Data Analysis - Lecture 3

Data visualization

Dr. Jérémie Sublime

LISITE Laboratory - DaSSIP Team - ISEP
LIPN - CNRS UMR 7030

jeremie.sublime@isep.fr

Plan

Outline

Definitions

Visualization: Definition

Visualization is a branch of computer science involving the processing, analysis and graphical representation of data from diverse fields: social sciences, finance, medicine, entertainment, etc.

- One goal of visualization is to visually represent data that does not necessarily have a natural geometric interpretation.

Data visualization relies on two main fields:

- Statistics
- Computer graphics

Definitions

It is important to distinguish between the areas of image processing, computer graphics and visualization.

Image processing VS Computer graphics VS Visualization

- Image processing is the study of 2D images to extract information or to modify their characteristics.
- Computer graphics allows to create, draw and render images (2D or 3D), or videos of almost anything using a computer.
- Visualization allows the exploration of data represented in a visual form that helps our understanding of a shown phenomenon.

Why visualizing data ?

- A better visualization leads to a better understanding.
- Graphical visualization is an efficient way to communicate information clearly and efficiently.
- A lot of data are high dimensional and difficult to grasp without a lower dimensionality projection.
- Many Machine Learning techniques can benefit from data visualization:
  - Regression and time series analysis: to visualize the data and guess the model
  - Clustering: To guess the number of clusters and their shape, or to detect anomalies/outliers.
  - Classification: To assess classes' separability.

Why visualizing data ?

The number of features can be very large:

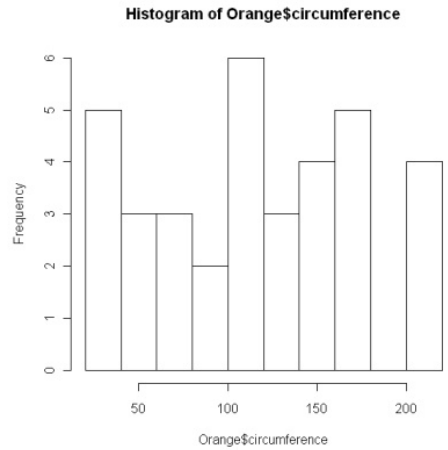
- Genomic data: thousands of variables
- Image data: a 64x64 image contains 4096 3D variables
- Text data: frequency of words or phrases in a web page
  - more than ten thousand features

When visualizing data ?

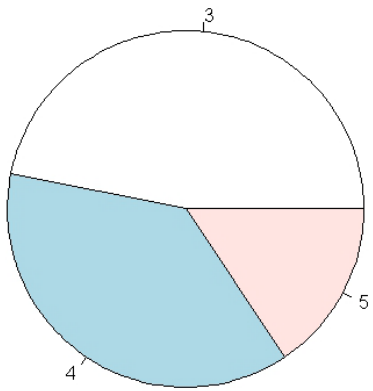
- When performing an exploratory task on new data.
- When the data are inhomogeneous or noisy.
- When the data are high dimensional and unseasy to grasp.
- To visualize and assess the result of a Machine Learning task in low dimension and see if it visually makes sense.
- To see if the data need to be transformed (rotation, dimension reduction, compression, etc.) before using a Machine Learning or data analysis task.

Univariate data

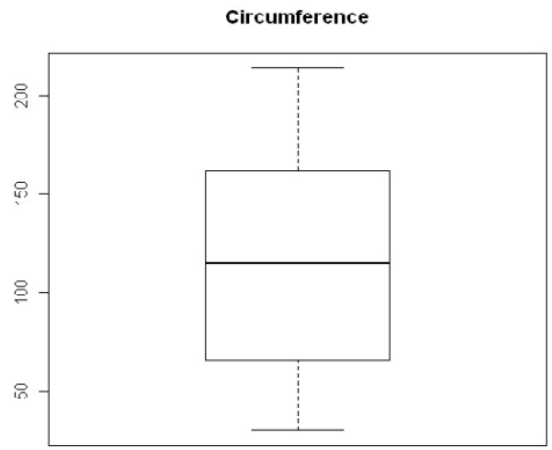
- Represents the distribution of a variable
- Represents categorical variable distribution
- Represents trends and error margins



(a) Histogram



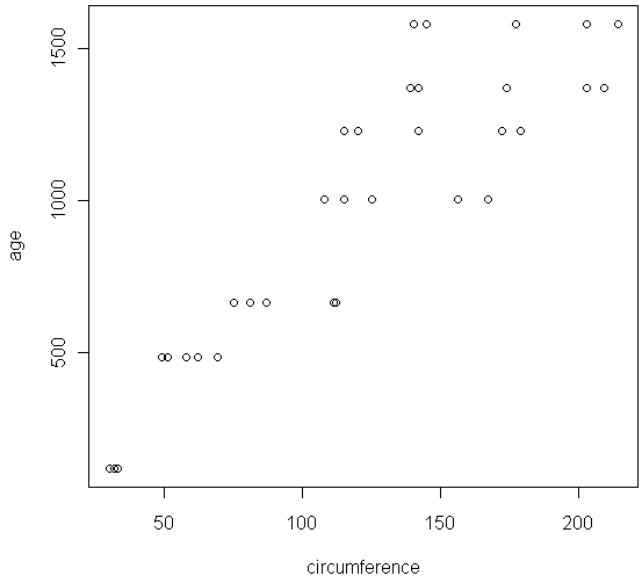
(b) Pie Plot (camembert)



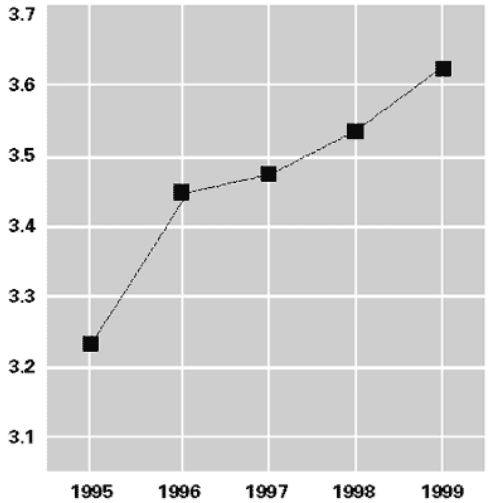
(c) Box plot (boîte à moustaches)

Bivariate data (1/2)

- Representation, correlation analysis, trend analysis



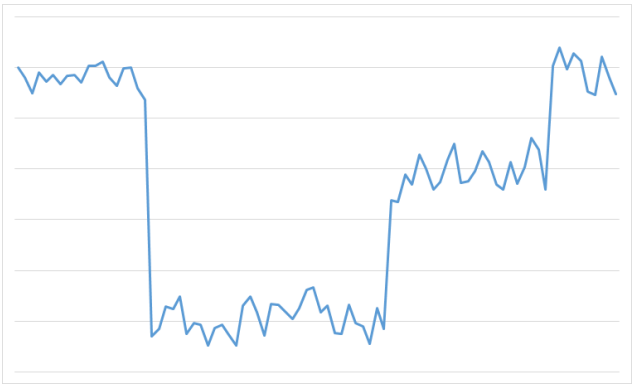
(d) Representation



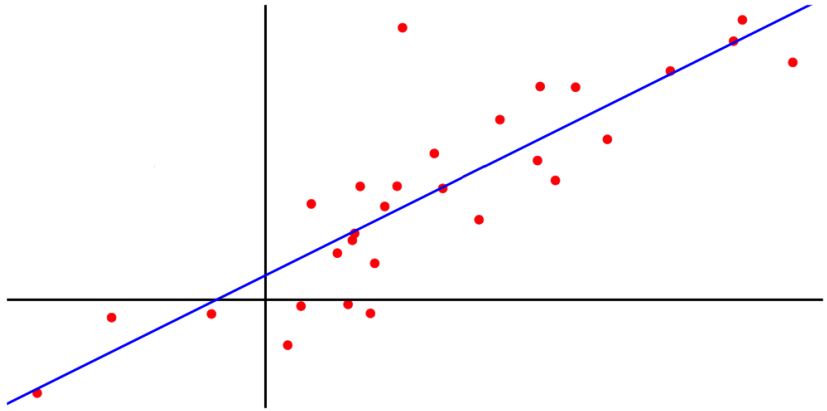
(e) Time series trends

Bivariate data (2/2)

- Representation, correlation analysis, trend analysis

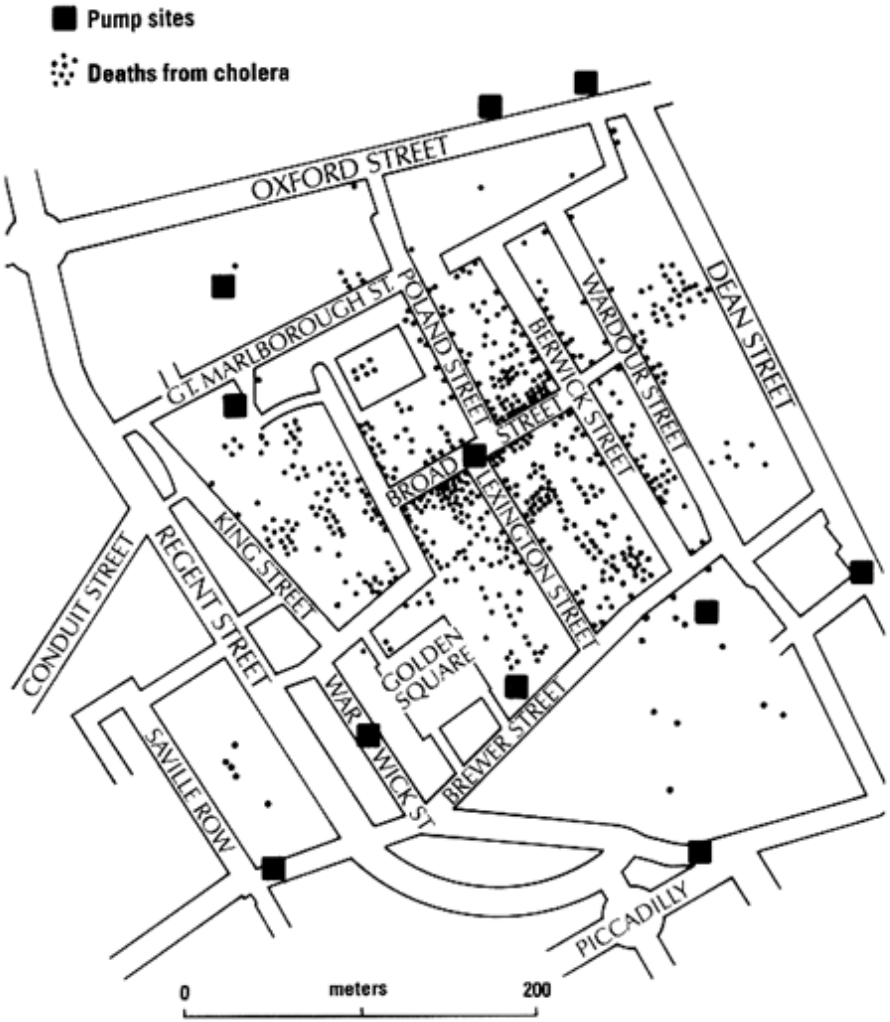


(f) Time series



(g) Correlation & Regression

Cross-bivariate data



Multivariate data

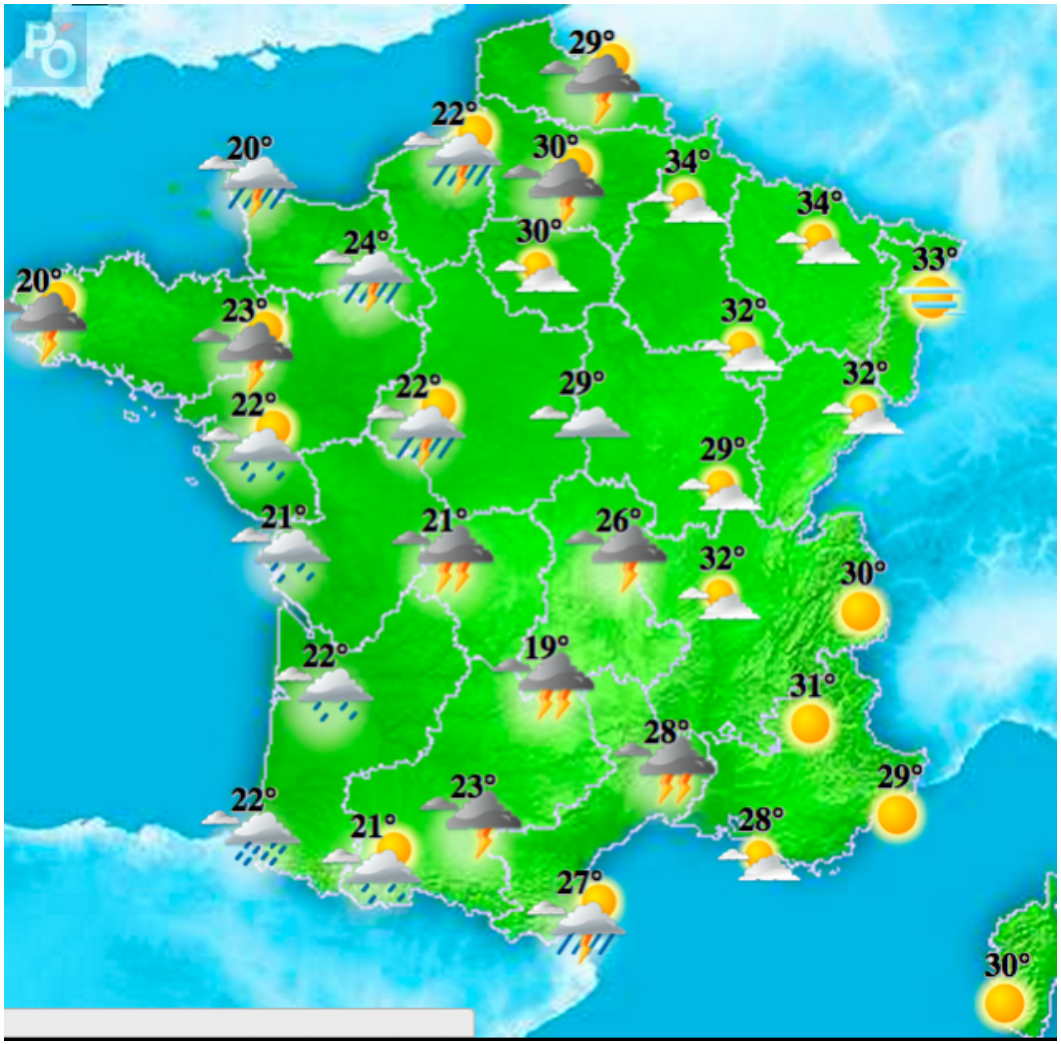
Representing multi-dimensional data in 2D (or 3D) is more complex but can help understand them.

Multi-variate data visualization techniques

- Dimension reduction and/or projection on 2 variables
- Icon-based, text-based, or color-based representations for the missing dimensions.

Multivariate data

Weather forecast: City coordinates (2 variables), weather, temperature.



The curse of dimensionality

The "*curse of dimensionality*" is a term introduced in 1961 by Bellman refering to the problem of the explosive increase in data volume associated with adding extra dimensions in a mathematical space.

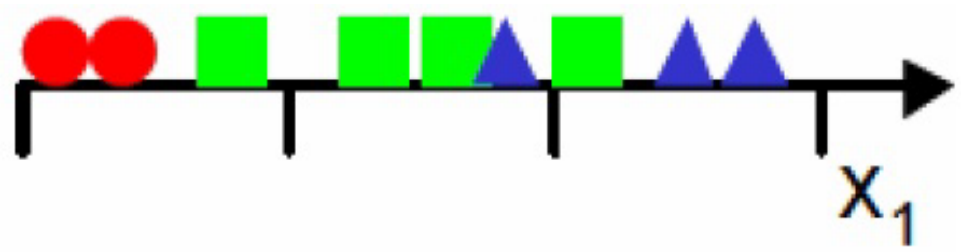
Why the curse of dimensionality matters

- High dimensional data are difficult to efficiently visualize in 2D without loosing meaningful information.
- While computer algorithms can grasp information in higher dimensional spaces than humans, past some point, they too have trouble extracting the meaningful information.

- We will illustrate this problem with a simple example.

The Toy problem

We consider a Pattern Recognition problem with 3 classes. We have 9 available observation represented in 1 dimension.



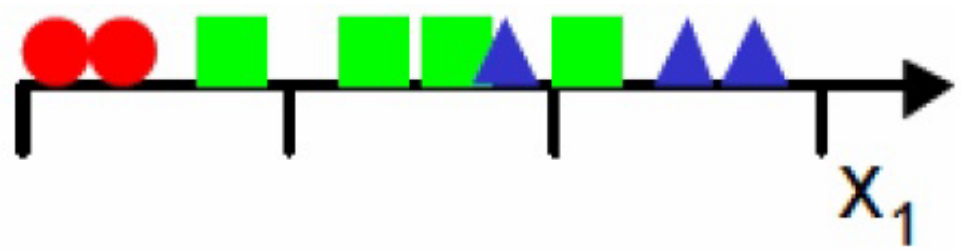
The Toy problem

We consider a Pattern Recognition problem with 3 classes. We have 9 available observation represented in 1 dimension.

A simple approach would be to do the following:

- Divide the feature space into uniform bins.
- Compute the class ratio in each bin and use a majority vote to classify the bin.
- When a new example comes, put it in its closest bin.

The Toy problem



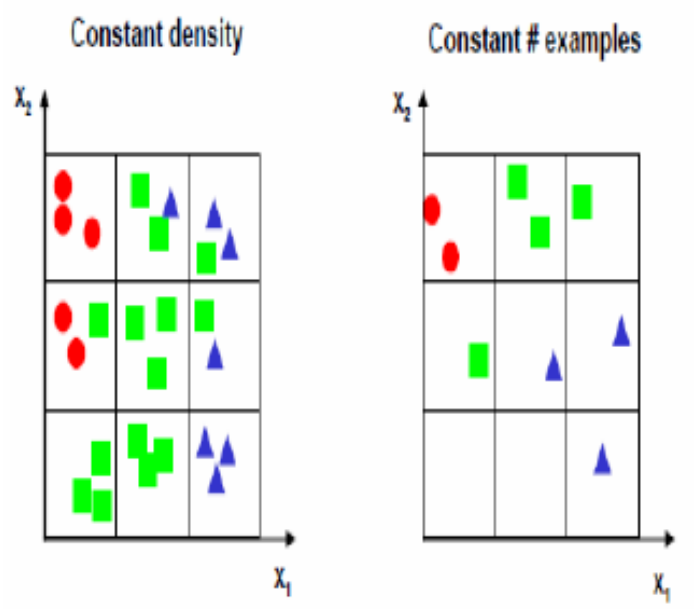
- In this first example with one feature, we observe that when dividing the space into 3 segments we get 3 homogeneous regions each with a similar density of 3 examples per bin.
- We also see that there is too much overlap between the classes, so we decide to add a second feature to try to improve the class separability.

The Toy problem 2D

- If we add a 2nd dimension we pass from 3 cases in 1D to $3^2 = 9$ in 2D.
- We have an another problem: do we maintain the density of examples per bin or do we keep the same number of example than in 1D?

The Toy problem 2D

- If we add a 2nd dimension we pass from 3 cases in 1D to $3^2 = 9$ in 2D.
- We have an another problem: do we maintain the density of examples per bin or do we keep the same number of example than in 1D?

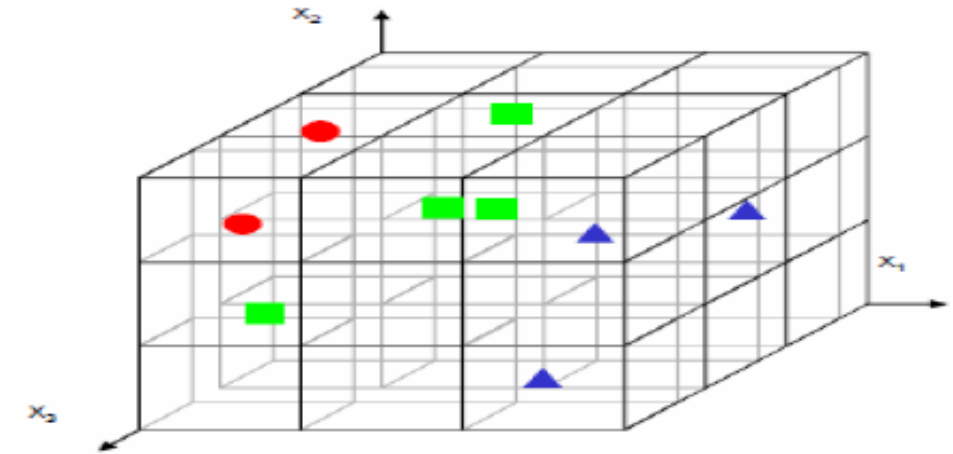


- If we want to maintain the density, we now need 27 examples instead of 9.
- Choosing to maintain the number of examples to 9 will result in a very sparse 2D scatter plot.

The Toy problem 3D

Adding a 3rd dimension makes the problem worse:

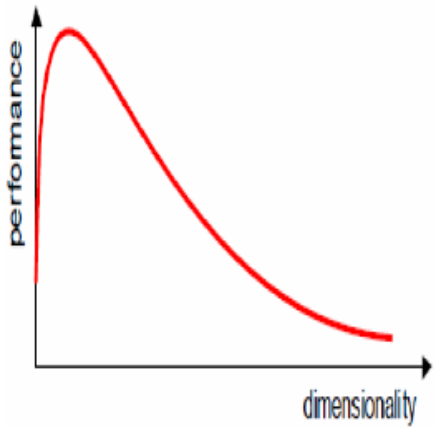
- We move to $3^3 = 27$ bins.
- Keeping a constant density now requires 81 observations.
- Keeping the same number of observation gives us an almost empty 3D plot.



The curse of dimensionality

The curse of dimensionality generates several phenomenons such as:

- The concentration of observations in given space areas
- The desertification of the data space
- The depopulation of the center of hyper-volumes



In practice, the curse of dimensionality means that, for a given sample size, there is a maximum number of variables beyond which a classifier performance will degrade rather than improve.

The approach used on the toy example is mostly ineffective:

- There are other approaches less affected by the curse of dimensionality, but the problem will still exist and will always re-surface at some point.

Consequences of the curse of dimensionality

- Exponential growth of the number of examples required to maintain a given sample density: For a density of N examples/bin in D dimensions, N^D examples are required.
- Exponential growth of the complexity of the target function (which estimates the density). To learn well, the target function requires a dense enough learning space.
- For one dimension, we have plenty of different density functions. However, for high dimension, we only have the Gaussian multivariate density. For large values of D, the density can only be treated in Gaussian simplified forms (due to the inverse covariance matrix in the Gaussian mixture model).

How to deal with the curse of dimensionality ?

These findings suggest that we need special treatment to manipulate high dimensional data:

- Incorporating prior knowledge to reduce the search space.
- Providing smoother and more generic target functions.
- Reducing dimensionality.

How to deal with the curse of dimensionality ?

These findings suggest that we need special treatment to manipulate high dimensional data:

- Incorporating prior knowledge to reduce the search space.
- Providing smoother and more generic target functions.
- Reducing dimensionality.

Reducing dimensionality

The reduction of dimensionality by the mean of space transformations, projections, or feature selection is usually the prefered solution in combination with the two others.

- Prior knowledge is not always available, and may alone result in over-fitting.
- Smoother target functions alone may result in a global decrease of performances and are cumbersome to design.

Outline

Dimension Reduction

Dimension reduction is based on the hypothesis that high dimensional data are not uniformly distributed:

- There are high density and low density areas.
- Dimension reduction looks for structures that define these high and low density areas to tackle the curse of dimensionality.

Dimension Reduction

Dimension reduction is based on the hypothesis that high dimensional data are not uniformly distributed:

- There are high density and low density areas.
- Dimension reduction looks for structures that define these high and low density areas to tackle the curse of dimensionality.

There are many techniques of dimension reduction:

- Linear and non-linear
- Deterministic and probabilistic
- Supervised and unsupervised

Selection vs Extraction

There are two main methodology for dimension reduction:

- **Feature selection:** Choosing a subset of all the features.
- **Feature extraction:** Creating a small set of new features by combinations of the original ones.

$$\begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_D \end{bmatrix} \xrightarrow{\text{feature selection}} \begin{bmatrix} X_{i_1} \\ X_{i_2} \\ \\ X_{i_M} \end{bmatrix}$$

$$\begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_D \end{bmatrix} \xrightarrow{\text{feature extraction}} \begin{bmatrix} Y_1 \\ Y_2 \\ \\ Y_M \end{bmatrix} = f\left(\begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_D \end{bmatrix}\right)$$

Feature selection

Definition

Variable selection is a process to choose an optimal subset of relevant variables from a set of variables, according to a performance criterion.

We can ask three basic questions:

- Q1 How to measure the relevance of the variables ?
- Q2 How to obtain the optimal subset ?
- Q3 Which optimality criterion to use ?

Feature selection

Q1 How to measure the relevance of the variables ?

- We need to find a measure of relevance, or evaluation criterion $J(X)$, to quantify the importance of one variable or a combination of variables.

Feature selection

Q1 How to measure the relevance of the variables ?

- We need to find a measure of relevance, or evaluation criterion $J(X)$, to quantify the importance of one variable or a combination of variables.

Q2 How to obtain the optimal subset ?

- To answer this question, we need to define a procedure, or algorithm, that will determine the optimal subset of relevant variables based on the evaluation criterion.

Feature selection

Q1 How to measure the relevance of the variables ?

- We need to find a measure of relevance, or evaluation criterion $J(X)$, to quantify the importance of one variable or a combination of variables.

Q2 How to obtain the optimal subset ?

- To answer this question, we need to define a procedure, or algorithm, that will determine the optimal subset of relevant variables based on the evaluation criterion.

Q3 Which optimality criterion to use ?

- With the answer to Q2 being an optimization algorithm, we need to define a stopping criterion for this optimization process.

Choosing a criterion for feature selection: Examples

- For a **Classification problem**, we can test the discriminant quality of the system in the presence presence or absence of a variable.
- For a **Regression problem**, we can test the quality of the prediction with respect to the other variables.

Feature selection: Branch & Bound based methods

Branch & Bound based methods use an optimization strategy that finds the optimal subset of variable based on a minimal bound to reach on the evaluation criterion.

Feature selection: Branch & Bound based methods

Branch & Bound based methods use an optimization strategy that finds the optimal subset of variable based on a minimal bound to reach on the evaluation criterion.

1. The algorithm starts with all features and removes one at each step.

Feature selection: Branch & Bound based methods

Branch & Bound based methods use an optimization strategy that finds the optimal subset of variable based on a minimal bound to reach on the evaluation criterion.

1. The algorithm starts with all features and removes one at each step.
2. If the evaluation criterion scores bellow the bound, the algorithm backtracks and tries to remove another feature.

Feature selection: Branch & Bound based methods

Branch & Bound based methods use an optimization strategy that finds the optimal subset of variable based on a minimal bound to reach on the evaluation criterion.

1. The algorithm starts with all features and removes one at each step.
2. If the evaluation criterion scores bellow the bound, the algorithm backtracks and tries to remove another feature.
3. Groups of features bellow the bound are remembered so that sub-sets of these groups are never tested later in the process.

Feature selection: Branch & Bound based methods

Branch & Bound based methods use an optimization strategy that finds the optimal subset of variable based on a minimal bound to reach on the evaluation criterion.

1. The algorithm starts with all features and removes one at each step.
2. If the evaluation criterion scores bellow the bound, the algorithm backtracks and tries to remove another feature.
3. Groups of features bellow the bound are remembered so that sub-sets of these groups are never tested later in the process.
4. When there is no feature left to try removing, the algorithm remembers the best score and backtracks to another branch.

Feature selection: Branch & Bound based methods

Branch & Bound based methods use an optimization strategy that finds the optimal subset of variable based on a minimal bound to reach on the evaluation criterion.

1. The algorithm starts with all features and removes one at each step.
2. If the evaluation criterion scores bellow the bound, the algorithm backtracks and tries to remove another feature.
3. Groups of features bellow the bound are remembered so that sub-sets of these groups are never tested later in the process.
4. When there is no feature left to try removing, the algorithm remembers the best score and backtracks to another branch.
5. The algorithm stops when all branches with valid subsets have been tested.

Monotonic VS non-monotonic feature selection criteria

Branch & Bound methods can only work if the selection criterion $J(X)$ is monotonic.

Monotonic criterion: Definition

Let us consider two subsets of features A_1 and A_2. Then, $J(\cdot)$ is monotonic if and only if $A_1 \subset A_2 \implies J(A_1) \leq J(A_2)$.

Monotonic VS non-monotonic feature selection criteria

Branch & Bound methods can only work if the selection criterion $J(X)$ is monotonic.

Monotonic criterion: Definition

Let us consider two subsets of features A_1 and A_2. Then, $J(\cdot)$ is monotonic if and only if $A_1 \subset A_2 \implies J(A_1) \leq J(A_2)$.

Problem: Most effective evaluation criteria are not monotonic ...

Monotonic VS non-monotonic feature selection criteria

Branch & Bound methods can only work if the selection criterion $J(X)$ is monotonic.

Monotonic criterion: Definition

Let us consider two subsets of features A_1 and A_2. Then, $J(\cdot)$ is monotonic if and only if $A_1 \subset A_2 \implies J(A_1) \leq J(A_2)$.

Problem: Most effective evaluation criteria are not monotonic ...

The use of sub-optimal methods is then required:

- Sequential Forward Selection (SFS)
- Sequential Backward Selection (SBS)
- Bidirectional Selection (BS)

Sequential Forward Selection

Let $X = \{X_1, \cdots, X_D\}$ be a set of variables. The SFS procedure is the following:

- Initially set the selected set of variables A_0 as empty.
- At each step k, select the variable X_i that maximizes the following criterion of evaluation $J(A_k)$:

$$J(A_k) = \max_{X_i \in (X \setminus A_{k-1})} J(A_{k-1} \cup X_i)$$

This will result in a list of variables ordered by their importance. The procedure can be stopped at any step when a given criterion (e.g. acceptable loss of information) has reached an acceptable value.

Sequential Backward Selection

Let $X = \{X_1, \cdots, X_D\}$ be a set of variables. The SBS procedure is the following:

- Initially start with a set $A_D = X$ containing all the variables.
- At each step for $k \in [D-1 \cdots 1]$, remove the variable X_i of the least importance according to:

$$J(A_k) = \max_{X_i \in A_{k+1}} J(A_{k+1} \backslash X_i)$$

This will also result in a list of variables ordered by their importance, with the most important being at the end of the list. The procedure can also be stopped has soon as the remaining set is bellow an acceptable lower bound of accuracy on a given criterion.

Bidirectional Selection

The Bidirectional Selection procedure performs the search in both Forward and Backward direction in a competitive manner:

The procedure stops when one of the following case occurs:

- One of the two direction has found the best subset of variables before reaching the middle of the search space.
- The two branches are reaching the middle.

This method reduces the search time as the search is performed in both directions and stops when there is a solution regardless of the direction.

Remark

The sets of best selected variables found respectively by SFS and SBS are not equal and rarely identical because of their different principles of selection.

Outline

Feature Extraction: Principle

Feature extraction consists in building new features from the original ones with one or several of the following goals:

- Having a lower number of features while keeping a maximum of information
- Having better features with which the data are easier to process
- Having feature with which the data are easier to visualize

$$\begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_D \end{bmatrix} \xrightarrow{\text{feature extraction}} \begin{bmatrix} Y_1 \\ Y_2 \\ \\ Y_M \end{bmatrix} = f\left(\begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_D \end{bmatrix}\right)$$

Types of feature extraction methods

Linear Methods

- Principal components Analysis (PCA)
- Linear Discriminant Analysis (LDA)
- Multi-Dimensional Scaling (MDS)
- ...

Non-linear methods

- Isometric feature mapping (Isomap)
- Locally Linear Embedding (LLE)
- Kernel PCA
- Spectral clustering
- Supervised methods (S-Isomap)
- ...

Reducing the dimensionality: goals

The Principal Component Analysis *(PCA) is a method of data analysis and dimensionality reduction that seeks the directions of space that best represent the correlations between D random variables.*

Goals

1. Obtaining an approximation of a scatter of K objects in a subspace of low dimension (2 or 3).
2. Summarizing a data set represented by a matrix X of N rows and D columns.
3. Getting a simple and reliable visualization of the information contained in the data.

Reducing the dimensionality: Principle of PCA

The Principal Component Analysis *(PCA) is a method of data analysis and dimensionality reduction that seeks the directions of space that best represent the correlations between D random variables.*

Reducing the dimensionality: Principle of PCA

The Principal Component Analysis *(PCA) is a method of data analysis and dimensionality reduction that seeks the directions of space that best represent the correlations between D random variables.*

Principle

1. Compute the covariances or correlations between variables.
2. Use this indexes to create new variables:
   - The original variables are replaced with the new variables.
   - Each new variable is a linear combination of the original variables.
   - These new variables must be as independent (uncorrelated) from each other as possible.
3. Select among the new variables the ones that best represent the distribution of the data and eliminate the others.

PCA : Principle

Breiman Waveform

-1.23,-1.56,-1.75,-0.28,0.60,2.22,0.85,0.21,-0.20,0.89,1.08,4.20,2.89,7.75,4.59,3.15,5.12,3.32,1.20,0.24,-0.56,2
-0.69,2.43,0.61,2.08,2.30,3.25,5.52,4.55,2.97,2.22,2.81,1.61,1.24,1.89,1.88,-1.34,0.83,1.41,1.78,0.60,2.42,1
-0.12,-0.94,1.29,2.59,2.42,3.55,4.94,3.25,1.90,2.07,0.51,1.45,2.50,0.12,1.41,2.78,0.64,0.62,-0.01,-0.79,-0.12,0
0.86,0.29,2.19,-0.02,1.13,2.51,2.37,5.45,5.45,4.84,4.65,4.05,2.58,1.40,1.24,1.41,1.07,-1.43,2.84,-1.18,1.12,1
1.16,0.37,0.40,-0.59,2.66,1.00,2.69,4.06,5.34,3.53,4.82,4.79,4.30,1.84,1.73,0.21,-0.18,0.13,-0.21,-0.80,-0.68,1
-0.00,0.77,1.32,0.29,-1.28,0.84,1.60,1.55,2.93,4.76,5.55,4.30,4.89,2.81,2.37,3.68,-0.98,0.69,0.91,-1.80,0.39,2
0.87,1.07,-0.65,1.46,0.84,2.70,3.67,2.94,3.81,5.20,8.16,3.29,4.24,2.43,0.40,1.60,0.72,0.66,0.05,-0.24,0.67,1
-0.22,-0.91,-1.18,0.35,-1.92,-1.59,1.91,0.75,1.72,2.02,3.63,3.91,2.73,4.29,4.89,2.04,1.13,-0.66,-1.33,0.41,-0.75,2
-1.11,-1.14,-0.89,0.00,0.53,0.44,0.24,2.15,1.64,1.75,3.92,5.68,3.39,4.24,3.81,4.56,3.18,1.51,2.90,0.14,-0.12,2
-0.75,1.10,-1.90,1.43,0.47,0.40,0.86,3.51,2.62,4.50,6.83,6.94,0.75,3.23,1.08,-0.25,0.73,-0.41,-1.50,0.46,1.47,2
0.14,-1.18,1.42,2.28,3.10,3.15,3.49,4.54,1.40,3.41,3.40,2.83,0.06,0.60,3.61,2.08,-0.83,0.55,-0.55,-0.43,-1.05,0

....

1.32,-0.40,-0.69,4.17,3.66,4.00,5.24,3.88,2.17,1.82,3.65,1.01,1.82,1.13,-0.07,0.26,0.50,1.38,1.25,-1.34,0.53,0
-0.93,2.48,1.20,2.97,2.91,3.57,3.68,4.19,3.22,3.53,2.46,2.17,0.77,0.52,2.42,-0.89,0.51,-0.39,0.82,0.14,-0.63,1
-1.06,0.59,1.01,3.33,2.05,3.20,4.70,4.21,4.73,2.22,2.67,2.79,2.05,-1.53,-1.54,0.37,-0.09,1.04,-0.08,-0.27,0.47,1
1.86,0.37,-0.35,0.74,0.84,0.21,1.97,1.52,1.85,2.39,3.92,3.76,3.27,1.61,3.08,2.78,1.58,1.68,2.61,-0.91,-0.27,2
-0.51,-0.48,0.35,-1.67,0.26,2.45,-0.09,2.03,0.79,1.42,1.13,2.52,2.06,4.50,4.28,4.66,3.30,0.38,0.75,1.76,0.37,0
1.16,-1.19,-2.26,0.63,0.32,1.51,2.11,2.58,1.03,2.01,4.04,4.55,5.65,2.74,3.12,2.67,2.01,4.12,-0.81,0.07,-0.96,2
-0.09,2.30,-0.43,0.36,0.11,-1.20,1.47,2.25,3.50,2.14,6.68,5.45,2.22,2.79,2.61,1.87,0.48,1.98,1.64,1.32,0.71,2
-1.43,-0.46,-0.52,1.45,-0.00,1.35,1.39,0.81,0.03,1.39,2.55,2.42,3.07,5.46,6.29,6.50,4.60,3.77,1.62,1.01,-1.86,0

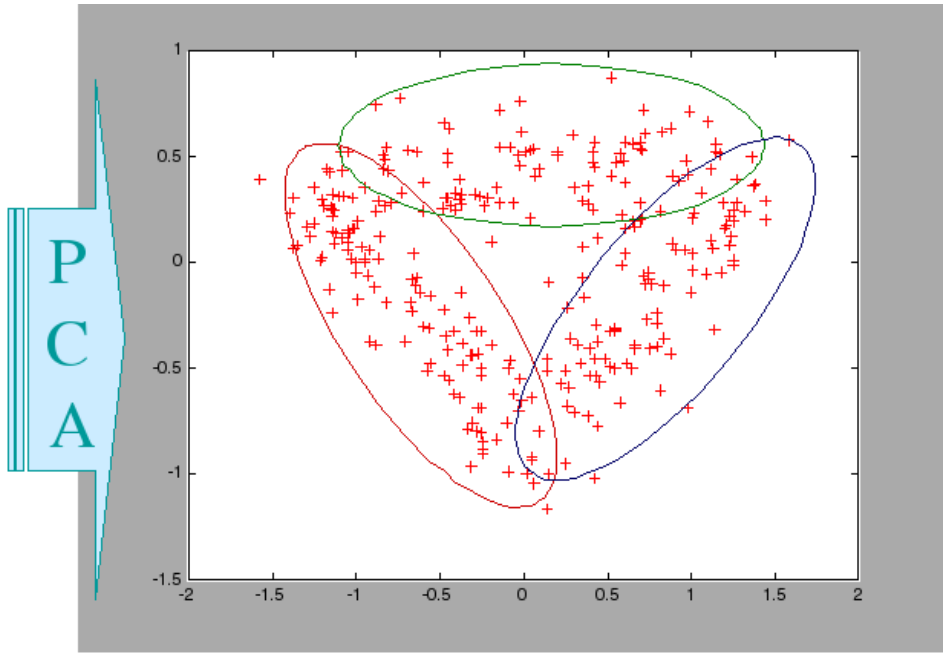


Figure: From 21 variables to 2 variables using PCA. Note that the 3 clusters are still visible.

PCA and Normalizing the data: Centering and reducing

Prior to a Principal Component Analysis, centering the variables around their means and scaling them may be necessary.

Centered variables

$$M = \begin{pmatrix} x_{1,1} - \mu_1 & \cdots & x_{1,D} - \mu_D \\ \vdots & \ddots & \vdots \\ x_{N,1} - \mu_1 & \cdots & x_{N,D} - \mu_D \end{pmatrix}$$

Centered Reduced variables

$$\tilde{M} = \begin{pmatrix} \frac{x_{1,1}-\mu_1}{\sigma_1} & \cdots & \frac{x_{1,D}-\mu_D}{\sigma_D} \\ \vdots & \ddots & \vdots \\ \frac{x_{N,1}-\mu_1)}{\sigma_1} & \cdots & \frac{x_{N,D}-\mu_D}{\sigma_D} \end{pmatrix}$$

PCA and the Variance-Covariance Matrix

Once the data have been normalized, computing the variance-covariance matrix is the next step in order to assess the potential relationships between the different variables.

Variance-Covariance

$$C = \begin{pmatrix} VAR(X_1) & \cdots & COV(X_1, X_D) \\ \vdots & \ddots & \vdots \\ COV(X_D, X_1) & \cdots & VAR(X_D) \end{pmatrix}$$

Calculation of new PCA variables

Now that we know the covariance relationships between all variables, we seek to create new variables using linear combinations of the old ones:

- A first new variable represented by a single axis u_1 so that the projection of u_1 has a maximum variance.
- Then we seek for a second axis u_2 independent of u_1 (i.e. orthogonal to u_1) that best explains the remaining variance.
- And so on.

Calculation of new PCA variables

Now that we know the covariance relationships between all variables, we seek to create new variables using linear combinations of the old ones:

- A first new variable represented by a single axis u_1 so that the projection of u_1 has a maximum variance.
- Then we seek for a second axis u_2 independent of u_1 (i.e. orthogonal to u_1) that best explains the remaining variance.
- And so on.

Note that for visualization purposes, 2 or max 3 axes are usually enough.

PCA : Example 1

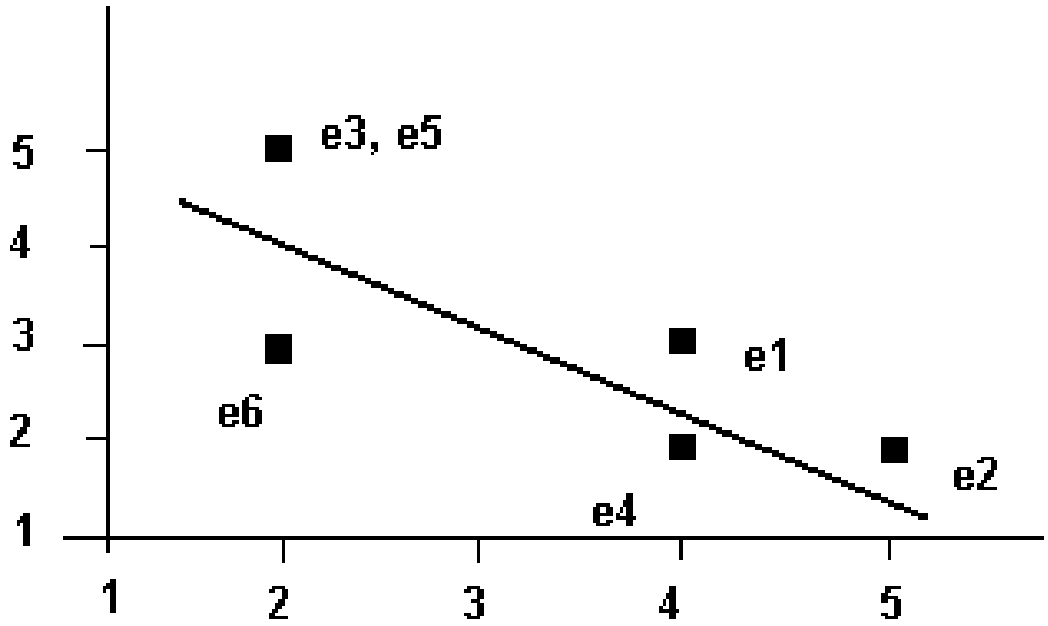


Figure: A possible projection on a new single axis

PCA : Example 2

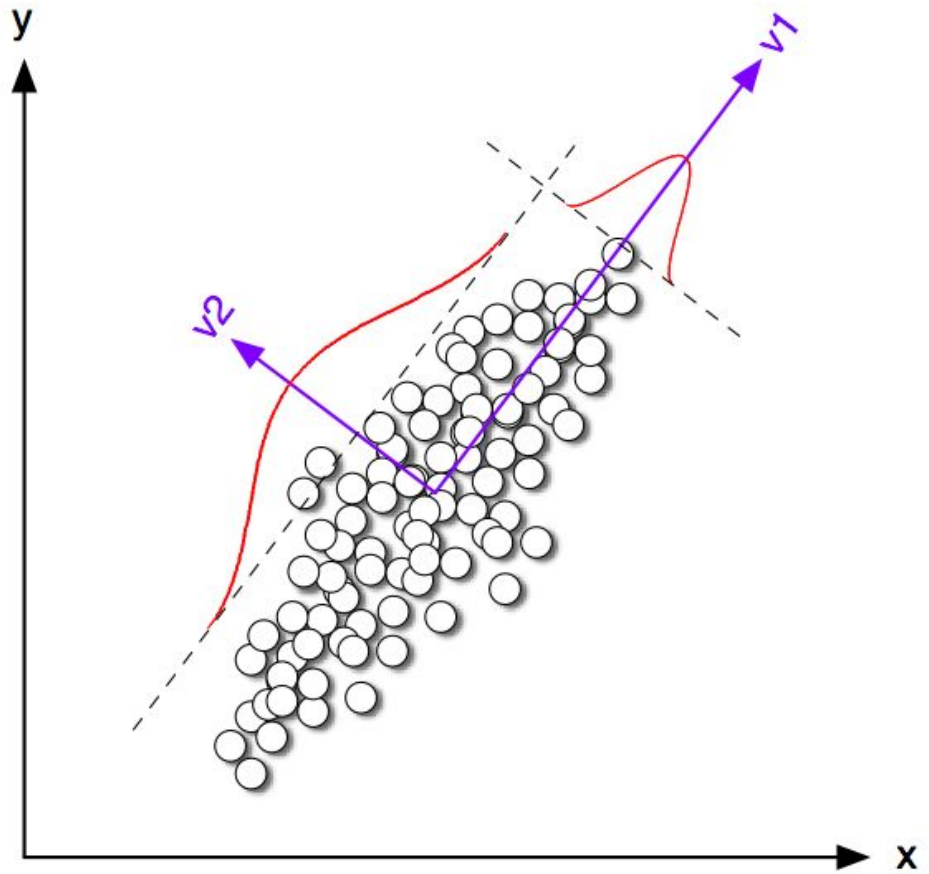


Figure: Changing the axis for more convenient variables

PCA : Example 3

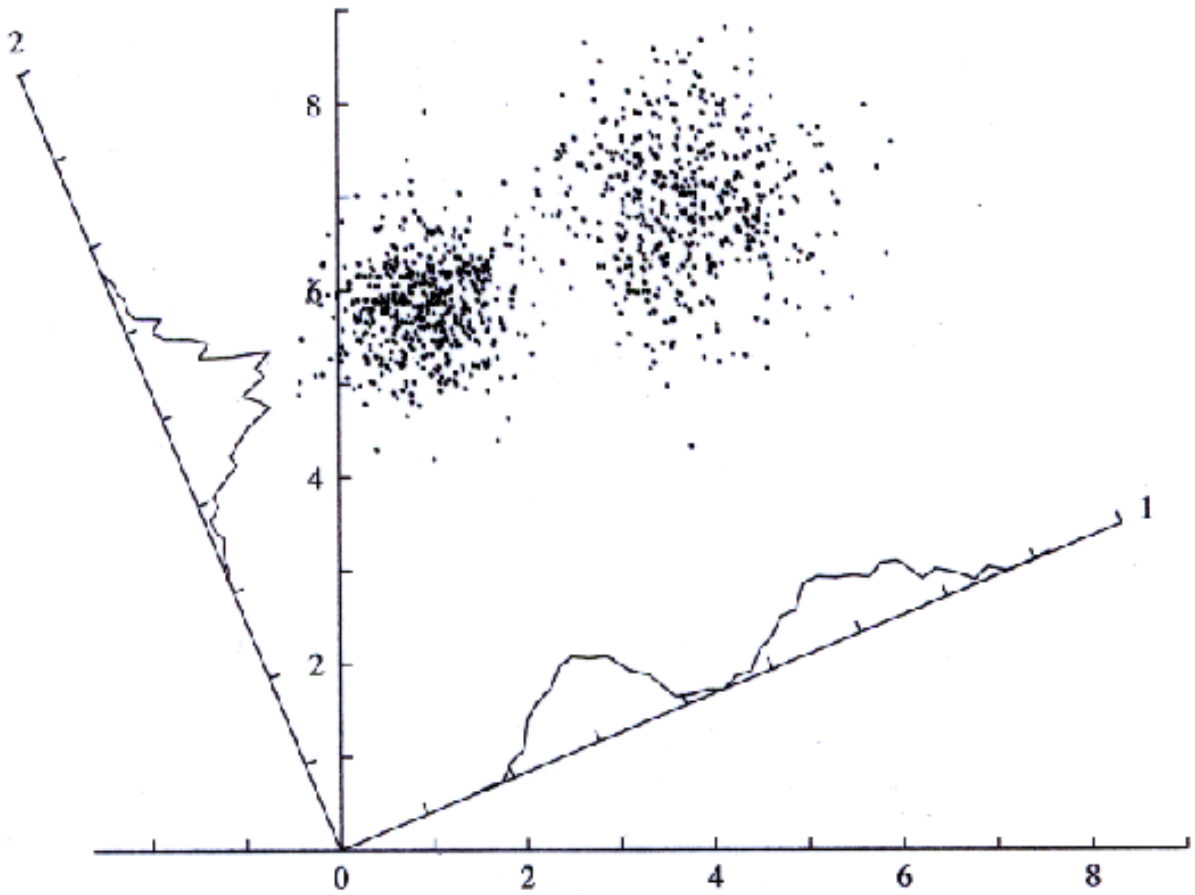


Figure: A possible projection on two new axis

PCA : Calculating the new variables (1/6)

Let us denote M the centered version of the data set X of n rows and d columns, u_1 the first loading vector, and $\pi_{u_1}(M) = M \cdot u_1$ the projection of the data set on this component.

- We seek u_1 of norm 1 so that the projection of the data on u_1 has a maximum variance.

Computing u_1

$$u_1 = \underset{||u||=1}{\text{argmax}} \left(\sum_{i=1}^{p} (\pi_u(M_i))^2 \right)$$

$$u_1 = \underset{||u||=1}{\text{argmax}} \left(||M \cdot u||^2 \right) = \underset{||u||=1}{\text{argmax}} \left(u^T \cdot M^T M \cdot u \right)$$

PCA : Calculating the new variables (2/6)

- We seek u_1 of norm 1 so that the projection of the data on u_1 has a maximum variance.

Computing u_1

Let us note V_π the variance of $\pi_u(M)$:

$$V_\pi = \frac{1}{N-1}\pi_u(M)^T \cdot \pi_u(M) = u^T \cdot \underbrace{\frac{M^T M}{N-1}}_{C} \cdot u = u^T \cdot C \cdot u$$

PCA : Calculating the new variables (2/6)

- We seek u_1 of norm 1 so that the projection of the data on u_1 has a maximum variance.

Computing u_1

Let us note V_π the variance of $\pi_u(M)$:

$$V_\pi = \frac{1}{N-1}\pi_u(M)^T \cdot \pi_u(M) = u^T \cdot \underbrace{\frac{M^T M}{N-1}}_{C} \cdot u = u^T \cdot C \cdot u$$

- With C the variance-covariance matrix of the original dataset X.

PCA : Calculating the new variables (2/6)

- We seek u_1 of norm 1 so that the projection of the data on u_1 has a maximum variance.

Computing u_1

Let us note V_π the variance of $\pi_u(M)$:

$$V_\pi = \frac{1}{N-1}\pi_u(M)^T \cdot \pi_u(M) = u^T \cdot \underbrace{\frac{M^T M}{N-1}}_{C} \cdot u = u^T \cdot C \cdot u$$

- With C the variance-covariance matrix of the original dataset X.
- Note that it only works because all variables M_i are centered !

PCA : Calculating the new variables (3/6)

Computing u_1

From $V_\pi = u^T \cdot C \cdot u$, maximizing V_π is equivalent to solve:

$$C \cdot u = V_\pi \cdot u$$

By definition, the solution of this equation is the set of couples : eigenvectors / eigenvalues of C.

- The largest eigenvalue λ_1 is the variance V_π, the maximum reachable.
- The associated eigenvector is the projection axis u_1 along which V_π is maximum.

PCA : Calculating the new variables (4/6)

The above reasoning has allowed us to conclude that the vector which explains the more inertia (i.e. the variance) of the data points is the first eigenvector. Similarly, the second vector which explains most of the remaining inertia is the second eigenvector, etc.

- We also saw that the variance of the data projected onto the n^{th} eigenvector (also called - variance explained by this axis) is λ_n.

Remark

In Matrix form, we have: $C \propto U \Lambda U^T$ where Λ is a diagonal matrix containing all single values.

PCA : Calculating the new variables (5/6)

Computing u_k

To find the k^{th} component, the data are updated:

$$\tilde{X}_k = X - \sum_{s=1}^{k-1} X \cdot u_s \cdot u_s^T$$

The process that we have just presented is repeated by finding the new maximal eigenvectors of the variance-covariance matrix of the new data.

PCA : Calculating the new variables (5/6)

Computing u_k

To find the k^{th} component, the data are updated:

$$\tilde{X}_k = X - \sum_{s=1}^{k-1} X \cdot u_s \cdot u_s^T$$

The process that we have just presented is repeated by finding the new maximal eigenvectors of the variance-covariance matrix of the new data.

It turns out that this process gives the remaining eigenvectors of the original variance-covariance matrix.

- Therefore, in practice the k^{th} component is picked by choosing the k^{th} largest eigenvalue and the associated eigenvector.

PCA : Calculating the new variables (6/6)

Algorithm

1. Choose p the number of dimensions that you want to keep.
2. Select the eigenvectors $(u_1, \cdots, u_p)$ associated with the p largest eigenvalues.
3. Project the data on the new axes in order to obtain a description of these data according to the new variables (called principal components): $\pi_u(M) = M \cdot u$.
4. If the number of dimensions is ≤ 3, it is possible to represent the data in the new space.

PCA : Calculating the new variables

Another possible algorithm

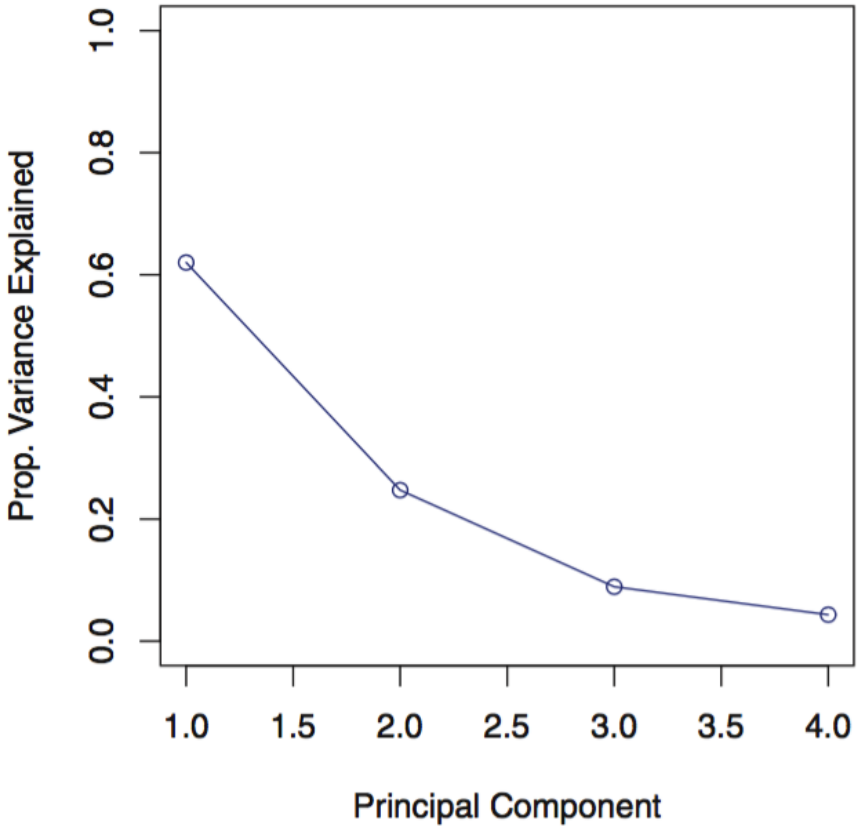
1. Choose P the proportion of information (also called inertia) that we want to keep.
2. Select y eigenvectors so that:

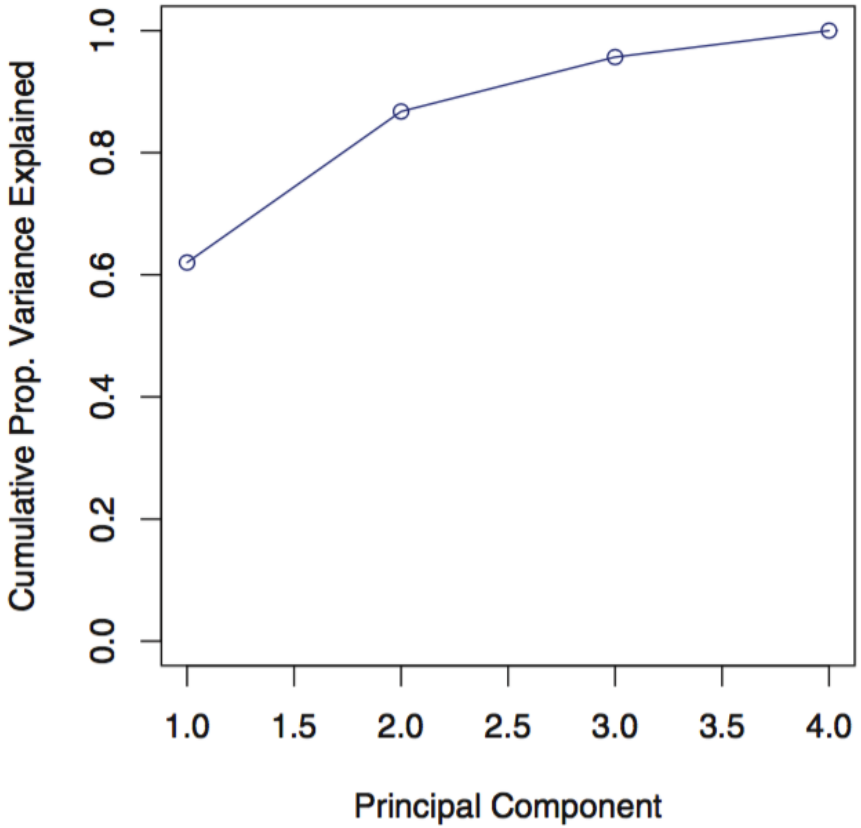
$$y = \underset{y}{\text{argmin}} \frac{\sum_{i=1}^{y} \lambda_i}{\sum \lambda} \geq P$$

3. Project the data on the new axes.
4. If the number of dimensions is ≤ 3, it is possible to represent the data in the new space.

PCA : choosing the number of components

We typically decide on the number of principal components required by examining a scree plot. Look for a point at which the proportion of variance explained by each subsequent principal component drops off, this is often referred to as an *elbow* in the scree plot.





Reminders

Reminder: eigenvalues

The eigenvalues of a matrix X are determined by the vector λ so that:

$$|X - \lambda I| = 0$$

Where $|\cdot|$ is the determinant operator.

Reminder: eigenvectors

The eigenvector v_i linked to a eigenvalue λ_i solves the following equation:

$$X \cdot v_i = \lambda_i \cdot v_i$$

Computing the correlations

We know that the new variables represent linear combinations of the old ones. It is therefore interesting to analyze the correlation between the old and new variables to interpret the results.

Correlations between an old variable and a new variable

$$r(\mathrm{M}_i, \pi_j) = \frac{1}{N} \frac{{}^T\mathrm{M}_i \cdot \pi_j}{\sigma_i \sqrt{\lambda_j}}$$

This expression follows directly the original formula for the correlation coefficient, recalling that λ_j is the variance of the projection of the data on the new u_j axis.

Circle of correlations (1/3)

- Now that we know the correlation between the old and new variables, we can simply visualize a **circle of correlations**.
- This allows to both characterize the new variables (which makes it possible to visually interpret the data) and to visualize the correlation between the old variables.

Circle of correlations (2/3)

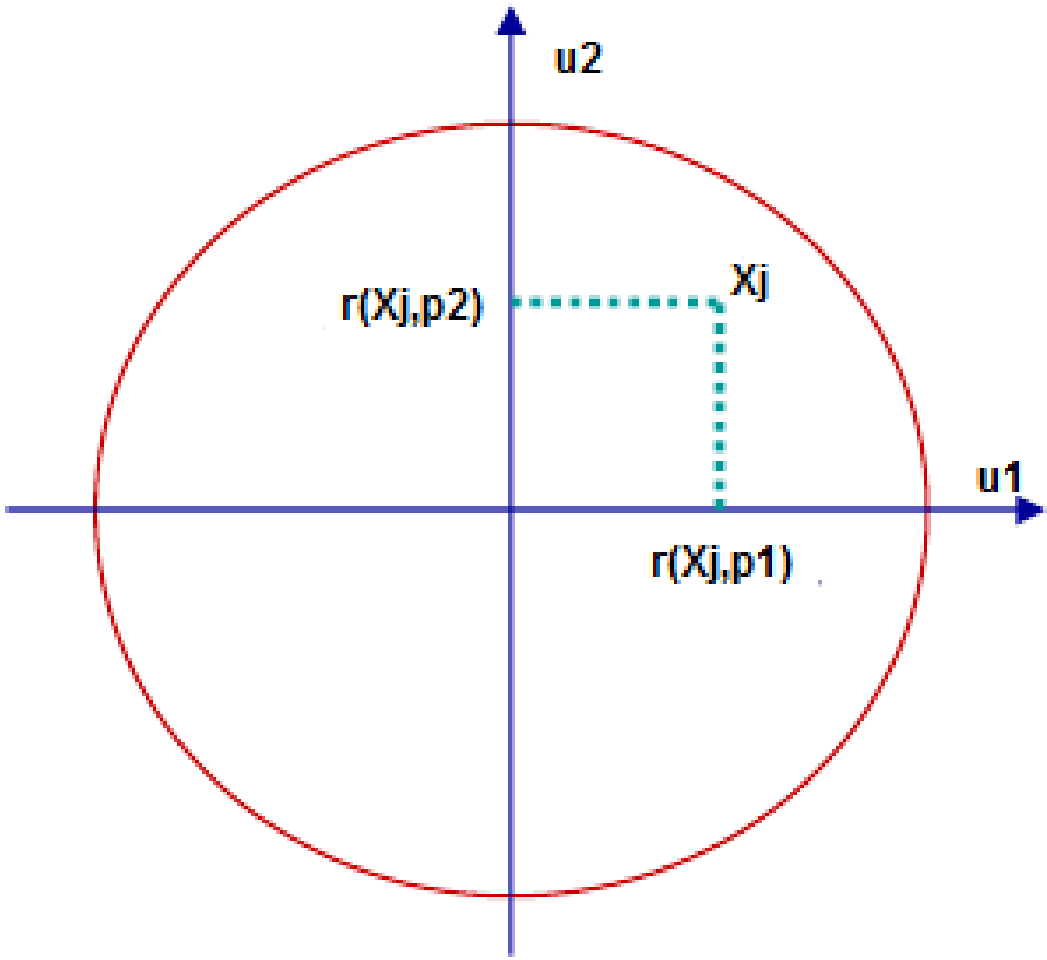
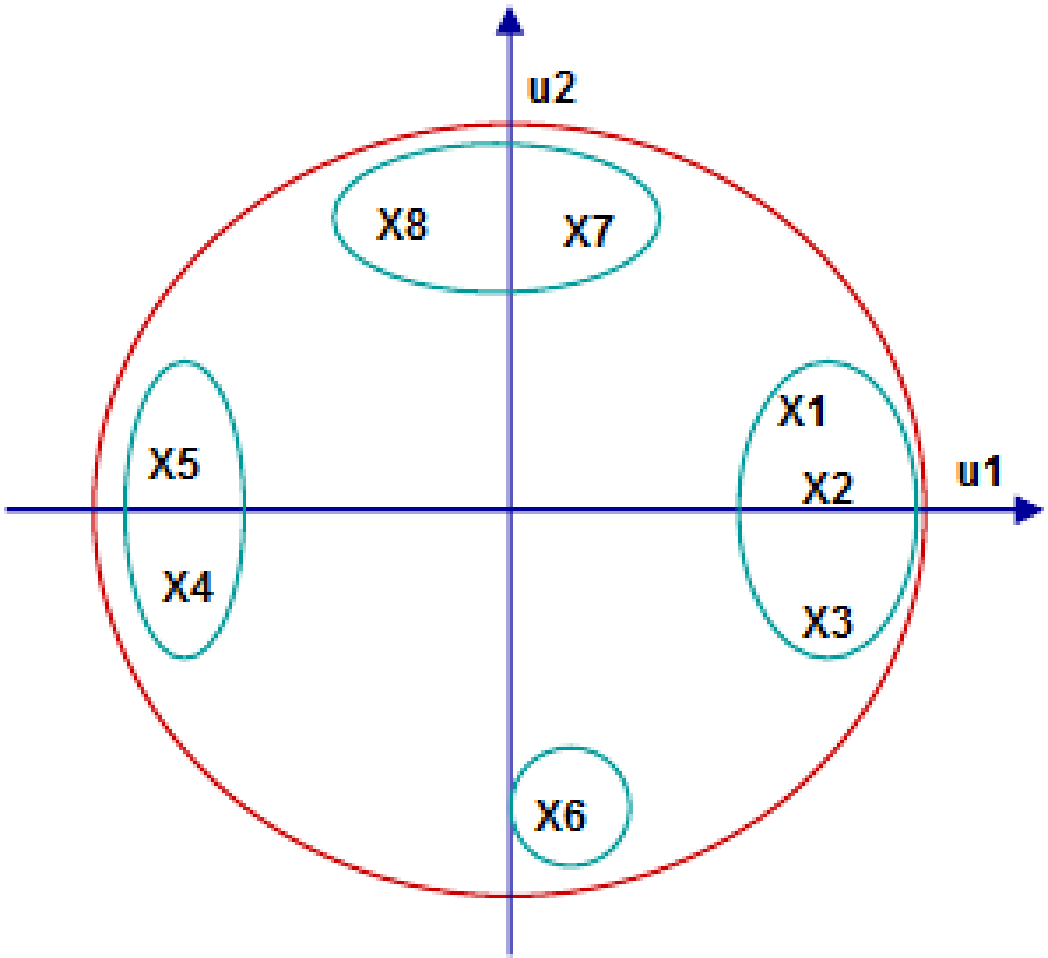


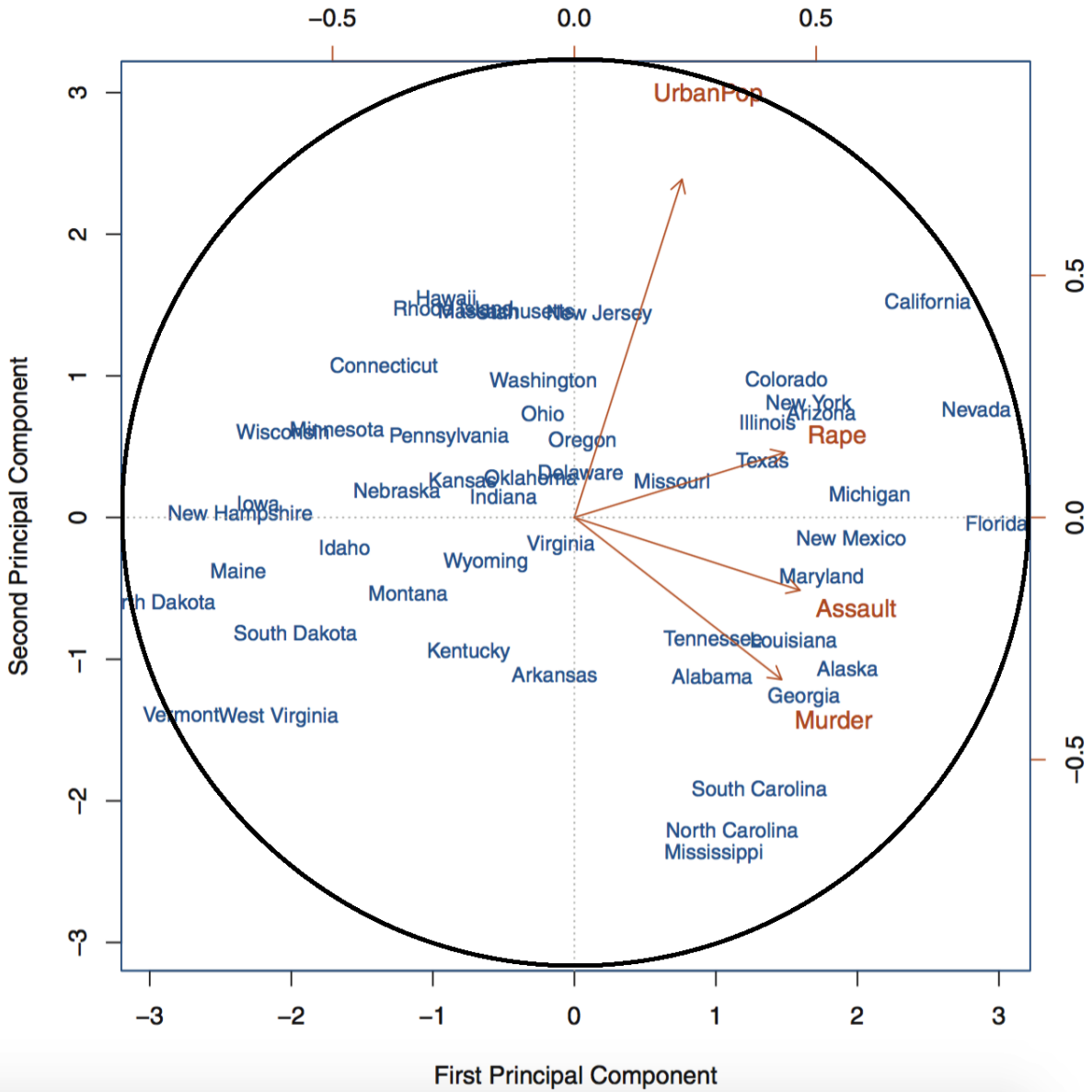
Figure: How to project on a circle of correlations

Circle of correlations (3/3)

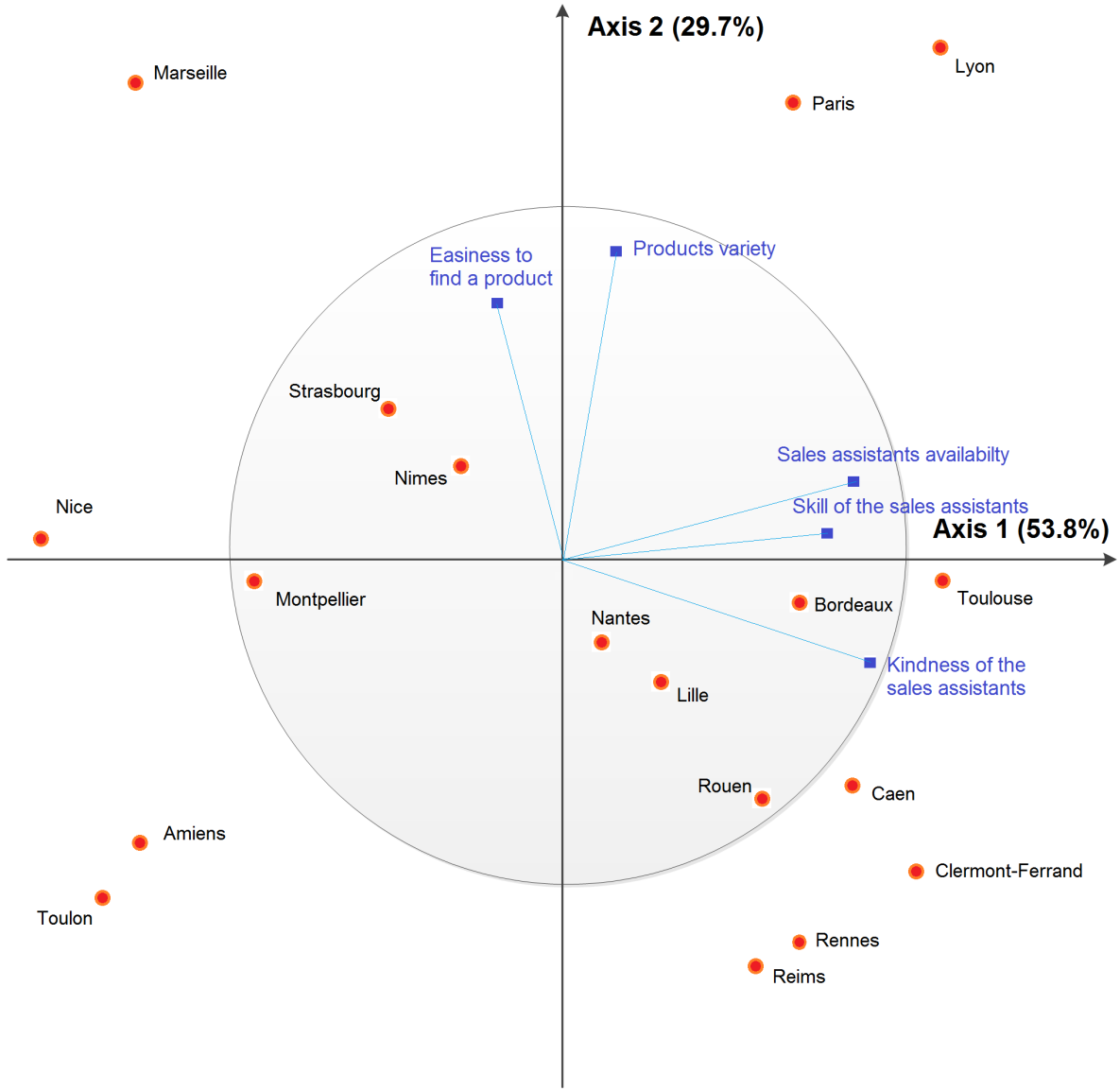


A circle of correlations makes it possible to see how the old variables relate with the new ones, but also how they relate to each other.

Circle of correlations: Example



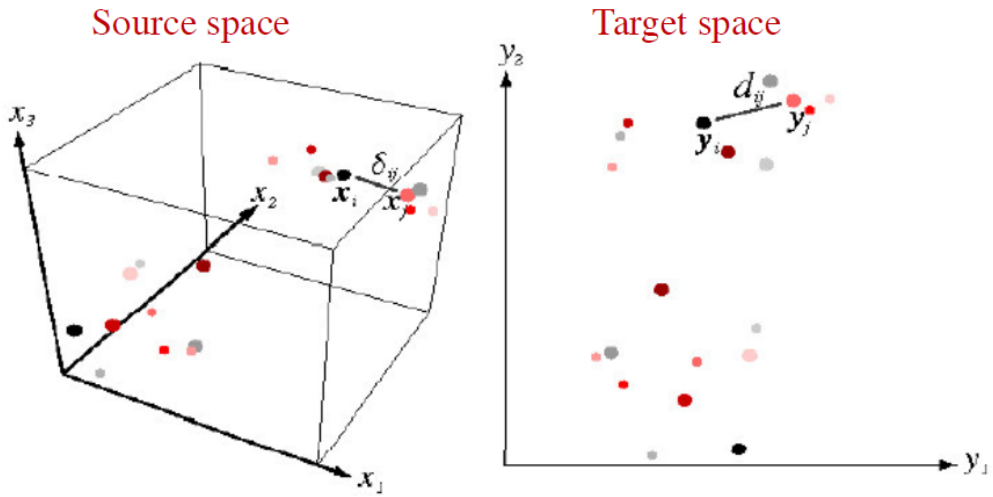
Circle of correlations: Example



Multi-dimensional scaling

Multi-dimensional scaling (MDS) originally proposed by Borg and Groenen in 1997:

- MDS encompasses a collection of reduction techniques that maps the distances between observations in a high dimensional space into a lower dimensional one.
- It aims at finding a configuration of the points in the low dimensional space so that the inter-point distances remain mostly proportional to the one in higher dimension.



Multi-dimensional scaling

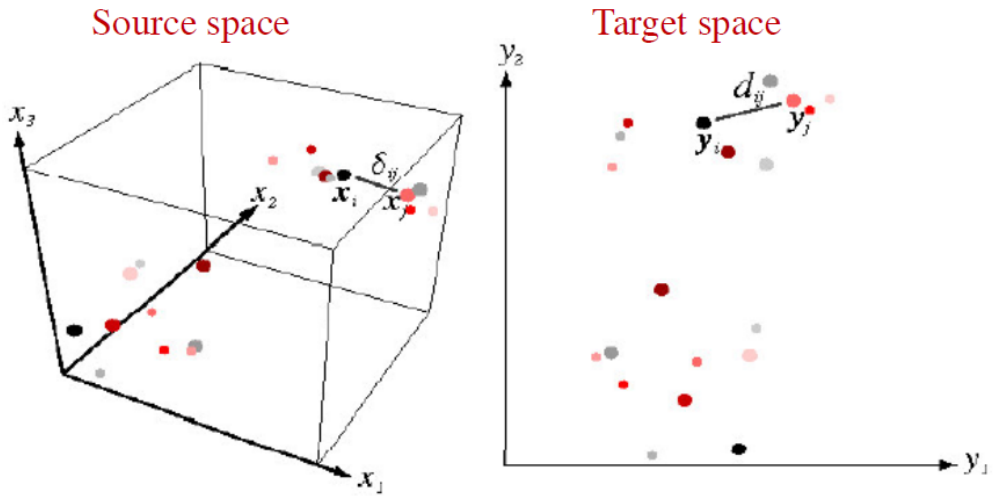
Besides dimension reduction, MDS is well suited for applications or algorithms based on the distances between the data, rather than the data themselves:

- Re-building the map of a country based on the distance between the cities.
- Building a 2D map from 3D coordinates (e.g. star constellation map).
- etc.

Remarks

- With $\frac{N(N-1)}{2}$ distances, it is always possible to generate the position of N points in a N dimension space.
- MDS computes an approximation for a lower dimensional space.

Multi-dimensional scaling: Problem formulation



We have

- The points $x_1, \cdots, x_N$ in D dimensions
- Any distance δ_{ij} between points x_i and x_j

We want to find

- The points $y_1, \cdots, y_N$ in 2 or 3 dimensions
- The distance d_{ij} between any y_i and y_j so that it is close to δ_{ij}

Multi-dimensional scaling: Cost function

- We must search d_{ij} so that it minimizes an objective function.
- We can define such function in a general manner:

$$Cost = \sum_{i<j} (d_{ij} - \delta_{ij})^2$$

$$\delta_{ij} = ||x_i - x_j||^2$$

$$d_{ij} = ||y_i - y_j||^2$$

Multi-dimensional scaling: Stress functions

Given that d_{ij} is a function of y_i and y_j (the δ_{ij} are constant), thereafter are a few possible objective functions:

$$J_{aa} = \frac{\sum_{i<j}(d_{ij} - \delta_{ij})^2}{\sum_{i<j}\delta_{ij}^2}$$ penalizes large absolute errors

$$J_{rr} = \sum_{i<j}\left(\frac{d_{ij} - \delta_{ij}}{\delta_{ij}}\right)^2$$ penalizes large relative errors

$$J_{ar} = \frac{1}{\sum_{i<j}\delta_{ij}}\sum_{i<j}\frac{(d_{ij} - \delta_{ij})^2}{\delta_{ij}}$$ A compromise between the two

The last one, J_{ar}, is also referred as **Sammon Criterion**.

Multi-dimensional scaling: Algorithm

- Choose an objective function J and a step parameter $0 < \eta < 1$.
- Compute the distances δ_{ij} if they are not provided.
- Initialize the points $y_1, \cdots, y_N$ randomly and compute the corresponding d_{ij}.
- Repeat until convergence:

$$\forall i \quad y_i \leftarrow y_i - \eta \nabla J(y_i)$$

Multi-dimensional scaling: Update Rule

$$\nabla J_{aa}(y_k) = \frac{2}{\sum_{i<j} \delta_{ij}^2} \sum_{j \neq k} (d_{kj} - \delta_{kj}) \frac{y_k - y_j}{d_{kj}}$$

$$\nabla J_{rr}(y_k) = 2 \sum_{j \neq k} \frac{d_{kj} - \delta_{kj}}{\delta_{kj}^2} \frac{y_k - y_j}{d_{kj}}$$

$$\nabla J_{ar}(y_k) = \frac{2}{\sum_{i<j} \delta_{ij}} \sum_{j \neq k} \frac{d_{kj} - \delta_{kj}}{\delta_{kj}} \frac{y_k - y_j}{d_{kj}}$$

Multi-dimensional scaling: Example 1

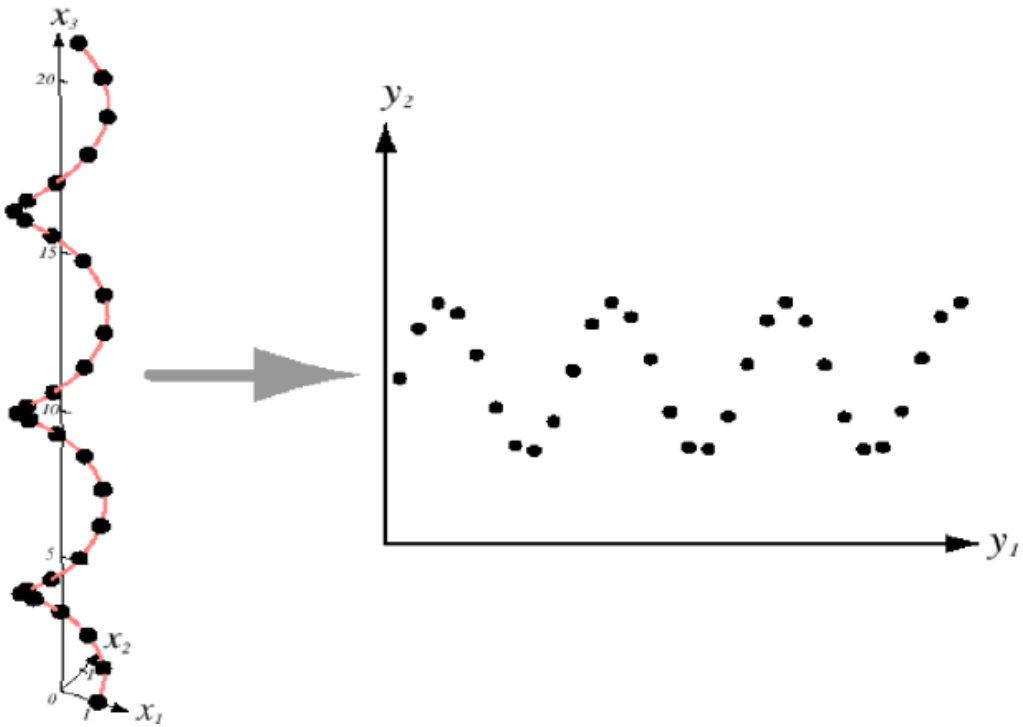


Figure: From 3D to 2D using MDS

Multi-dimensional scaling: Example 2

- The dataset "Eurodist" represents the distance in kilometers between 21 European cities.
- The source data set is a square matrix containing the distances between the cities:

	Athens	Barcelona	Brussels	Calais	Cherbourg	Cologne	Copenhagen	Geneva	Gibraltar	Hamburg
Barcelona	3313									
Brussels	2963	1318								
Calais	3175	1326	204							
Cherbourg	3339	1294	583	460						
Cologne	2762	1498	206	409	785					
Copenhagen	3276	2218	966	1136	1545	760				
Geneva	2610	803	677	747	853	1662	1418			
Gibraltar	4485	1172	2256	2224	2047	2436	3196	1975		
Hamburg	2977	2018	597	714	1115	460	460	1118	2897	

Multi-dimensional scaling: Example 2

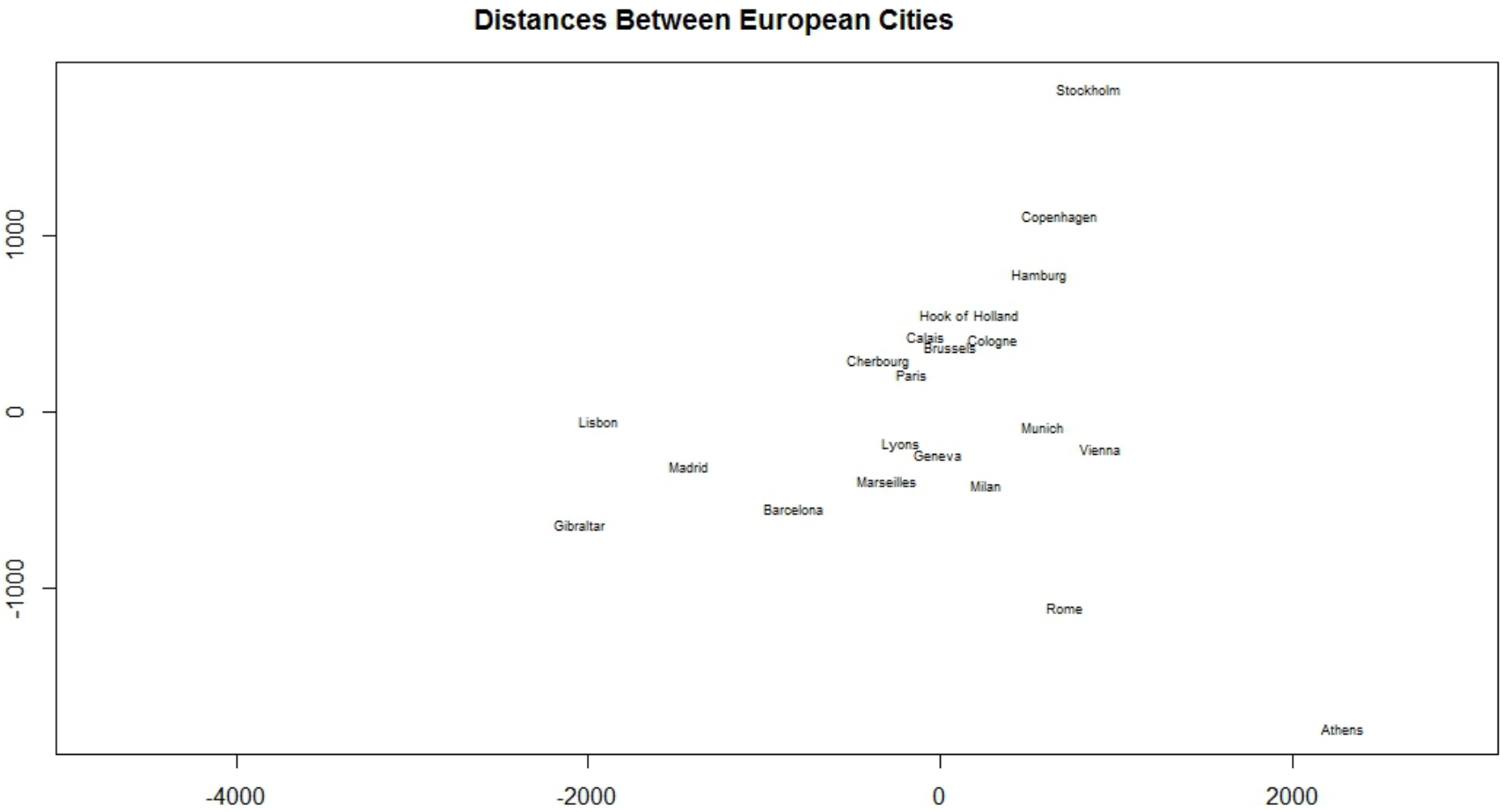


Figure: Reconstructed map of Europe using MDS

Multi-dimensional scaling: Conclusions

The different MDS algorithms differ in the following points:

- The distance used in the source space.
- The Stress (objective) functions: Using a different stress function will lead to a different result.
- The optimization procedure: Linear MDS can be solved analytically but cannot model complex (non-linear) low dimensional manifolds well. However, non-linear MDS requires a more complex and iterative algorithm.

Limits of linear methods (1/2)

Techniques such as LDA, PCA and their variants perform a global transformation of the data using basic operations:

- Rotation
- Translation
- Rescaling

The problematic of non-linearity

- Using the 3 operations, linear methods assume that most of the information in the data is contained in a linear subspace.
- This raises the question of how to deal with data that are actually embedded in a non-linear subspace (a low-dimensional manifold).

Limits of linear methods (2/2)

Linear methods such as PCA cannot discover the structure of a data set with a non-linear shape.

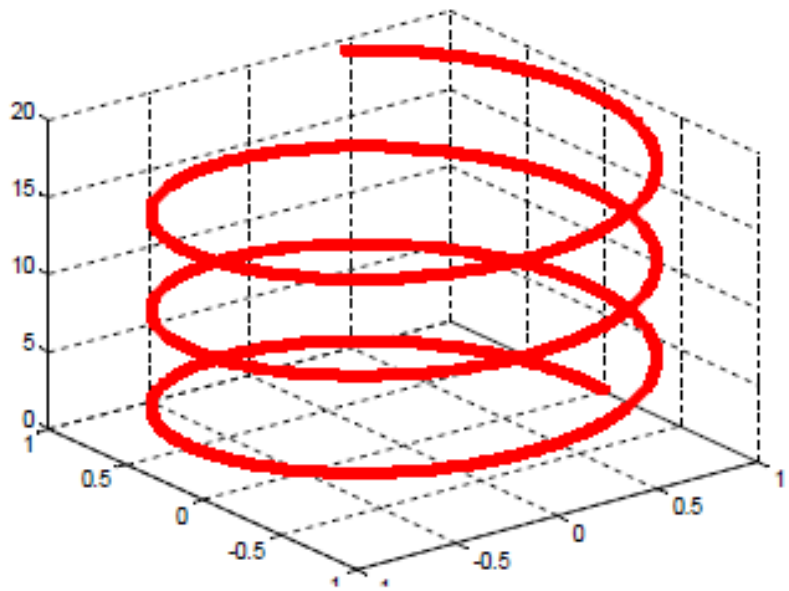


Figure: Example of a spiral data set that could be modeled in 2D or even 1D, but that PCA and non-linear methods cannot handle.

Euclidian distance VS Geodesic distance

- Euclidian based distances are one of the main problem when dealing with data embedded in a non-linear subspace.
- Custom distances embedded in the low-dimensional manifold, such as the **geodesic distance**, can help dealing with such data.

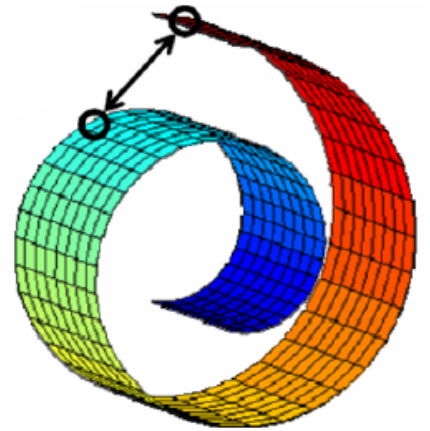


Figure: Euclidian distance

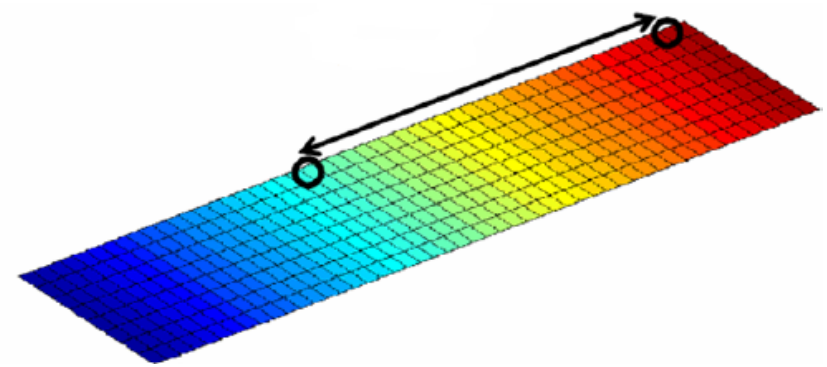


Figure: Geodesic distance

ISOMAP: Principle

The **Isometric feature mapping** (Isomap) [Tenebaum et al. 2000] is a reduction method that can handle non-linear subspaces. It is based on the following principles:

- For neighboring samples (close data points), the Euclidian distance provides a good approximation of the geodesic distance.
- For distant points, the geodesic distance can be approximated using an Euclidian distance-based **neighborhood graph** of the data and searching the shortest path between the two points.

ISOMAP: Algorithm

1. Build a distance neighborhood graph G using the Euclidian distance in the input space: This can be done using KNN, or by connecting the points within a radius ϵ.
2. Approximate the geodesic distance between all data points: Compute the shortest path between all points of the graph using **Dijkstra** algorithm or **Floyd** algorithm.
3. Apply the **Multi-dimensional scaling** (MDS) algorithm to the geodesic distance matrix from Step 2.

Remark

The Isomap algorithm can be seen as a non-linear variant of the MDS algorithm.

ISOMAP: Algorithmic complexity

The main issue of the Isomap algorithm is that it can be quite slow. For a data set of size N, an input space of dimension D, an output space of dimension d, and considering a neighborhood of size k in the first step, we have:

- Step 1: K-nearest neighbors $O(N^2 D)$
- Step 2: Djikstra $O(N^2 log(N) + N^2 k)$, Floyd $O(N^3)$
- Step 3: MDS algorithm $O(N^2 d)$

ISOMAP: The "Swiss roll" Example (1/3)

- The "swiss roll" data set contains 20000 points.
- We will show thereafter the development of the Isomap algorithm on this data set.

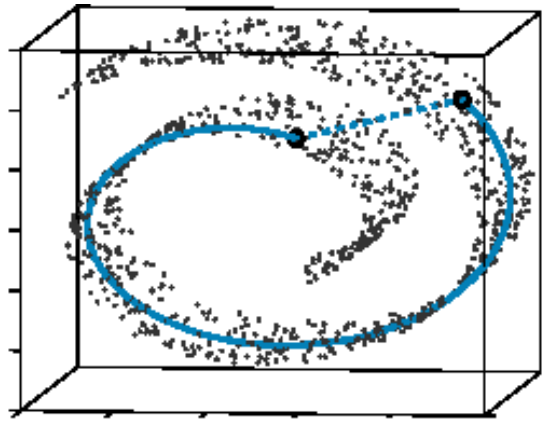


Figure: A 1000 points extract of the Swiss roll data set, with an example of Euclidian and Geodesic distances.

ISOMAP: The "Swiss roll" Example (2/3)

In the figure bellow, we show 3 illustrations of how the Isomap algorithm works on the Swiss roll data set with the following parameters:

- K-nearest neighbors (K=7)
- Approximation of the Geodesic distance: Djikstra algorithm

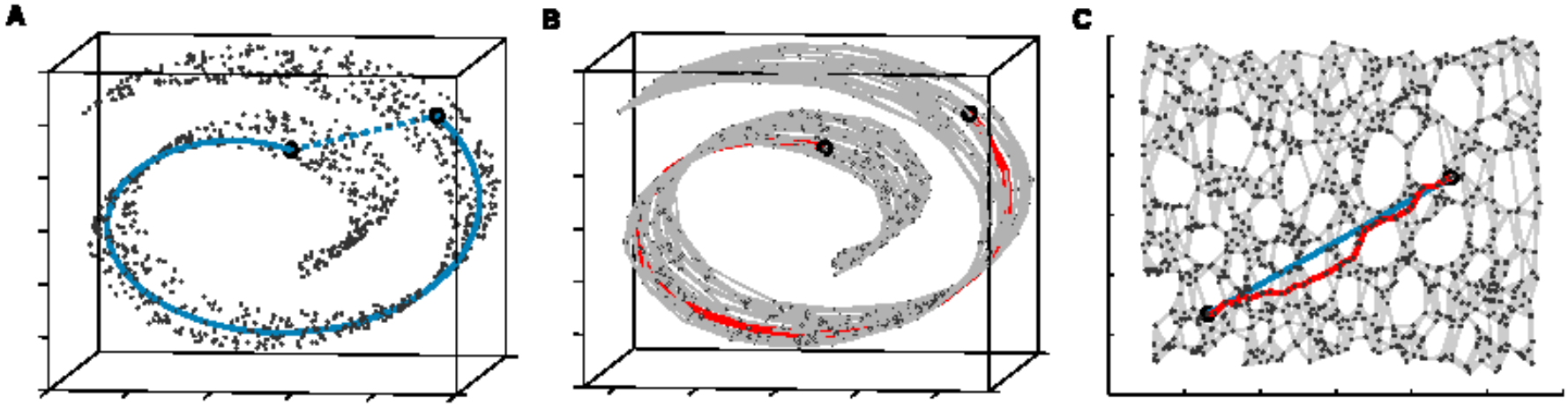


Figure: A 1000 points extract of the Swiss roll data set: The Euclidian distance is in dotted blue, the Geodesic distance in solid blue, and the approximated Geodesic distance is in red. [Tenebaum et al. 2000]

ISOMAP: The "Swiss roll" Example (3/3)

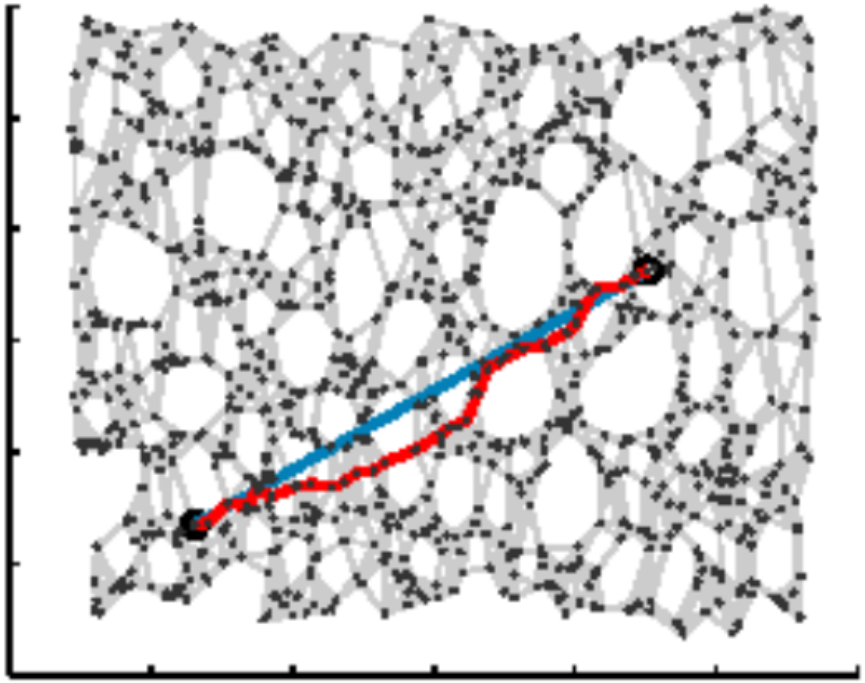


Figure: Final manifold in 2D that preserves pairwise geodesic distances. [Tenebaum et al. 2000]

ISOMAP: Images Example (1/2)

Figure: For each image we have $64 \times 64 = 4096$ pixels

ISOMAP: Images Example (2/2)

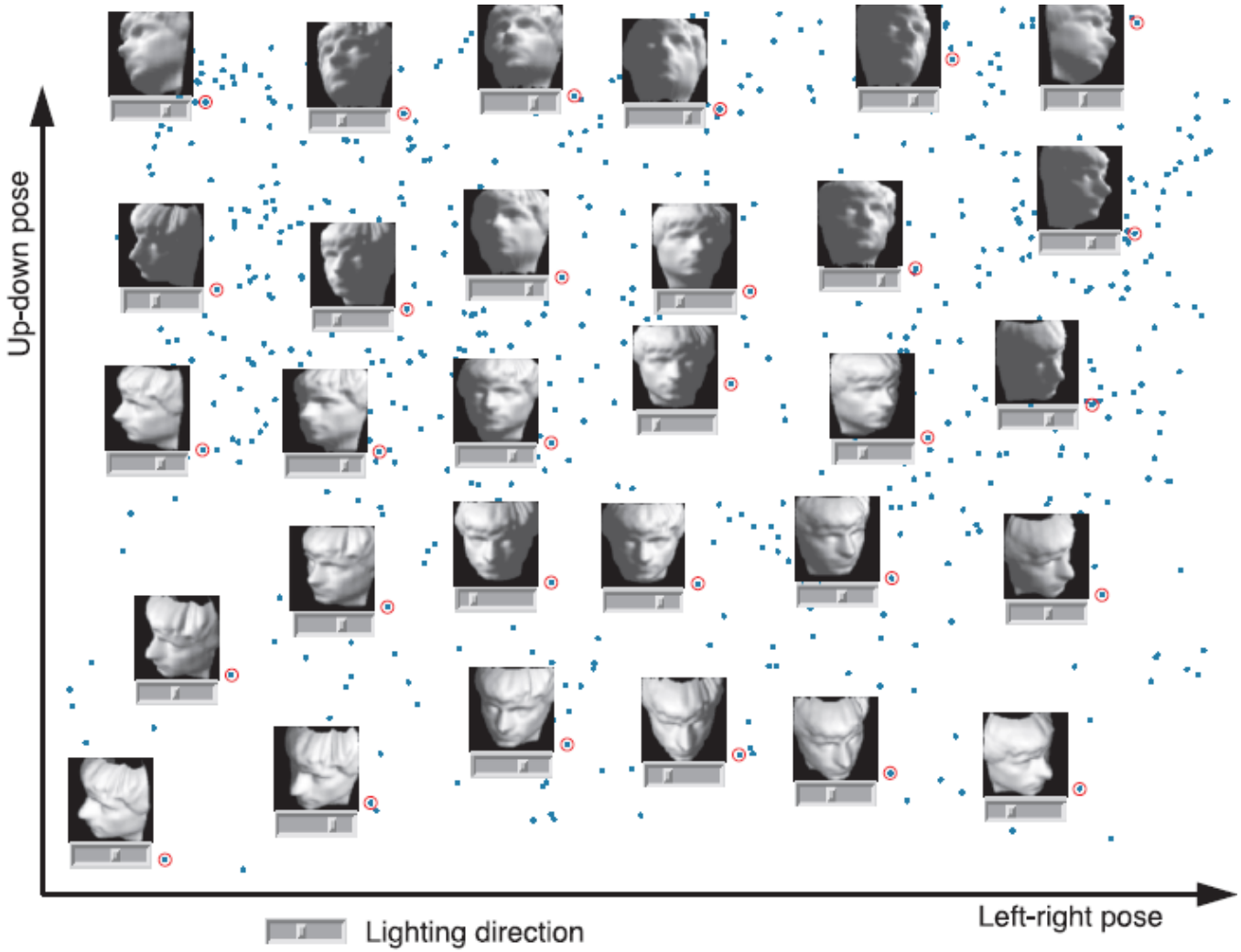
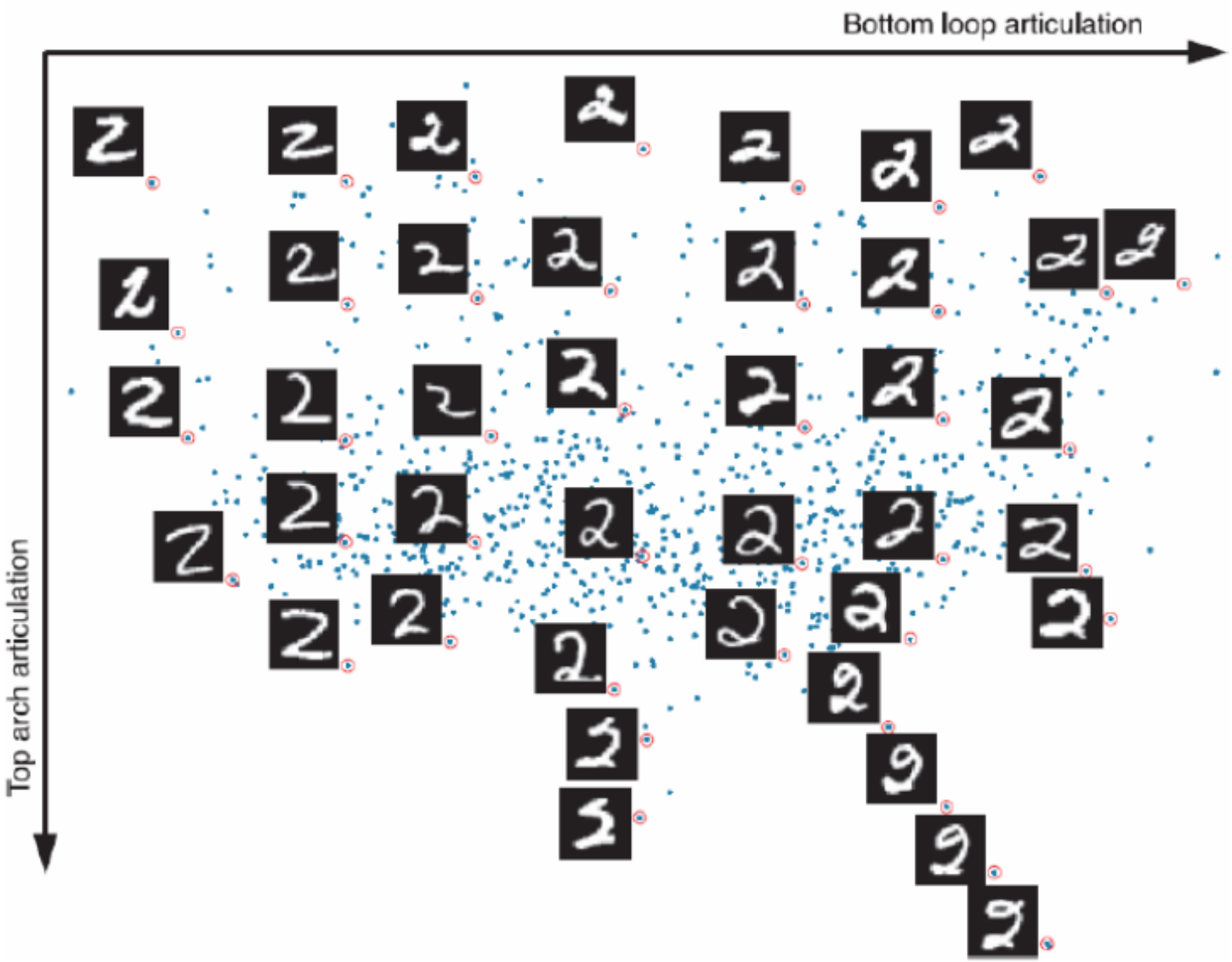


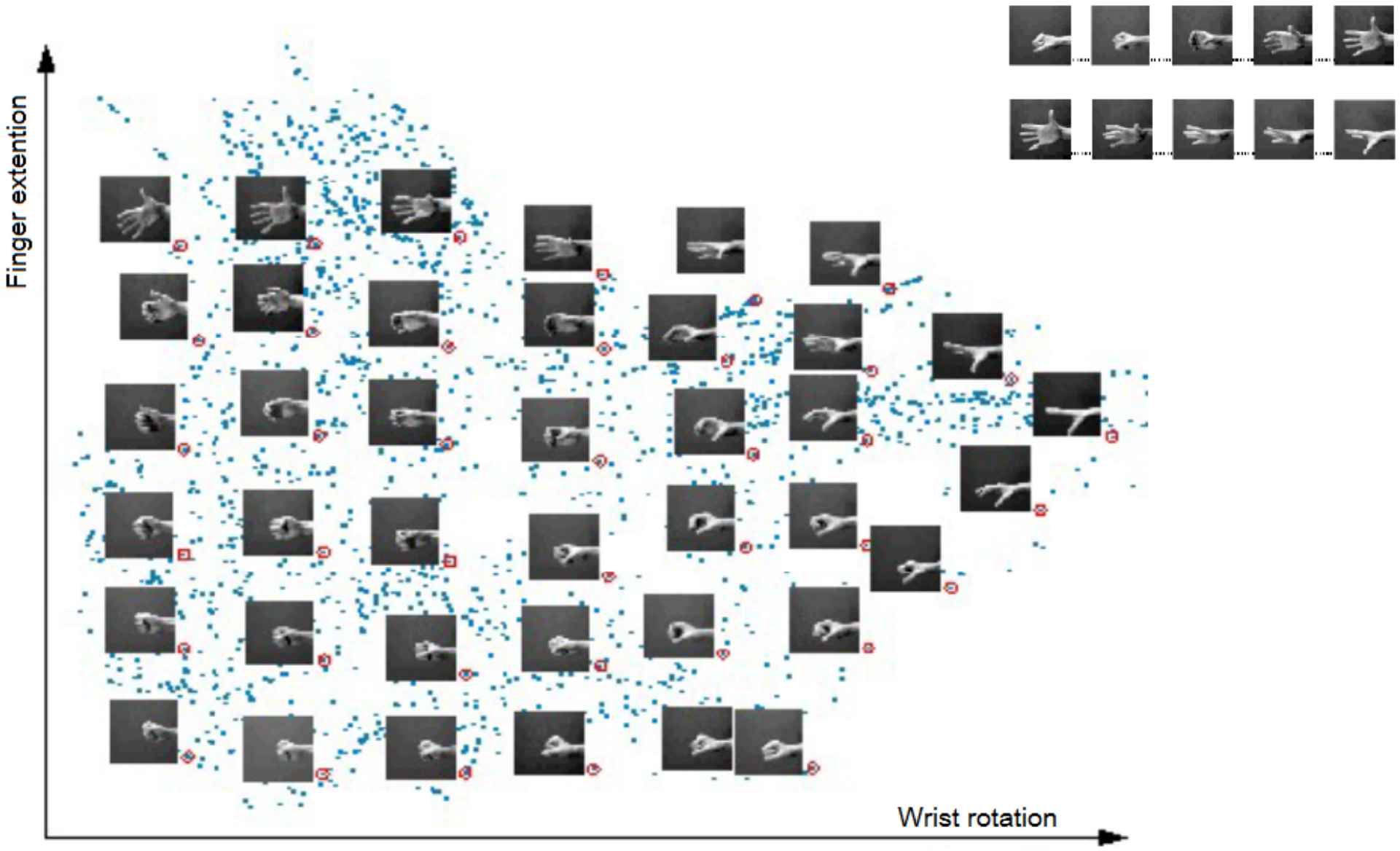
Figure: From D=4096 to d=2

ISOMAP: Hand writing Example



Original images

ISOMAP: Another Image Example



ISOMAP: Conclusion

Advantages of Isomap

- Handles non-linearity
- Is non-iterative
- Preserves the global properties of the data

Limitations of Isomap

- Sensitive to noise (it may create unwanted shortcut in the neighborhood graph)
- The optimal parameters k for KNN or ϵ can be difficult to determine:
  - Small k or ϵ may lead to linear shortcuts and a poor approximation of the geodesic distance, or in worst cases a partitioned graph.
  - Large k or ϵ slow the search for the shortest path, and may at some point loose the low-dimensional manifold.
- Relatively slow with large data sets $\sim O(N^2 log(N))$

Locally Linear Embedding: Principle

The locally Linear Embedding algorithm (LLE) [Roweis and Saul 2000] addresses the same problem as Isomap in a different way:

- It preserves the local properties of the data by representing each point by a linear combination of its nearest neighbors.
- It builds a projection to a linear low dimensional space preserving the neighborhood.

Locally Linear Embedding: Principle

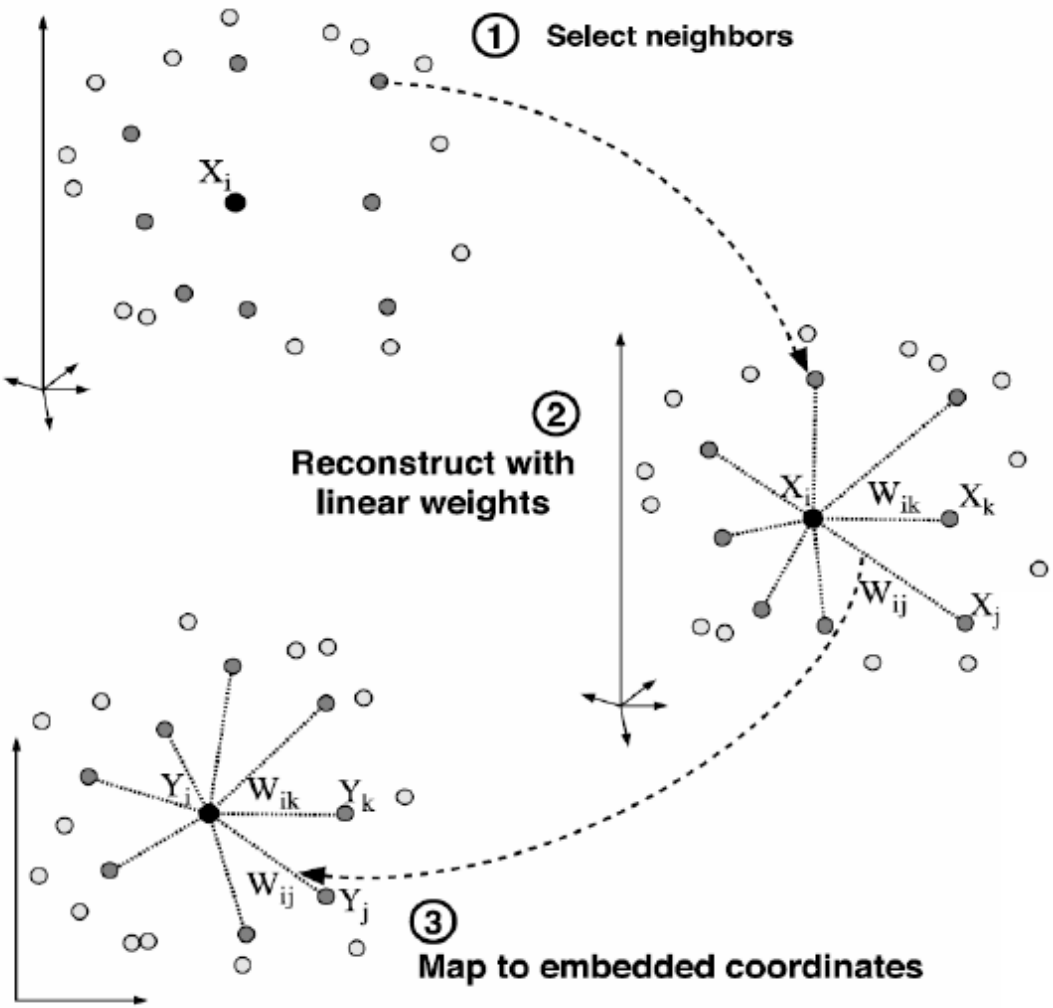
The locally Linear Embedding algorithm (LLE) [Roweis and Saul 2000] addresses the same problem as Isomap in a different way:

- It preserves the local properties of the data by representing each point by a linear combination of its nearest neighbors.
- It builds a projection to a linear low dimensional space preserving the neighborhood.

LLE algorithm

1. Compute the k nearest neighbors
2. Compute the weights needed to reconstruct each point using a linear combination of its neighbors
3. Project the results in a lower dimensional space using the weights

Locally Linear Embedding: Steps



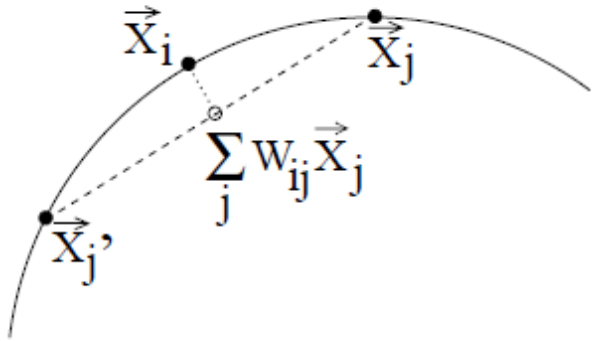


Figure: LLE algorithm [Roweis and Saul 2000]

Locally Linear Embedding: Computing the weights (1/2)

- The local geometry is modeled by linear weights that reconstruct each data point as a linear combination of its nearest neighbors.
- Let us note $W = (w_{ij})_{N \times N}$ the weight matrix, where w_{ij} is the weight given to x_j in the reconstruction of x_i.

Reconstruction error cost function

$$\epsilon(W) = \sum_{i=1}^{N} |x_i - \sum_{j \neq i} w_{ij} x_j|^2$$

The weights w_{ij} are minimized subject to two constraints:

1. Each data point is reconstructed only from its neighbors (i.e. $w_{ij} = 0$ is x_i and x_j are not neighbors)
2. Rows of the weight matrix W sum to 1: $\forall i \quad \sum_j w_{ij} = 1$

Locally Linear Embedding: Computing the weights (2/2)

Let us consider a sample x_i with its k nearest neighbors η_{ij}. With the constraint $\sum_j w_{ij} = 1$, the weights can be found in 3 steps:

Locally Linear Embedding: Computing the weights (2/2)

Let us consider a sample x_i with its k nearest neighbors η_{ij}. With the constraint $\sum_j w_{ij} = 1$, the weights can be found in 3 steps:

1. Compute the neighborhood correlation matrices C_{jk} and their inverse: $C_{jk} = (x_i - \eta_{ij}) \cdot (x_i - \eta_{ik})$

Locally Linear Embedding: Computing the weights (2/2)

Let us consider a sample x_i with its k nearest neighbors η_{ij}. With the constraint $\sum_j w_{ij} = 1$, the weights can be found in 3 steps:

1. Compute the neighborhood correlation matrices C_{jk} and their inverse: $C_{jk} = (x_i - \eta_{ij}) \cdot (x_i - \eta_{ik})$
2. Compute the lagrangian multiplier λ that enforces the constraint $\sum_j w_{ij} = 1$:

$$\lambda = \frac{1 - \sum_{j,k} C_{jk}^{-1}(x_i^T \eta_{ik})}{\sum_{j,k} C_{jk}^{-1}}$$

Locally Linear Embedding: Computing the weights (2/2)

Let us consider a sample x_i with its k nearest neighbors η_{ij}. With the constraint $\sum_j w_{ij} = 1$, the weights can be found in 3 steps:

1. Compute the neighborhood correlation matrices C_{jk} and their inverse: $C_{jk} = (x_i - \eta_{ij}) \cdot (x_i - \eta_{ik})$
2. Compute the lagrangian multiplier λ that enforces the constraint $\sum_j w_{ij} = 1$:

$$\lambda = \frac{1 - \sum_{j,k} C_{jk}^{-1}(x_i^T \eta_{ik})}{\sum_{j,k} C_{jk}^{-1}}$$

3. Compute the reconstruction weights:

$$w_{ij} = \sum_k C_{jk}^{-1}(x_i^T \eta_{ik} + \lambda)$$

Locally Linear Embedding: Mapping the data (1/2)

- Once the weights W have been computed, we seek to find the $Y = \{y_1, \cdots, y_n\}, y_i \in \mathbb{R}^d$ the new points in a low dimensional space.
- This can be done by searching Y that minimizes an embedding cost function similar to the reconstruction error cost function from the input space.

Embedding error cost function

$$\phi(Y) = \sum_{i=1}^{N} |y_i - \sum_{j \neq i} w_{ij} y_j|^2$$

Locally Linear Embedding: Mapping the data (2/2)

The embedding vectors Y_i are found by minimizing the cost function $\phi(Y)$.

- To make the optimization problem well-posed we introduce two constraints: $\sum_j y_j = 0$ and $\frac{1}{N}\sum_i Y_i Y_i^T = I$

Locally Linear Embedding: Mapping the data (2/2)

The embedding vectors Y_i are found by minimizing the cost function $\phi(Y)$.

- To make the optimization problem well-posed we introduce two constraints: $\sum_j y_j = 0$ and $\frac{1}{N}\sum_i Y_i Y_i^T = I$
- Then, the cost function becomes:

$$\phi(Y) = \sum_{i,j} M_{ij}(Y_i^T Y_j)$$

  - Where $M_{ij} = \delta_{ij} - W_{ij} - W_{ji} + \sum_k w_{ki} w_{kj}$
  - δ_{ij} is equal to 1 if $i = j$ and 0 otherwise

Locally Linear Embedding: Mapping the data (2/2)

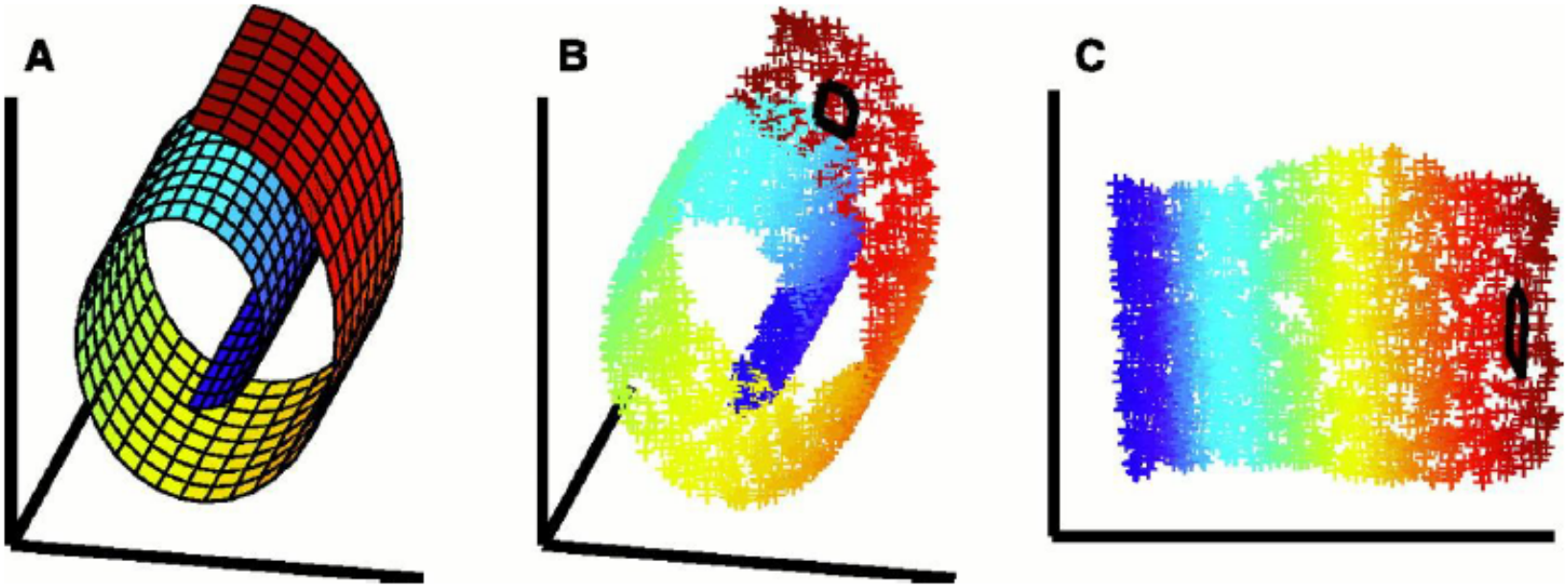
The embedding vectors Y_i are found by minimizing the cost function $\phi(Y)$.

- To make the optimization problem well-posed we introduce two constraints: $\sum_j y_j = 0$ and $\frac{1}{N}\sum_i Y_i Y_i^T = I$
- Then, the cost function becomes:

$$\phi(Y) = \sum_{i,j} M_{ij}(Y_i^T Y_j)$$

  - Where $M_{ij} = \delta_{ij} - W_{ij} - W_{ji} + \sum_k w_{ki} w_{kj}$
  - δ_{ij} is equal to 1 if $i = j$ and 0 otherwise
- The optimal embedding for a d-dimensional space is found by computing the bottom $d + 1$ eigenvectors of the matrix M.

Locally Linear Embedding: The "Swiss roll" Example



[Roweis and Saul, 2000]

Locally Linear Embedding: Face data set Example

- The initial data represent images of faces.
- In the 2D space, these images are grouped according to the position, lighting and expression.

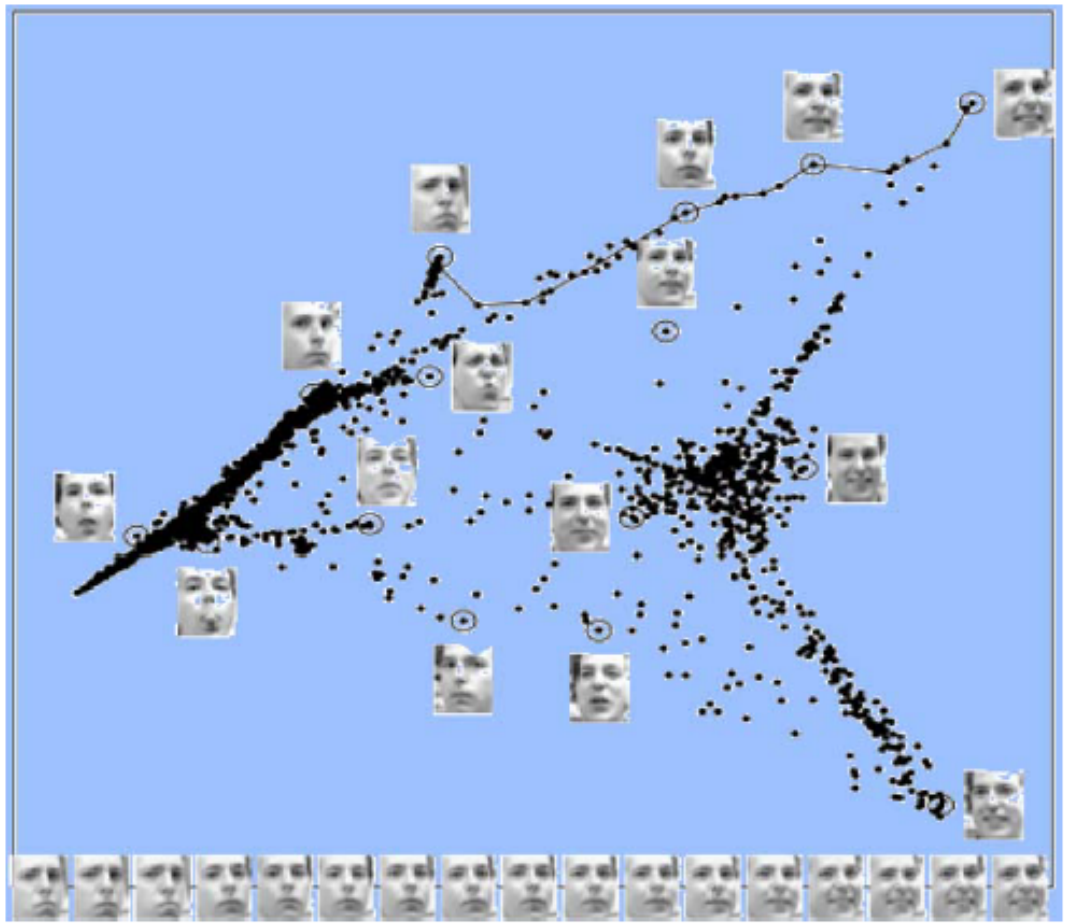
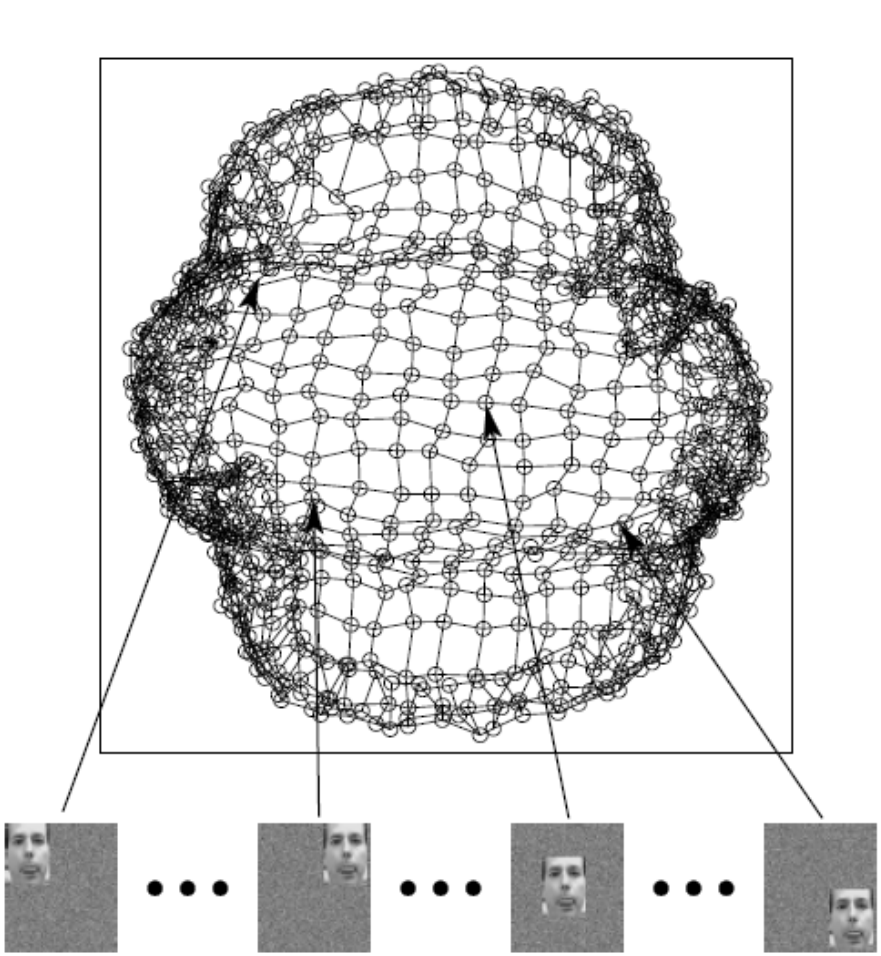


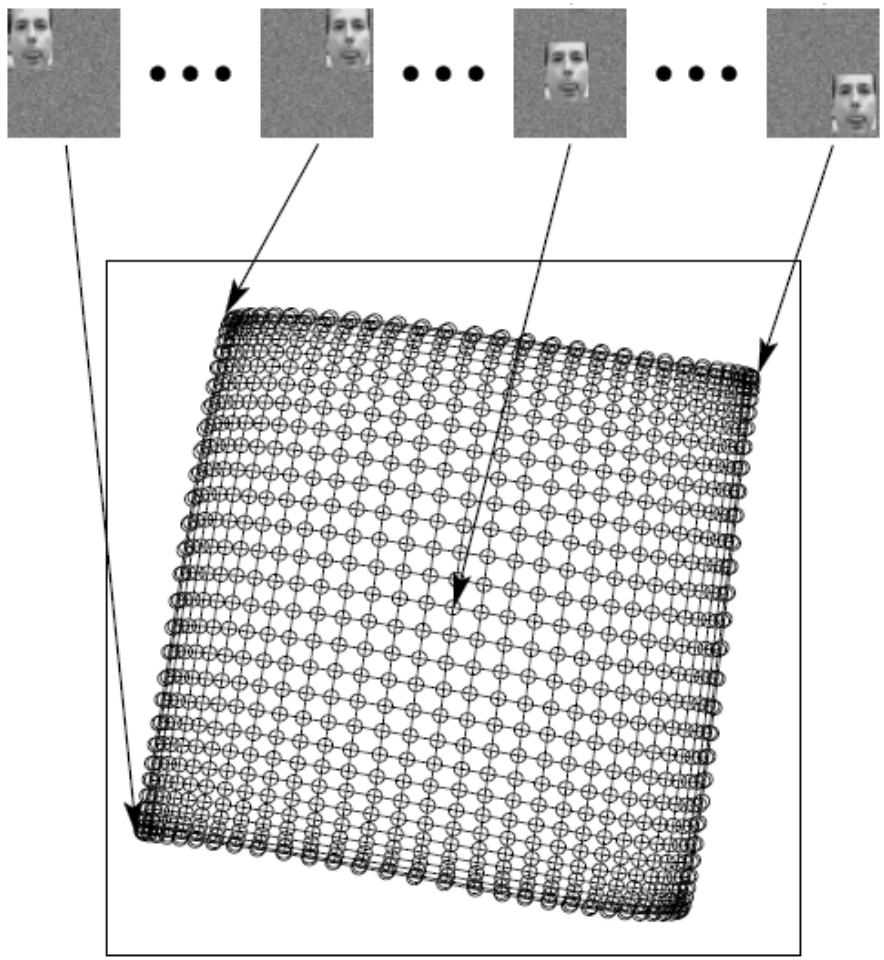
Figure: Images placed at the bottom of the figure correspond to successive points encountered on the line at the top right, sweeping a continuum of facial expression. [Roweis and Saul 2000]

Locally Linear Embedding: PCA VS LLE (1/2)

Unlike PCA, LLE preserves the local topology

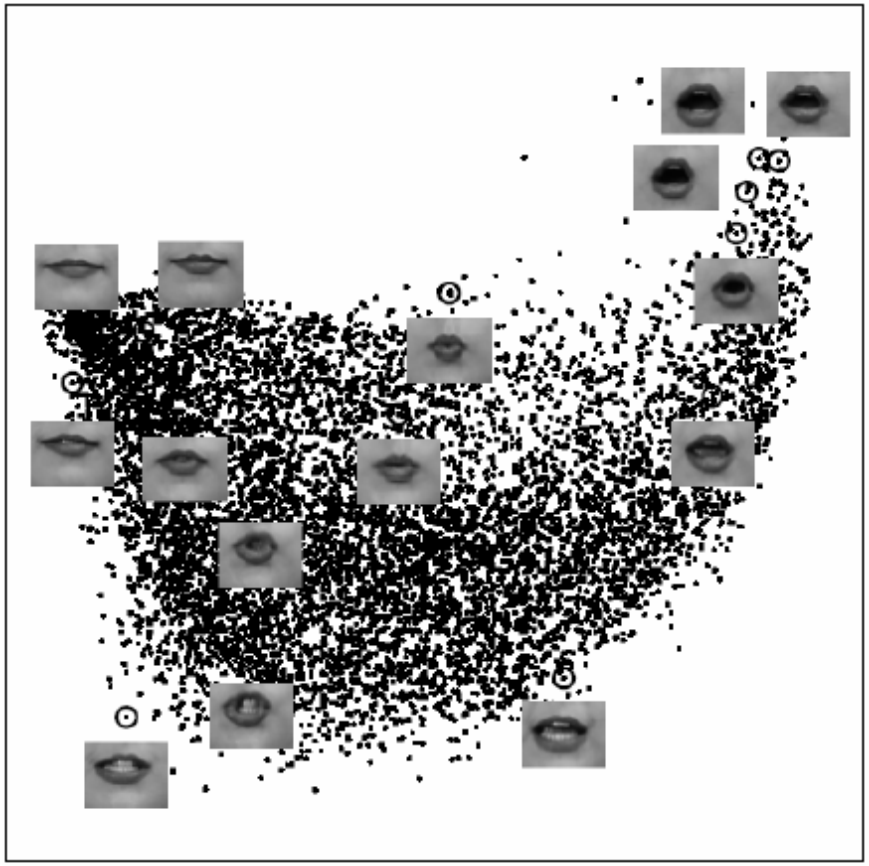


(a) PCA

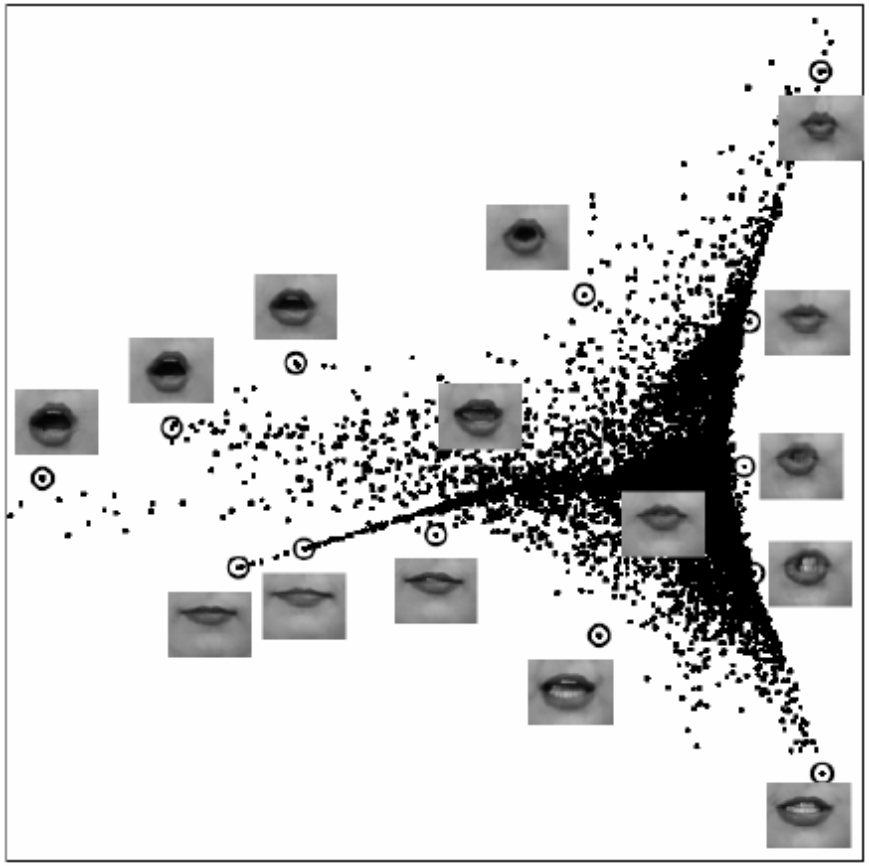


(b) LLE

Locally Linear Embedding: PCA VS LLE (2/2)



(c) PCA



(d) LLE

Locally Linear Embedding: Conclusions

Advantages of LLE

- Handles non-linearity
- Is non-iterative
- Preserves local topologies

Limitations of LLE

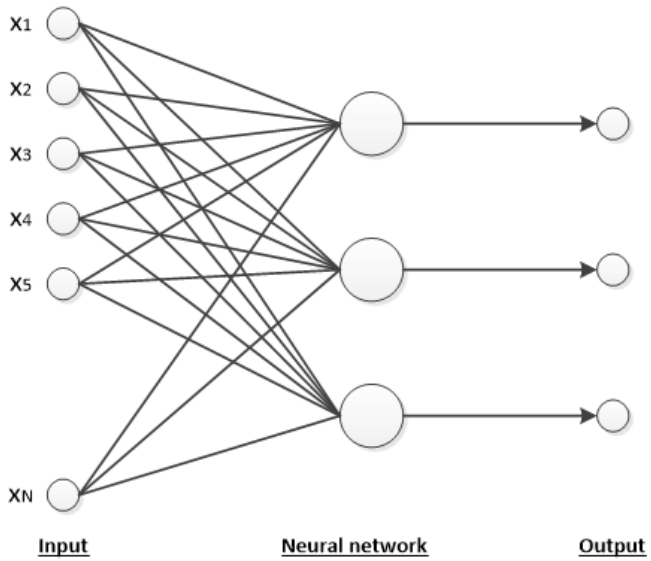
- Sensitive to noise
- The optimal parameters k for KNN can be difficult to determine:
- Relatively slow with large data sets due to the large number of covariance matrices inversions.

Outline

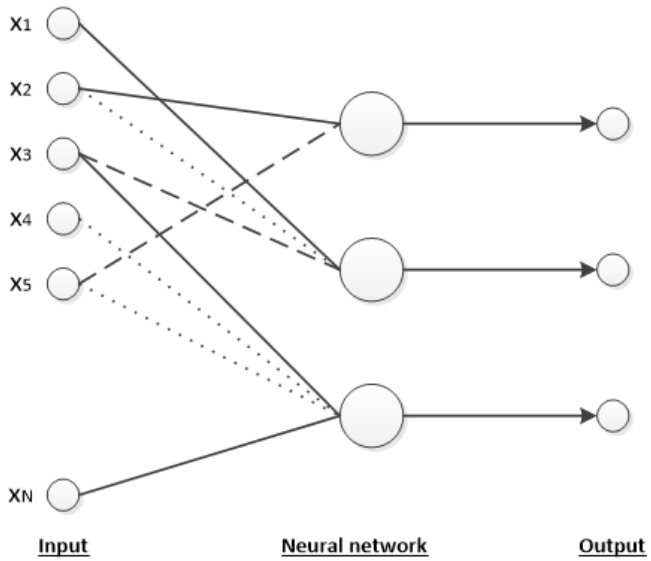
Neural networks

Neural networks are a variety of algorithms that mimic brain cells:

- Each neuron, or each group of neuron specializes in recognizing certain patterns.



(e) Untrained neural network



(f) Trained neural network

Figure: Example of a single layer neural network

Self-organizing maps: Introduction

Self-organization in brain cells

- Brain cells are self organizing themselves in groups according to incoming information.
- This incoming information is not only received by a single neural cell, but also influences other cells in its neighborhood.

Self-organizing maps (SOM) [T. Kohonen 1984] are a family of unsupervised neural networks based on this principle:

- It learns in an unsupervised way using prototypes that play the role of neurons.
- The prototypes follow a given topology so that neighborhood dependencies exist between them.
- Groups of neighbor prototypes specialize in recognizing similar groups of data.

Self-organizing maps: Architecture (1/2)

Self-organizing maps can have all sorts of 1D, 2D or 3D architecture that is convenient for visualization:

- 1D architecture: line of prototypes
- 2-Dimensional architectures (the most common): square or rectangular maps with any type of neighborhood (triangular neighborhood, square neighborhood, hexagonal neighborhood, or octagonal neighborhood)
- 3-Dimensional architectures (uncommon): Cubes, spheres.

Self-organizing maps: Architecture (2/2)

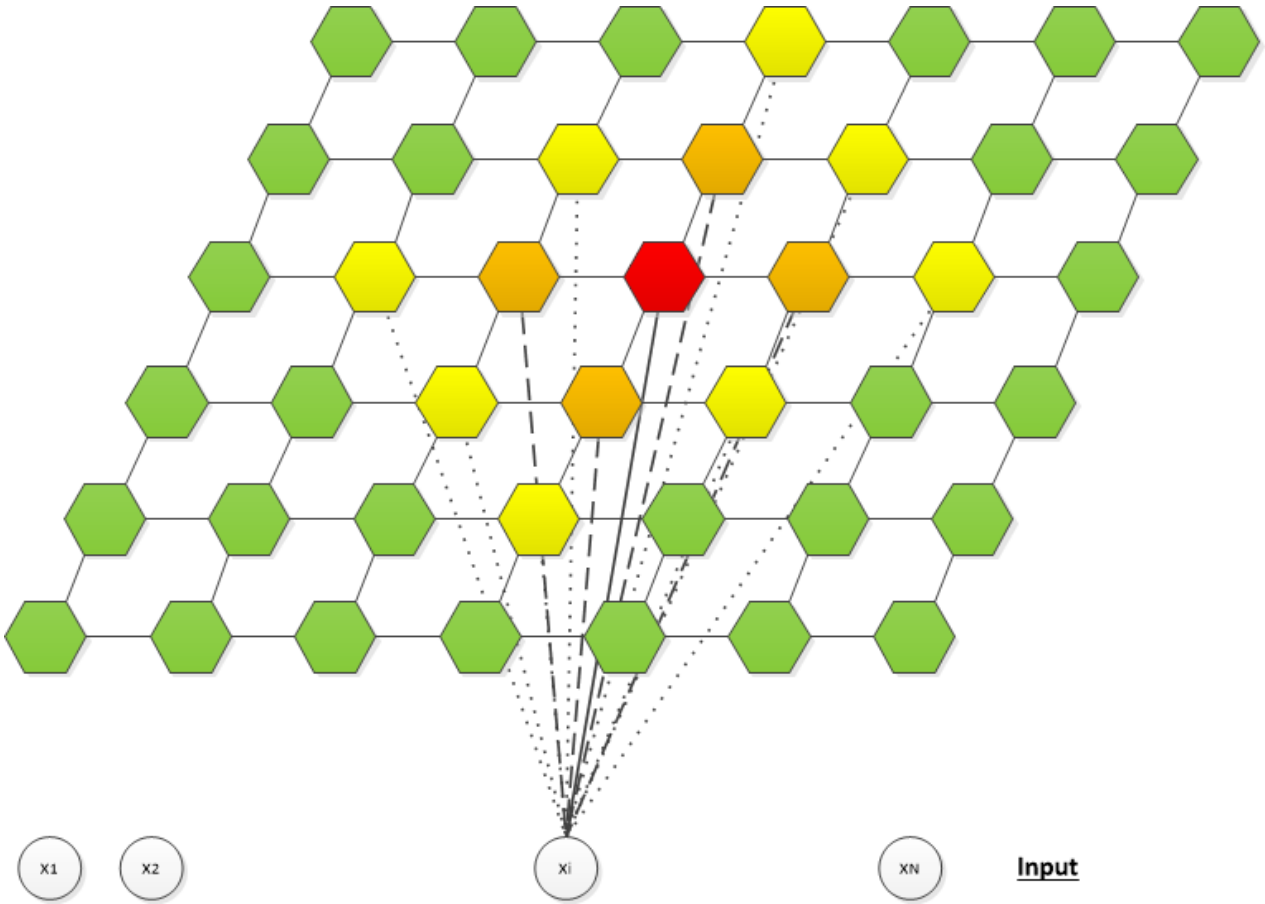


Figure: Example of a rectangle grid SOM architecture with 4 neighbors per neuron: Each example activates a winning neuron and also affects its neighbors.

Self-organizing maps: Cost function

For a data set $X = \{x_1, \cdots, x_N\}$ and a set of prototypes $W = \{w_1, \cdots, w_K\}$, the original SOM algorithm tries to minimize the following cost function:

$$C(W) = \sum_{i=1}^{N} \sum_{k=1}^{K} \mathcal{K}_{k,\chi(x_i)} ||x_i - w_k||^2$$

- $\chi(x_i)$ is the id of the best matching prototype w_j so that: $\chi(x_i) = \text{argmin}_j ||x_i - w_j||^2$.
- $0 \leq \mathcal{K}_{k,j} \leq 1$ is the neighborhood topological function (e.g. $\mathcal{K}_{k,j} = 0$ if w_j and w_k are far on the topological grid).

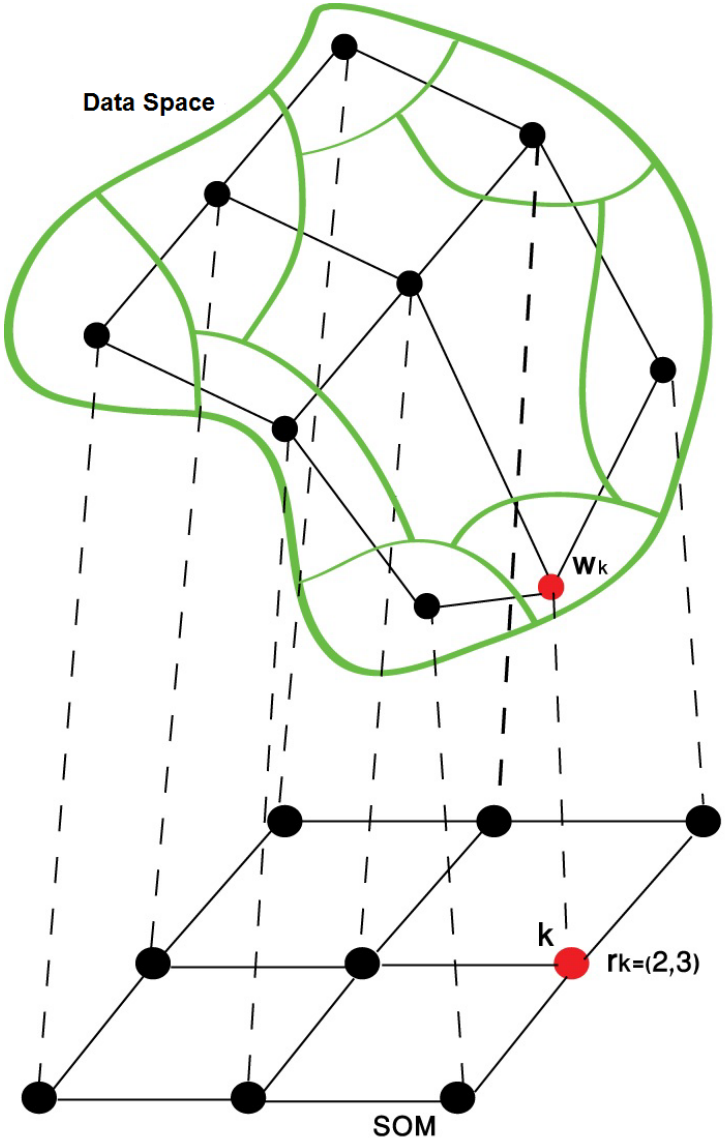
Self-organizing maps: Algorithm (1/2)

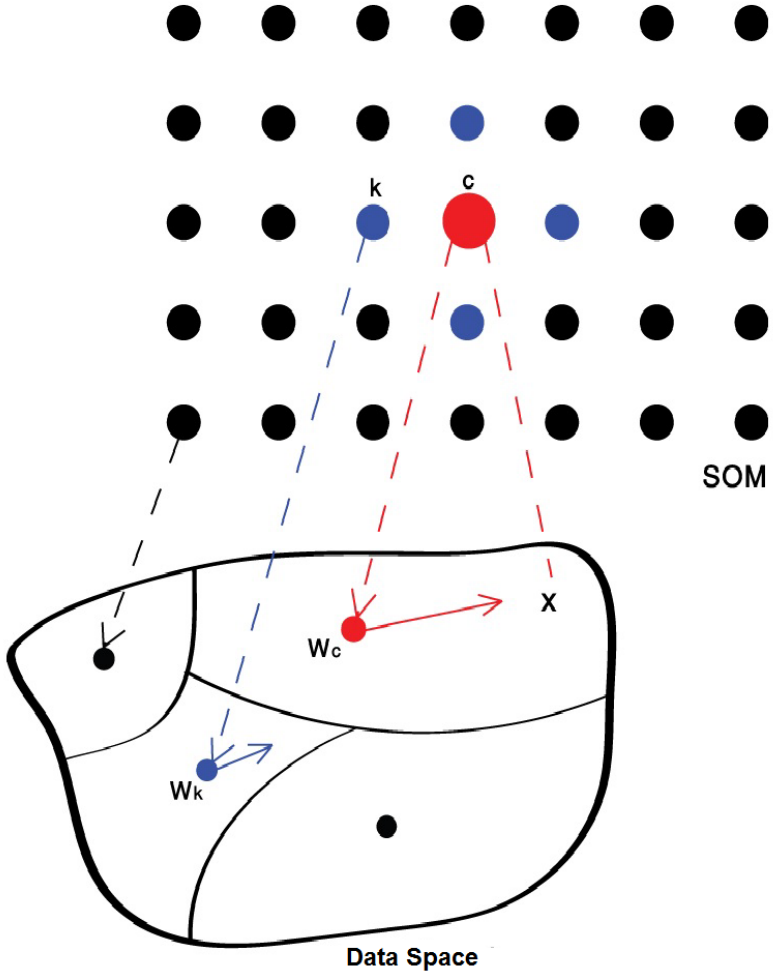
1. Initialization step
   - Define the topology of the map
   - Randomly initialize the prototypes for each neuron
   - Define a learning rate function $\epsilon(t)$
2. Competition step
   - Choose a data x_i randomly
   - Determine the winning neuron $\chi(x_i) = \text{argmin}_{1 \leq j \leq K} ||x_i - w_j||^2$
3. Adaptation step: Update the prototypes

$$w_k(t+1) = w_k(t) + \epsilon(t)\mathcal{K}_{k,\chi(x_i)}\left(x_i - w_k(t)\right)$$

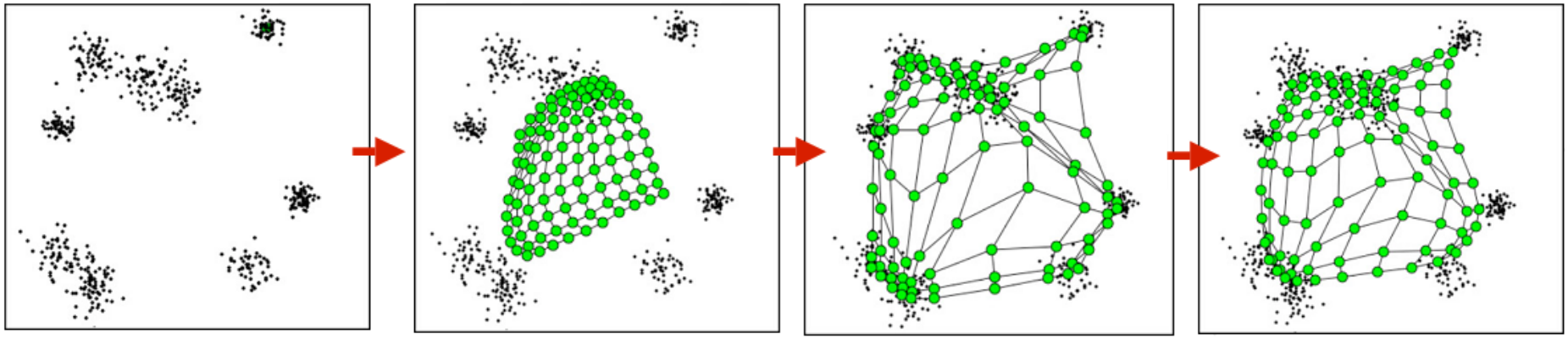
4. Repeat 2 and 3 until the prototypes updates are insignificant.

Self-organizing maps: Algorithm (2/2)





Self-organizing maps: Topological learning



Remark

- Isomap and LLE try to find a low dimensional manifolds in the data using neighborhood dependencies of the data.
- SOM has the opposite approach by trying to wrap a low-dimensional manifold around the data using prototypes neighborhood dependencies.

Self-organizing maps for visualization purposes

Once they are built, SOM can be used in several ways for visualization purposes:

- Cluster visualization on the 2D map using the best matching neuron
- Topological visualization of the data set using colors to project distance matrices between the prototypes on the map.
- Component plane analysis by projecting the components on the map.
- Data projection and transformation

Self-organizing maps: Visualization Example 1

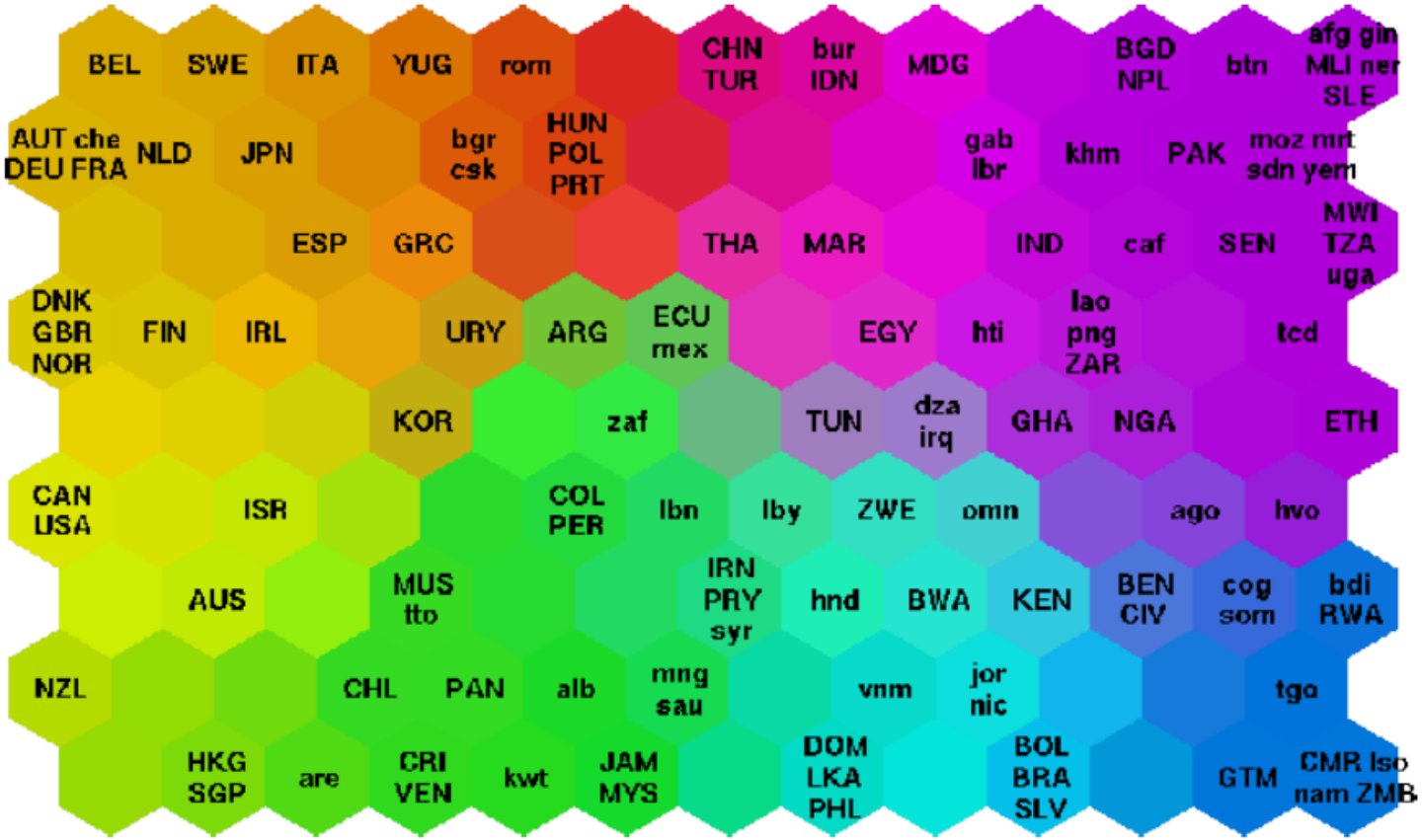


Figure: Visualizing world countries clusters projected on a topological mal based on their characteristics from the Worldbank data.

Self-organizing maps: Visualization Example 2 (1/2)

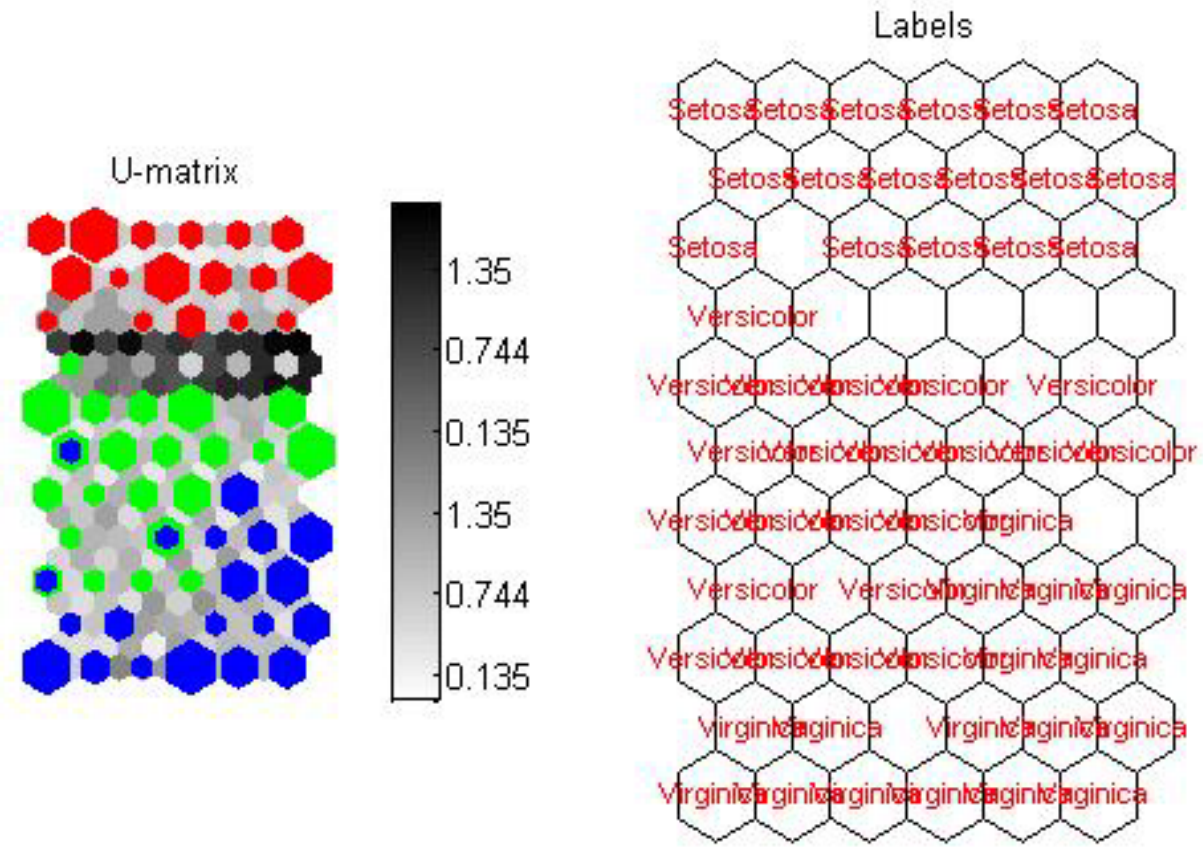


Figure: U-Matrix (distance) and class repartition for the Iris data set using SOM

Self-organizing maps: Visualization Example 2 (2/2)

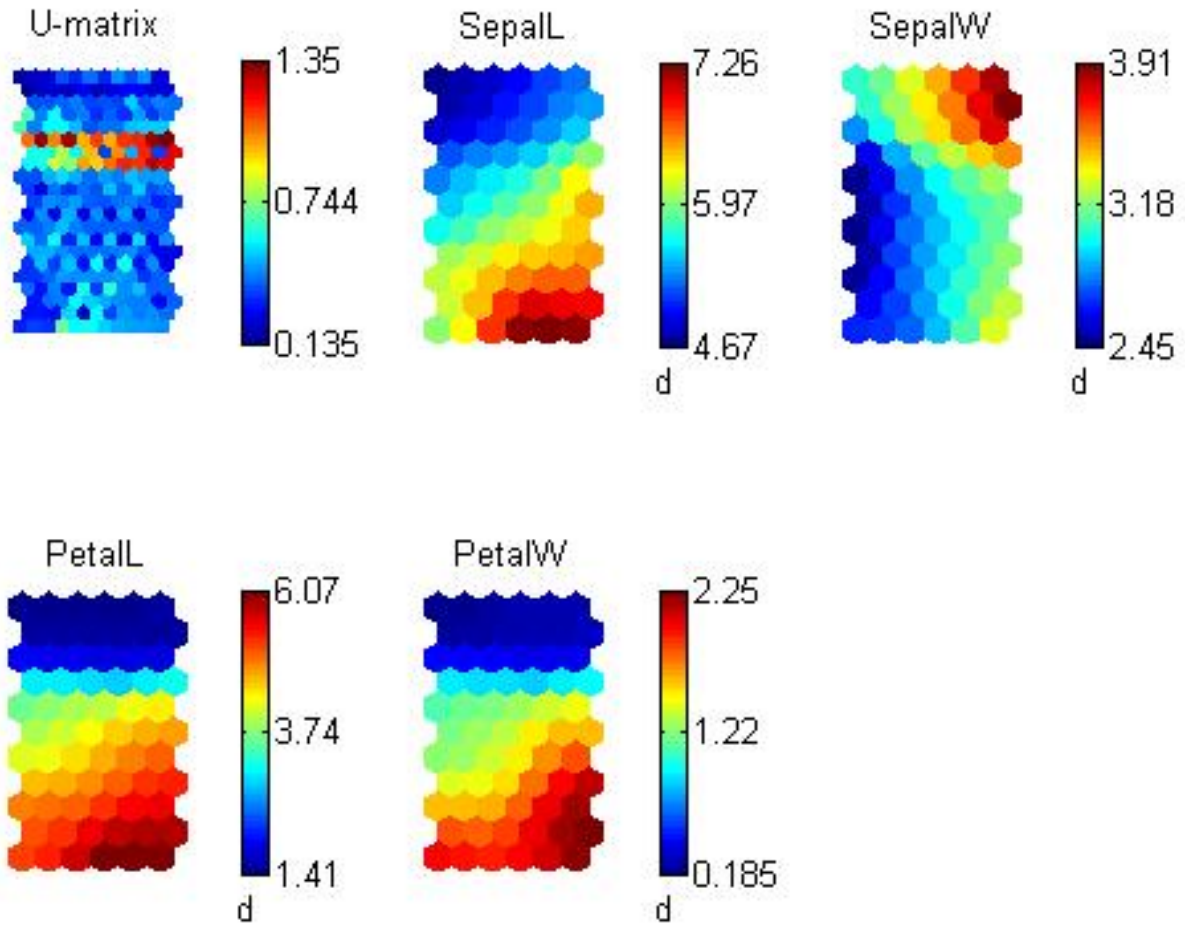
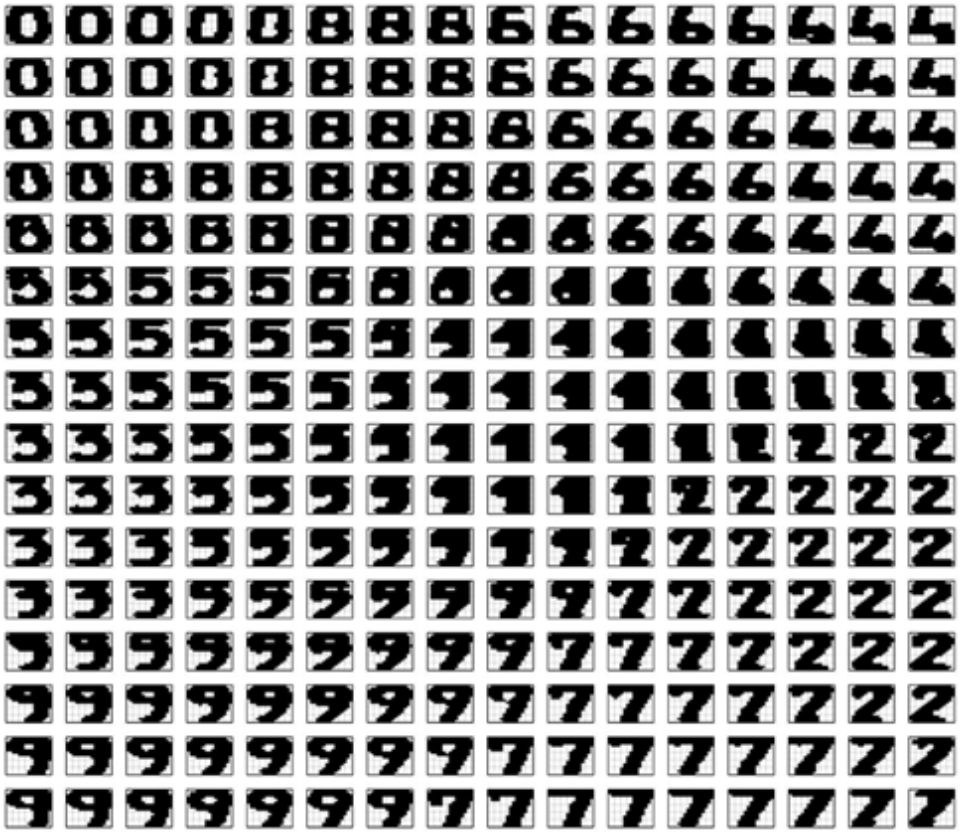
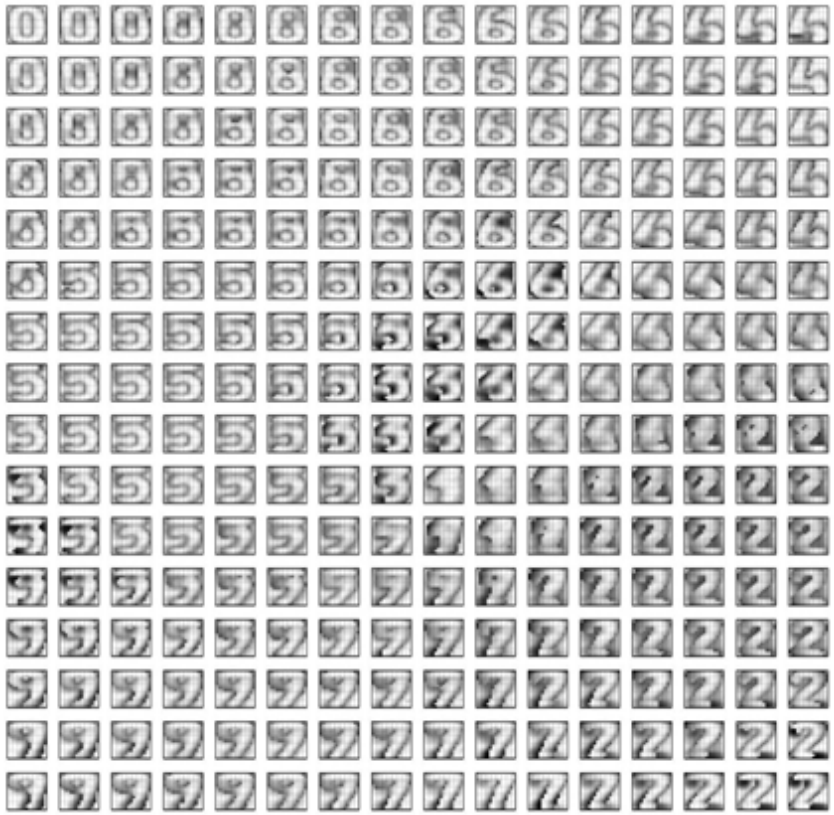


Figure: Feature projection for the Iris data set using SOM

Self-organizing maps: Visualization Example 3



(a) Original SOM



(b) Contrast only

Figure: Number recognition using SOM [Rogovschi et al. 2008]

Self-organizing maps: SOM vs PCA

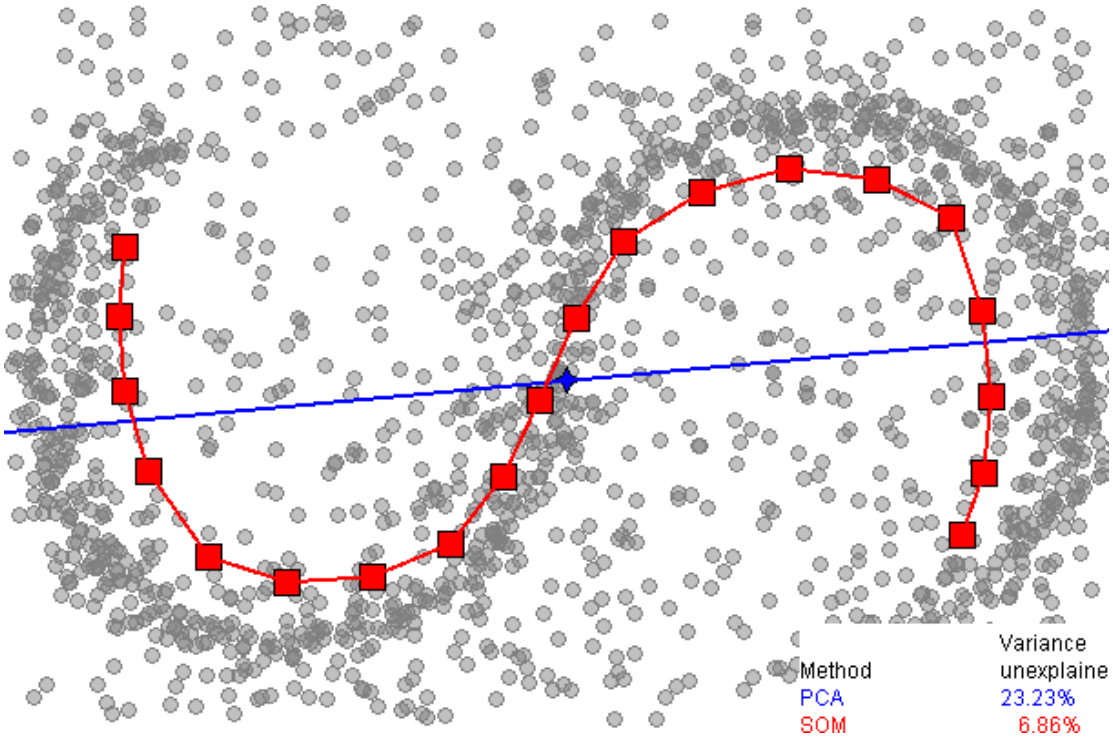


Figure: Projecting the data along a linear SOM network can work better than a PCA on non-linear data

Self-organizing maps: Conclusions

Advantages of SOM

- Is useful for all kinds of visualizations and does a pre-clustering
- Can handle non-linear data
- Is fast $O(KN)$
- The initial topology of the map does not matter much: If the map is big enough, it will work.

Limitations of SOM

- The learning rate is a critical parameter.
- There are plenty of variations of the algorithm each of them with different strengths and weaknesses.

Outline

Bibliography

Books:

- Christopher M. Bishop, Pattern Recognition and Machine Learning (2006)
- Tom M. Mitchell, Machine Learning (1997)

Articles:

- J. B. Tenenbaum, V. De Silva, and J. C. Langford: *A global geometric framework for nonlinear dimensionality reduction*, Science, 290:2319-2323, 2000.
- Sam Roweis and Lawrence Saul: *Nonlinear dimensionality reduction by locally linear embedding*, Science, 290:2323-2326, 2000
- Teuvo Kohonen: *Self-Organizing Maps*, vol. 30, Springer Verlag, 1995.