of dispersion.



### 4.1 Lagrange multipliers

The method of Lagrange multipliers is a general framework for finding solutions of constrained optimization problems of the type

$$\arg \max_{\theta} f(\theta) \quad g(\theta) = 0$$

It consists in the following steps:

1. Construct the **Lagrangian**

$$\mathcal{L}(\theta, \lambda) = f(\theta) + \lambda \cdot g(\theta) \quad (13)$$

where  $\epsilon$  is the **Lagrangian multiplier**

2. Solve the equation

$$\nabla \mathcal{L}(\theta, \lambda) = 0 \quad (14)$$

This includes that the gradient of objective and constraint are aligned but point in opposite direction:

$$\nabla f(\theta) = -\lambda \nabla g(\theta) \quad (15)$$

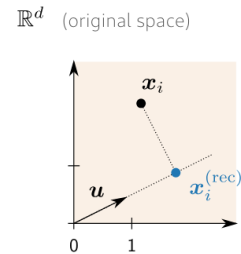
### 4.2 PCA

Principal Component Analysis is a specific type of dispersion which can be described in terms of **directions** in input space.

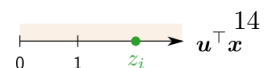
We start with the following **preliminaries**:

- Let  $\mathbf{x}_1, \dots, \mathbf{x}_N \in \mathbb{R}^d$  be a **dataset**, where  $d$  is the number of **input features** and  $N$  is the number of **data points**
- Let  $u \in \mathbb{R}^d$  be a vector of same dimensions, that represents some **directions** in the input space and is constrained to be of norm  $\|u\| = 1$
- Data points can be **projected** onto the direction via the dot product

$$\forall_{i=1}^N : z_i = \mathbf{u}^T \mathbf{x}_i \quad (16)$$



$\mathbb{R}$  (projected space)



- The projections can be **backprojected** on the input space by doing the dot product once again

$$\forall_{i=1}^N : \mathbf{x}_i^{(\text{rec})} = \mathbf{u}\mathbf{u}^T \mathbf{x}_i \quad (17)$$

#### 4.2.1 Formulations

We have two possible formulations:

**Dispersion maximization** Find a projection  $z = \mathbf{u}^T \mathbf{x}$  of the data under which the dispersion (variance) is maximized.

This means finding a direction  $\mathbf{u}$  (with  $\|\mathbf{u}\| = 1$ ) so that the data projected onto this direction  $z_1, \dots, z_N$  has maximum variance.

$$\arg \max_{\mathbf{u}} \left[ \frac{1}{N} \sum_{i=1}^N (\mathbf{u}^T \mathbf{x}_i - \tilde{m})^2 \right] \quad \mathbf{u}^T \mathbf{x}_i = z_i \quad (18)$$

where  $\tilde{m} = \frac{1}{N} \sum_{i=1}^N z_i$  is the dataset mean in the projected space. Since we center the data, we then have  $\tilde{m} = 0$  and therefore

$$\arg \max_{\mathbf{u}} \left[ \frac{1}{N} \sum_{i=1}^N (\mathbf{u}^T \mathbf{x}_i)^2 \right] \quad (19)$$

**Error minimization** Find the direction that minimizes the reconstruction error (MSE) between the original data point  $\mathbf{x}$  and its backprojection  $\mathbf{x}^{(\text{rec})} = \mathbf{u}\mathbf{u}^T \mathbf{x}$ .

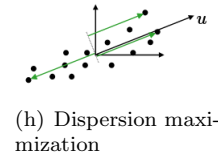
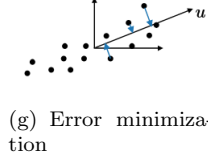
This means finding the direction  $\mathbf{u}$  (with  $\|\mathbf{u}\| = 1$ ) so that the data projected on the corresponding subspace best reconstructs the original data, having **minimal squared distance** to it.

$$\arg \min_{\mathbf{u}} \left[ \frac{1}{N} \sum_{i=1}^N \|\mathbf{x}_i - \mathbf{u}\mathbf{u}^T \mathbf{x}_i\|^2 \right] \quad \mathbf{u}\mathbf{u}^T \mathbf{x}_i = \mathbf{x}_i^{(\text{rec})} \quad (20)$$

**Observation 4.2.1.** As of Pearson 1901, the two views coincide:

$$\begin{aligned} & \arg \min_{\mathbf{u}} \left[ \frac{1}{N} \sum_{i=1}^N \|\mathbf{x}_i - \mathbf{u}\mathbf{u}^T \mathbf{x}_i\|^2 \right] \\ &= \arg \min_{\mathbf{u}} \left[ \frac{1}{N} \sum_{i=1}^N (\mathbf{x}_i - \mathbf{u}\mathbf{u}^T \mathbf{x}_i)^T (\mathbf{x}_i - \mathbf{u}\mathbf{u}^T \mathbf{x}_i) \right] \\ &= \arg \min_{\mathbf{u}} \left[ \frac{1}{N} \sum_{i=1}^N \mathbf{x}_i^T \mathbf{x}_i - 2\mathbf{x}_i^T \mathbf{u}\mathbf{u}^T \mathbf{x}_i + (\mathbf{u}\mathbf{u}^T \mathbf{x}_i)^T (\mathbf{u}\mathbf{u}^T \mathbf{x}_i) \right] \\ &= \arg \min_{\mathbf{u}} \left[ \frac{1}{N} \sum_{i=1}^N -2(\mathbf{x}_i^T \mathbf{u})^2 + \mathbf{x}_i^T \mathbf{u}\mathbf{u}^T \mathbf{u}\mathbf{u}^T \mathbf{x}_i \right] \\ &= \arg \min_{\mathbf{u}} \left[ \frac{1}{N} \sum_{i=1}^N -2(\mathbf{x}_i^T \mathbf{u})^2 + (\mathbf{x}_i^T \mathbf{u})^2 \right] \\ &= \arg \max_{\mathbf{u}} \left[ \frac{1}{N} \sum_{i=1}^N (\mathbf{u}^T \mathbf{x}_i)^2 \right] \end{aligned}$$





#### 4.2.2 Solution

To find the principal component, we start from the dispersion maximization equation 19 and rewrite it as

$$\begin{aligned}
 & \arg \max_{\mathbf{u}} \left[ \frac{1}{N} \sum_{i=1}^N (\mathbf{u}^T \mathbf{x}_i)^2 \right] \\
 &= \arg \max_{\mathbf{u}} \left[ \frac{1}{N} \sum_{i=1}^N (\mathbf{u}^T \mathbf{x}_i)(\mathbf{x}_i^T \mathbf{u}) \right] \\
 &= \arg \max_{\mathbf{u}} \left[ \mathbf{u}^T \left( \frac{1}{N} \sum_{i=1}^N \mathbf{x}_i \mathbf{x}_i^T \right) \mathbf{u} \right] \quad \|\mathbf{u}\| = 1
 \end{aligned}$$

where  $\Sigma = \frac{1}{N} \sum_{i=1}^N \mathbf{x}_i \mathbf{x}_i^T$  is the **covariance matrix** that, since it does not depend on  $\mathbf{u}$ , can be precomputed. We apply the Lagrangian multipliers method:

1. We rewrite the constraint as  $\|\mathbf{u}\|^2 = 1$  and build the Lagrangian

$$\mathcal{L}(\mathbf{u}, \lambda) = \mathbf{u}^T \Sigma \mathbf{u} + \lambda \cdot (1 - \|\mathbf{u}\|^2)$$

2. We set the gradient to zero

$$\nabla_{\mathbf{u}} \mathcal{L}(\mathbf{u}, \lambda) = 0 \Rightarrow \Sigma \mathbf{u} = \lambda \mathbf{u}$$

$$\nabla_{\lambda} \mathcal{L}(\mathbf{u}, \lambda) = 0 \Rightarrow \|\mathbf{u}\|^2 = 1$$

This means that the solution is an **eigenvector** of  $\Sigma$ , in particular

$$\begin{aligned}
 \Sigma \mathbf{u} &= \lambda \mathbf{u} \\
 \mathbf{u}^T \Sigma \mathbf{u} &= \lambda \mathbf{u}^T \mathbf{u} \\
 \mathbf{u}^T \Sigma \mathbf{u} &= \lambda
 \end{aligned}$$

meaning that for the *objective* (left part of the equation) to be maximized, we have to choose the eigenvector in a way that the corresponding eigenvalue  $\lambda$  is maximum, therefore the **leading eigenvector**.

#### 4.2.3 Problems

The major problems are:

- PCA is not very robust to **outliers**. We need to clean the data before.
- PCA does not describe well the data when it's strongly **non-Gaussian**. It fails to account for the fact that data may vary locally in different directions.

#### 4.2.4 Multiple components

The basic PCA method consists of rewriting the data as the sum of two components: what PCA is able to **reconstruct** and a **residue** of what cannot be captured.

$$\mathbf{x} = \mathbf{u}\mathbf{u}^T\mathbf{x} + (\mathbf{x} - \mathbf{u}\mathbf{u}^T\mathbf{x}) \quad (21)$$

It's possible to find secondary principal components in the residue with the following algorithm:

---

```

 $X_{\text{res}} \leftarrow X$ 
for  $j=1$  to  $h$  do
   $w_j \leftarrow \text{PCA}(X_{\text{res}})$ 
   $X_{\text{res}} \leftarrow X_{\text{res}} - \mathbf{w}_j\mathbf{w}_j^T X_{\text{res}}$ 
end for

```

---

This gives as output a collection of directions  $w_1, \dots, w_h$  which are called the **principal components**.

**Observation 4.2.2.** The principal components are equivalent to the eigenvectors  $\mathbf{u}_1, \dots, \mathbf{u}_h$  of the covariance matrix  $\Sigma$  sorted by decreasing associated eigenvalues  $\lambda_1 > \lambda_2 > \dots > \lambda_h$ .

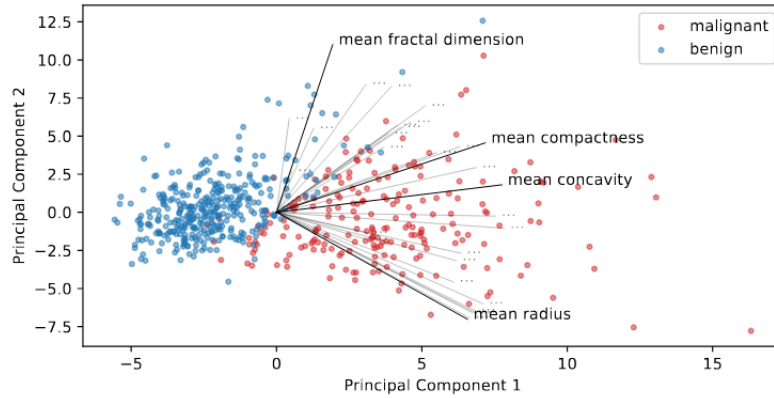
It's therefore sufficient to compute all eigenvectors and eigenvalues of  $\Sigma$  and then sort them to get the full solution of PCA.

#### 4.2.5 Biplot

The biplot is a common visualization on the two leading principal components. Each instance corresponds to a point in a scatter plot and its coordinates are given by the pair

$$(\mathbf{u}_1^T\mathbf{x}, \mathbf{u}_2^T\mathbf{x})$$

Input features can also be visualized in this plot by projecting their associate canonical coordinate vector in PCA space. They are usually depicted as arrows with the feature name and rescaled for visualization purposes.



#### 4.2.6 Explained variance

The **dispersion** of a multivariate set can be measured as the generalized variance, which can be expressed in terms of  $\Sigma$ :

$$s_{\text{tot}} = \mathbb{E}[\|\mathbf{x} - \mathbf{m}\|^2] = \sum_{j=1}^d \mathbb{E}[(x_{ij} - m_j)^2] = \text{Tr}(\Sigma) \quad (22)$$

where  $\mathbb{E}[\cdot]$  denotes an average over points in the dataset.

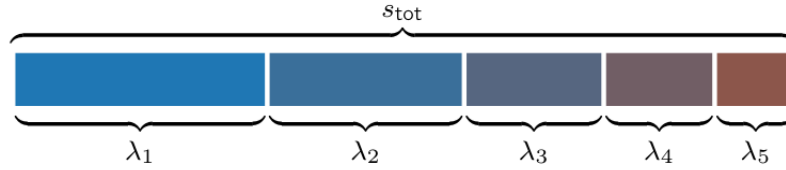
The variation of the data projected on the  $k$ th principal component can be expressed as

$$s_k = \mathbb{E}[(\mathbf{u}_k^T(\mathbf{x} - \mathbf{m}))^2] = \mathbf{u}_k^T \Sigma \mathbf{u}_k = \lambda_k \quad (23)$$

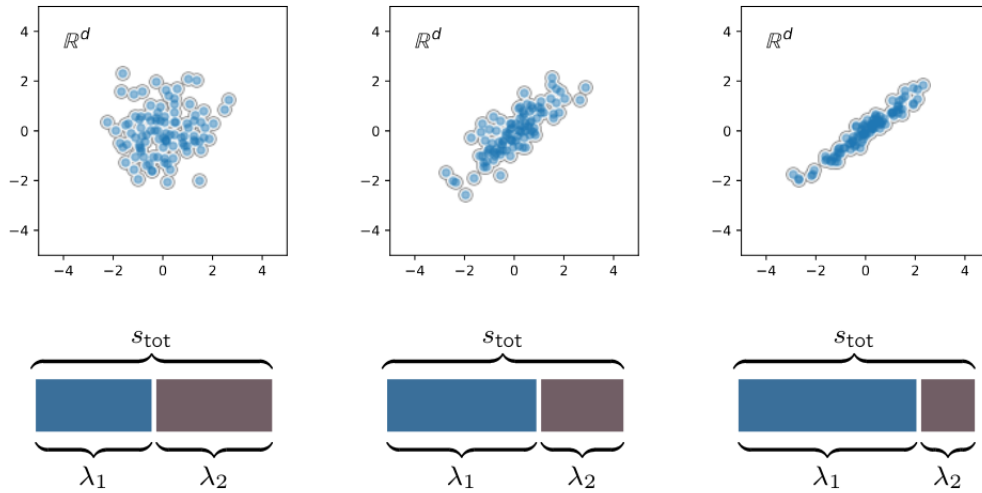
meaning that it corresponds to the  $k$ th eigenvalue of the covariance matrix. Therefore the following is true:

$$\text{Tr}(\Sigma) = \sum_{k=1}^h \lambda_k \quad (24)$$

This means that PCA produces an additive **decomposition** of the total dispersion in terms of principal components.

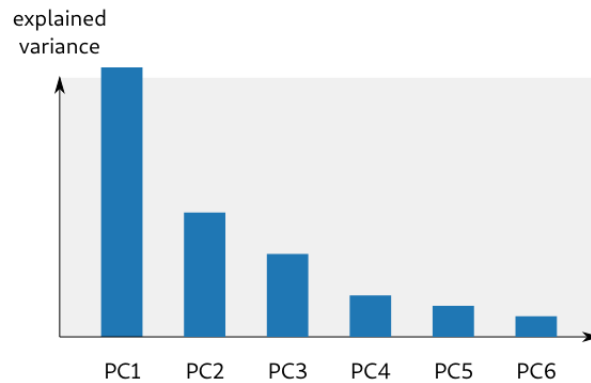


**Example 4.2.1.** We could have three two dimensional datasets that have the same overall dispersion  $s_{\text{tot}}$  but distributed differently over principal components:



#### 4.2.7 Scree plot

The scree plot is useful to get a picture of the effective **dimensionality of the data**. If only the first few bars are large, it means that the effective dimensionality is small and the data is simple.



In this plot, each bar corresponds to the *explained variance* associated to a particular principal component. Its height is given by the associated *eigenvalue*. It can be interpreted as the share of the total variance explained by this component.

*Note 4.2.7.1.* Sometimes the information is better depicted as a **cumulative plot** where the  $k$ th bar indicates the variance obtained retaining the leading  $k$  principal components.

#### 4.2.8 Applications

PCA is not used only to describe the data but also to remove certain factors that contribute to data dispersion, such as:

- **Artifact** removal: it may be reasonable to remove first principal components (e.g. eye blinking in EEG)
- **Denoising**: remove last principal components

#### 4.2.9 Improvements

Since many data are high-dimensional, the standard implementation of the PCA would have to compute a covariance matrix  $\Sigma$  of  $d \times d$  size, which would be time and space consuming.

### SVD

**Definition 4.2.1** (SVD). *A singular value decomposition factorizes a matrix  $M = U\Lambda V$  where*

- *$U$  contains the eigenvectors of  $MM^T$*
- *$V$  contains the eigenvectors of  $M^T M$*
- *$\Lambda$  is a diagonal matrix, with diagonal elements of  $\Lambda^2$  containing the eigenvalues of  $MM^T$*

Since SVD extracts the eigenvectors and eigenvalues of the matrix  $MM^T$  and PCA solutions corresponds to those of the covariance matrix  $\Sigma$ , we can compute the latter by setting  $\Sigma = MM^T$ , therefore

$$M = \frac{1}{\sqrt{N}} X \quad (25)$$

The **algorithm** is the following:

1. Let  $X$  be our data matrix of size  $d \times N$
2. Define  $M = \frac{1}{\sqrt{N}} X$
3. Feed the matrix  $M$  to SVD and get  $U$ ,  $\Lambda$  and  $V$
4. PCA eigenvectors are the columns of  $U$  while eigenvalues are the diagonal elements of  $\Lambda^2$

When computing the matrix  $U$  we have the option to not calculate all the matrices, which for  $d > N$  would be a huge amount. In that case  $U$  is  $d \times N$ .

The **complexity** of the algorithm is  $O(\min(N^2d, d^2N))$ , but when  $d \approx N$ , it becomes as bad as the one for computing the eigenvectors of  $\Sigma$  ( $O(d^3)$ ).

**Power Iteration** Since in large datasets even SVD is prohibitive and often we just need the first principal components, we can use the power iteration algorithm to get them. The following finds the **first** principal component:

---

```

u ≈ random()
repeat
  v ← Σu
  u ← v / ||v||
until convergence

```

---

It **always** converges **exponentially fast**.

To get the following ones:

---

```

Σ = 1/N X X^T
for j=1 to h do
  u_j ← POWIT(Σ)
  Σ ← Σ - u_j u_j^T Σ
end for

```

---