

# Lecture 15. Dimensionality Reduction

COMP90051 Statistical Machine Learning

Semester 2, 2017  
Lecturer: Andrey Kan



THE UNIVERSITY OF  
MELBOURNE

# This lecture

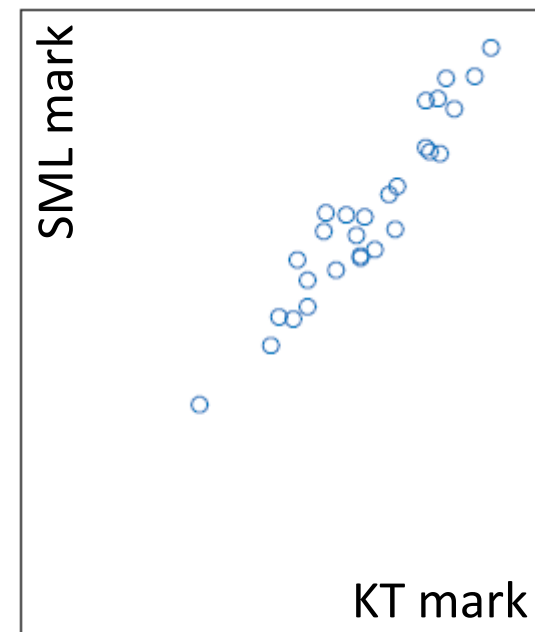
- Principal components analysis
  - \* Linear dimensionality reduction method
  - \* Diagonalising covariance matrix
- Multidimensional scaling
  - \* Non-linear dimensionality reduction methods
  - \* Explicit search for a low-dimensional configuration

# Dimensionality reduction

- Moving on from clustering to the next unsupervised learning topic
- Dimensionality reduction refers to representing the data using a smaller number of variables (dimensions) while preserving the “interesting” structure of the data
- Such a reduction can serve several purposes
  - \* Visualisation (e.g., by mapping multidimensional data on 2D)
  - \* Computational efficiency
  - \* Data compression

# Exploiting data structure

- Dimensionality reduction in general results in loss of information
- The trick is to ensure that most of the “interesting” information (signal) is preserved, while what is lost is mostly noise
- This is often possible because real data may have inherently fewer dimensions than recorded variables
- Example 1: GPS coordinates are 3D, while car locations on a flat road are actually 2D
- Example 2: Marks\* for Knowledge Technology and Statistical Machine Learning



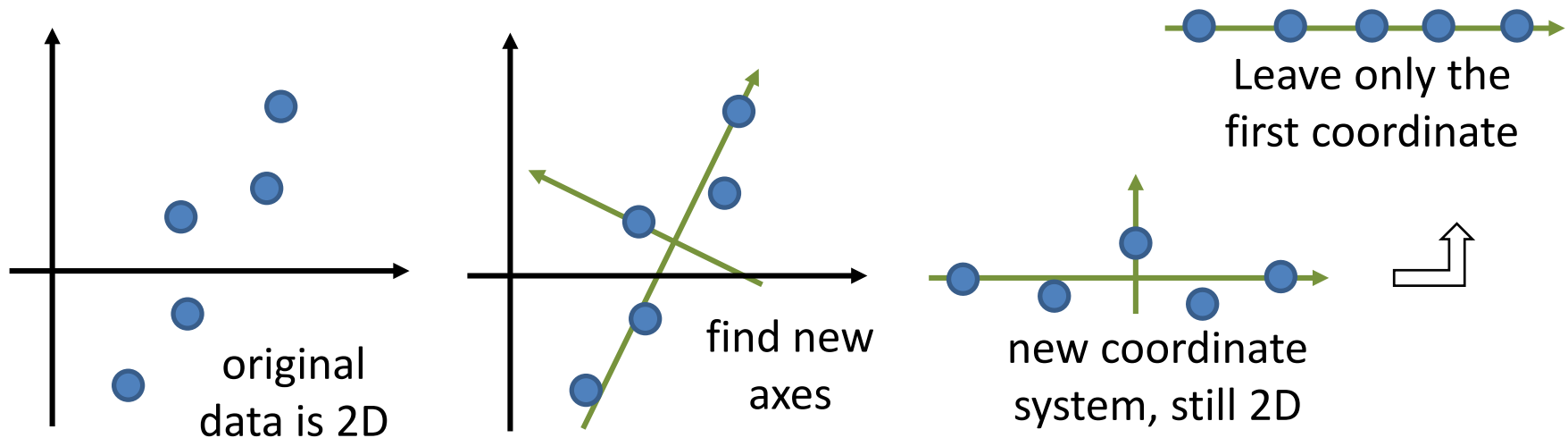
\* synthetic data :)

# Principal Component Analysis

Finding a rotation of data  
that minimises covariance  
between variables

# Principal components analysis

- Principal components analysis (PCA) is a popular method for dimensionality reduction and data analysis in general
- Given a dataset  $\mathbf{x}_1, \dots, \mathbf{x}_n, \mathbf{x}_i \in \mathbf{R}^m$ , PCA aims to find a new coordinate system such that most of the variance is concentrated along the first coordinate, then most of the remaining variance along the second coordinate, etc.
- Dimensionality reduction is based on discarding all coordinates except the first  $l < m$



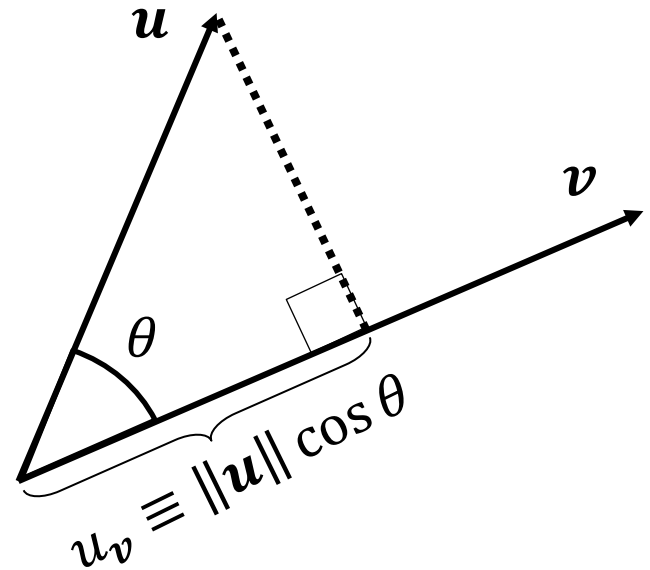
# Naïve PCA algorithm

- In principle, PCA operation can be described as follows
  1. Choose an axis, such that the variance along this axis is maximised
  2. Choose the next axis perpendicular to all axes so far, such that the (remaining) variance along this axis is maximised
  3. Repeat 2, until you have the same number of axes (i.e., dimensions) as in the original data
  4. Project original data on the axes. This gives new coordinates (“PCA coordinates”)
  5. For each point, keep only the first  $l$  coordinates

*Such an algorithm if implemented directly would work, but there's a better solution*

# Formalising the problem

- The main part of PCA is finding the new coordinate system, such that most of the variation is captured by “earlier” axes
- Let’s write down this aim formally and see how it can be achieved
- First, recall the geometric definition of a dot product  $\mathbf{u} \cdot \mathbf{v} = u_v \|\mathbf{v}\|$
- Suppose  $\|\mathbf{v}\| = 1$ , so  $\mathbf{u} \cdot \mathbf{v} = u_v$
- Vector  $\mathbf{v}$  can be considered an coordinate axis, and  $u_v$  a coordinate of point  $\mathbf{u}$





# Data transformation

- So the “new coordinate system” is a set of vectors  $\mathbf{p}_1, \dots, \mathbf{p}_m$ , where each  $\|\mathbf{p}_i\| = 1$
- Consider an original data point  $\mathbf{x}_j, j = 1, \dots, n$ , and a principal axis  $\mathbf{p}_i, i = 1, \dots, m$
- The corresponding  $i^{\text{th}}$  coordinate for the first point after the transformation is  $(\mathbf{p}_i)'(\mathbf{x}_1)$ 
  - \* For the second point it is  $(\mathbf{p}_i)'(\mathbf{x}_2)$ , etc.
- Collate all these numbers into a vector  $[(\mathbf{p}_i)'(\mathbf{x}_1), \dots, (\mathbf{p}_i)'(\mathbf{x}_n)]' = ((\mathbf{p}_i)'X)' = X'\mathbf{p}_i$ , where  $X$  has original data points in columns

# Refresher on basic statistics

- Consider a random variable  $U$  and the corresponding sample  $\mathbf{u} = [u_1, \dots, u_n]'$
- Everyone knows how to compute sample mean  $\bar{u} \equiv \frac{1}{n} \sum_{i=1}^n u_i$ . Most people will also remember sample variance  $\frac{1}{n-1} \sum_{i=1}^n (u_i - \bar{u})^2$
- Suppose the mean was subtracted beforehand (the sample is *centered*). In this case, the variance is a scaled dot product  $\frac{1}{n-1} \mathbf{u}'\mathbf{u}$
- Similarly, if we have a centered random sample  $\mathbf{v}$  from another random variable, sample covariance is  $\frac{1}{n-1} \mathbf{u}'\mathbf{v}$
- Finally, if our data is  $\mathbf{x}_1 = [u_1, v_1]'$ , ...,  $\mathbf{x}_n = [u_n, v_n]'$  organised into a matrix  $\mathbf{X}$  with data in columns and centered variables in rows, we have that covariance matrix is  $\boldsymbol{\Sigma}_X \equiv \frac{1}{n-1} \mathbf{X}\mathbf{X}'$ 
  - \* In this example, data is 2D, but the same hold for any number of dimensions

# The objective of PCA

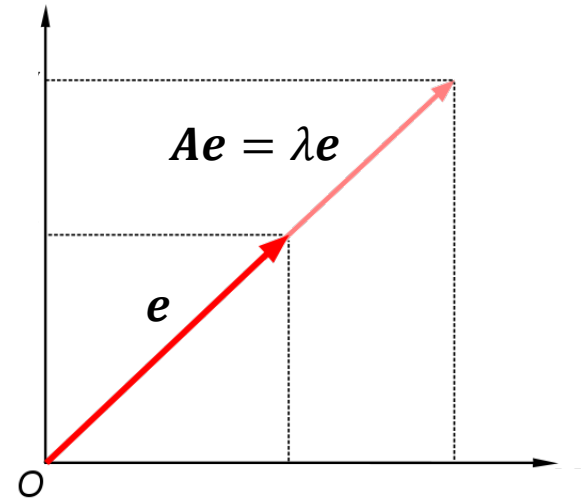
- From now we shall assume that the data is centered
- Let's start with the objective for the first principal axis only. The data projected on this axis is described by  $X'p_1$
- Accordingly, the variance along this principal axis is
$$\frac{1}{n-1} (X'p_1)'(X'p_1) = \frac{1}{n-1} p_1'XX'p_1 = p_1'\Sigma_X p_1$$
  - \* Here  $\Sigma_X$  is the covariance matrix of the original data
- PCA aims to find  $p_1$  that maximises  $p_1'\Sigma_X p_1$ , subject to  $\|p_1\| = 1$

# Solving the optimisation problem

- PCA aims to find  $\mathbf{p}_1$  that maximises  $\mathbf{p}'_1 \boldsymbol{\Sigma}_X \mathbf{p}_1$ , subject to  $\|\mathbf{p}_1\| = \mathbf{p}'_1 \mathbf{p}_1 = 1$
- Recall our old friend Lagrange. Introduce a Lagrange multiplier  $\lambda_1$ , and set derivatives of the Lagrangian to zero
- $L = \mathbf{p}'_1 \boldsymbol{\Sigma}_X \mathbf{p}_1 - \lambda_1 (\mathbf{p}'_1 \mathbf{p}_1 - 1)$
- $\frac{\partial L}{\partial \mathbf{p}_1} = 2\boldsymbol{\Sigma}_X \mathbf{p}_1 - 2\lambda_1 \mathbf{p}_1 = 0$
- $\boldsymbol{\Sigma}_X \mathbf{p}_1 = \lambda_1 \mathbf{p}_1$
- The latter is precisely the definition of an eigenvector with  $\lambda_1$  being the corresponding eigenvalue

# Refresher on eigenvectors (1/2)

- Given a square matrix  $\mathbf{A}$ , a column vector  $\mathbf{e}$  is called an eigenvector if  $\mathbf{A}\mathbf{e} = \lambda\mathbf{e}$ . Here  $\lambda$  is the corresponding eigenvalue



- Geometric interpretation: compare  $\mathbf{A}\mathbf{e}$  with  $\mathbf{P}\mathbf{x}_i$  from previous slides. Here  $\mathbf{A}$  is a transformation matrix (“new axes”) for some vector  $\mathbf{e}$ . Vector  $\mathbf{e}$  is such that it still points to the same direction after transformation

# Refresher on eigenvalues (2/2)

- Algebraic interpretation: if  $\mathbf{A}\mathbf{e} = \lambda\mathbf{e}$  then  $(\mathbf{A} - \lambda\mathbf{I})\mathbf{e} = 0$ , where  $\mathbf{I}$  is the identity matrix
- This equation has a non-zero solution  $\mathbf{e}$  if and only if the determinant is zero  $|\mathbf{A} - \lambda\mathbf{I}| = 0$ . Eigenvalues are roots of this equation called characteristic equation
- Eigenvectors and eigenvalues are prominent concepts in linear algebra and arise in many practical applications
- Spectrum of a matrix is a set of its eigenvalues
  - \* Hence name “spectral clustering” in a next lecture
- There are efficient algorithms for computing eigenvectors

# Finding the first PCA axis

- We conclude that in order to maximise variance along the first principal axis, the axis should be chosen such that  $\Sigma_X \mathbf{p}_1 = \lambda_1 \mathbf{p}_1$
- In other words,  $\mathbf{p}_1$  has to be an eigenvector of centered data covariance matrix  $\Sigma_X$
- Note that  $\lambda_1 = \mathbf{p}_1' \Sigma_X \mathbf{p}_1$ , and recall that  $\mathbf{p}_1' \Sigma_X \mathbf{p}_1$  is the variance of the projected data
- Thus we need to choose  $\mathbf{p}_1$  that corresponds to the largest eigenvalue of centered data covariance matrix  $\Sigma_X$

# Efficient solution for PCA

- This type of reasoning can be continued to find subsequent axes  $\mathbf{p}_2$ ,  $\mathbf{p}_3$ , etc.
- Note that constraint  $\|\mathbf{p}_i\| = 1$  is important because otherwise variance  $\mathbf{p}_i' \boldsymbol{\Sigma}_X \mathbf{p}_i$  can be arbitrary increased by rescaling  $\mathbf{p}_i$
- Each time we add additional constraints that the next axis is orthogonal to all previous
- It turns out that the final solution is to set  $\mathbf{p}_i$  as eigenvectors of centered data covariance matrix  $\boldsymbol{\Sigma}_X$  in the order of decreasing corresponding eigenvalues
- But is this possible to do with any  $\boldsymbol{\Sigma}_X$ ?
- Lemma: a real symmetric  $m \times m$  matrix has  $m$  real eigenvalues and the corresponding eigenvectors are orthogonal



# Interim summary on PCA (1/2)

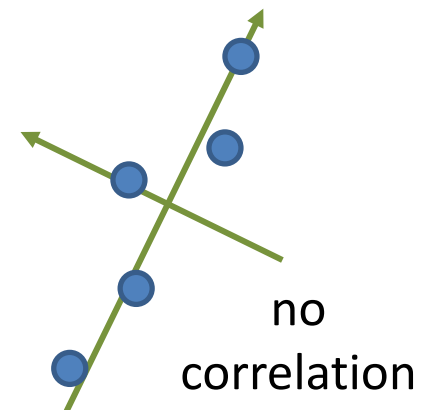
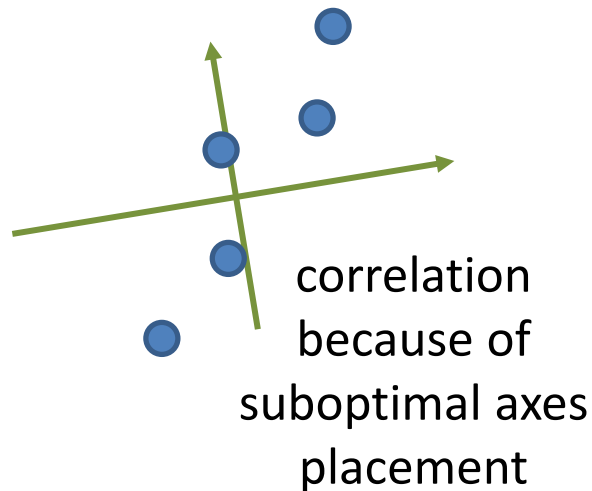
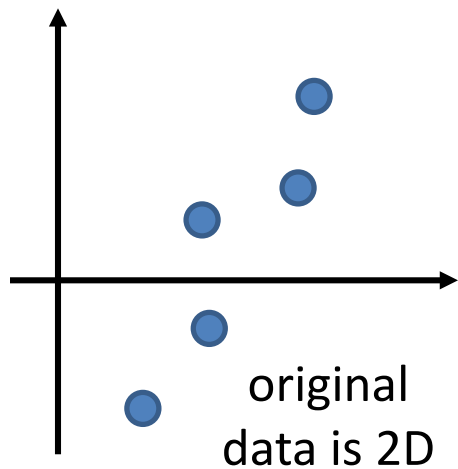
- Assume data points are arranged in columns of  $\mathbf{X}$ . That means that the *variables* are in rows
- Ensure that the data is centered: subtract the mean of each row from each row
- We seek for an *orthonormal* basis  $\mathbf{p}_1, \dots, \mathbf{p}_m$ 
  - \* That is, each axis vector is of unit length and perpendicular to every other axis
- In order to find such a basis, find eigenvalues of centered data covariance matrix  $\boldsymbol{\Sigma}_X \equiv \frac{1}{n-1} \mathbf{X}\mathbf{X}'$ 
  - \* This is always possible, and there are efficient ways of doing this

# Interim summary on PCA (2/2)

- Sort eigenvalues from largest to smallest
  - \* Each eigenvalue equals to variance along the corresponding PCA axis
- Set  $\mathbf{p}_1, \dots, \mathbf{p}_m$  as corresponding eigenvectors
- Project data  $\mathbf{X}$  onto these new axes to get coordinates of the transformed data
- Keep only the first  $s$  coordinates to reduce dimensionality

# Additional effect of PCA

- PCA aims to find axes such that the variance along each subsequent axis is maximised
- Consider axes  $i$  and  $(i + 1)$ . Informally, if there's a correlation between them, this means that axis  $i$  can be rotated further to capture more variance
- PCA should end up finding new axes (i.e., the transformation) such that the transformed data is uncorrelated



# Spectral theorem for symmetric matrices

- In order to explore this effect further, we need to refer to one of the fundamental results in linear algebra
  - \* The proof is outside the scope of this subject
  - \* This is a special case of singular value decomposition theorem
- Theorem: for any a real symmetric matrix  $\Sigma_X$  there exists a real orthogonal matrix  $P$  with eigenvectors of  $\Sigma_X$  arranged in rows and a diagonal matrix of eigenvalues  $\Lambda$  such that  $\Sigma_X = P' \Lambda P$

# Diagonalising covariance matrix (1/2)

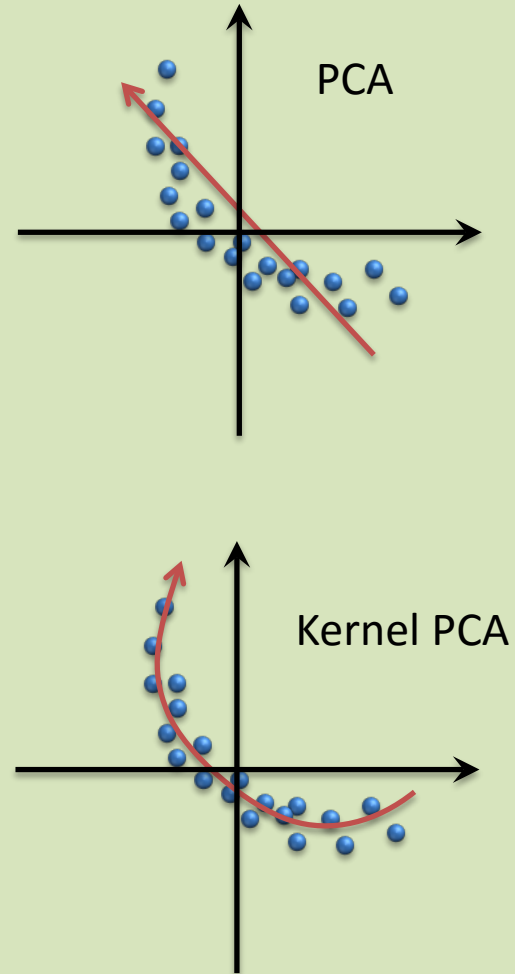
- Form a transformation matrix  $\mathbf{P}$  with eigenvectors (the new axes) as rows
  - \* By our problem formulation,  $\mathbf{P}$  is an orthonormal matrix
- Note that  $\mathbf{P}'\mathbf{P} = \mathbf{I}$ , where  $\mathbf{I}$  is the identity matrix
  - \* To see this recall that each element of the resulting matrix multiplication is a dot product of the corresponding row and column
  - \* So element  $(i, j)$  of  $\mathbf{P}'\mathbf{P}$  is the dot product  $\mathbf{p}'_i\mathbf{p}_j$ , which is 1 if  $i = j$ , and 0 otherwise
- The transformed data is  $\mathbf{P}\mathbf{X}$ 
  - \* Similar to above, note that element  $(i, j)$  of  $\mathbf{P}\mathbf{X}$  is the dot product  $\mathbf{p}'_i\mathbf{x}_j$ , which is the projection of  $\mathbf{x}_j$  on axis  $\mathbf{p}_i$ , i.e., the new  $i^{\text{th}}$  coordinate for  $j^{\text{th}}$  point

# Diagonalising covariance matrix (2/2)

- The covariance of the *transformed data* is
- $\Sigma_{PX} \equiv \frac{1}{n-1} (PX)(PX)' = \frac{1}{n-1} (PX)(X'P') = P\Sigma_X P'$
- By spectral decomposition theorem we have  $\Sigma_X = P' \Lambda P$
- Therefore  $\Sigma_{PX} = P P' \Lambda P P' = \Lambda$
- The covariance matrix of the transformed data is diagonal with eigenvalues on the diagonal of  $\Lambda$
- The transformed data is uncorrelated

# Non-linear data and kernel PCA

- Low dimensional approximation need not be linear
- Kernel PCA: map data to feature space, then run PCA
  - \* Express principal components in terms of data points. Solution uses  $\mathbf{X}'\mathbf{X}$  that can be kernelised  $(\mathbf{X}'\mathbf{X})_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$
  - \* The solution strategy differs from regular PCA
  - \* Changing the kernel leads to a different feature space transformation



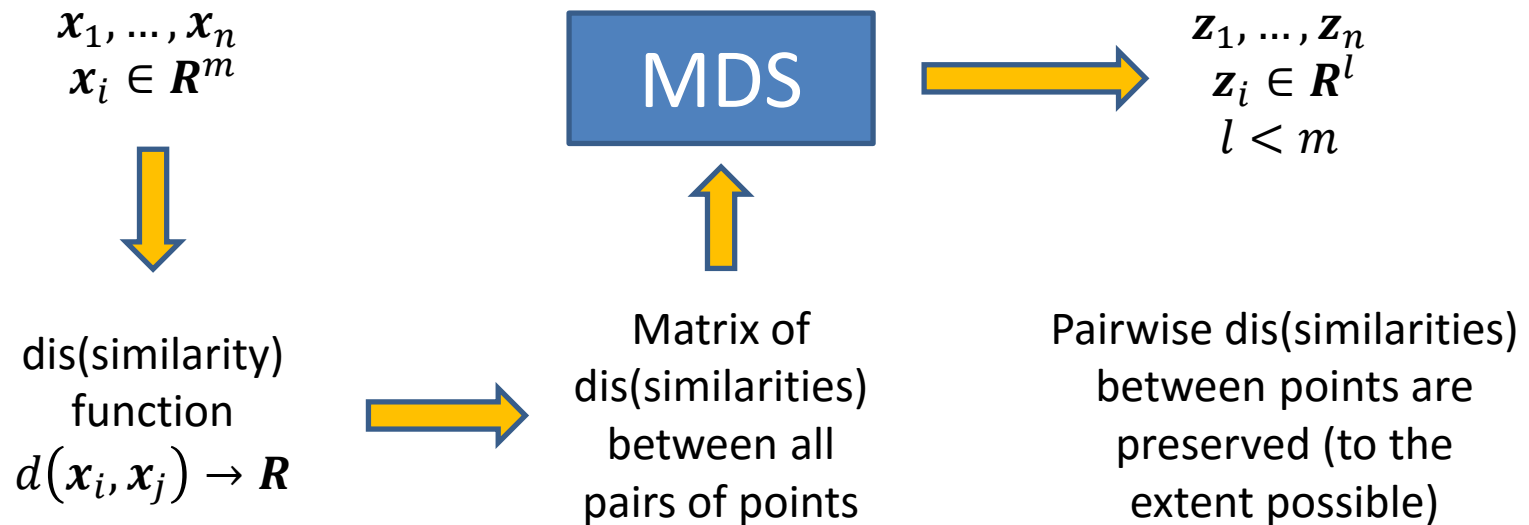
# Multidimensional Scaling

A brief overview of a family  
of scaling methods



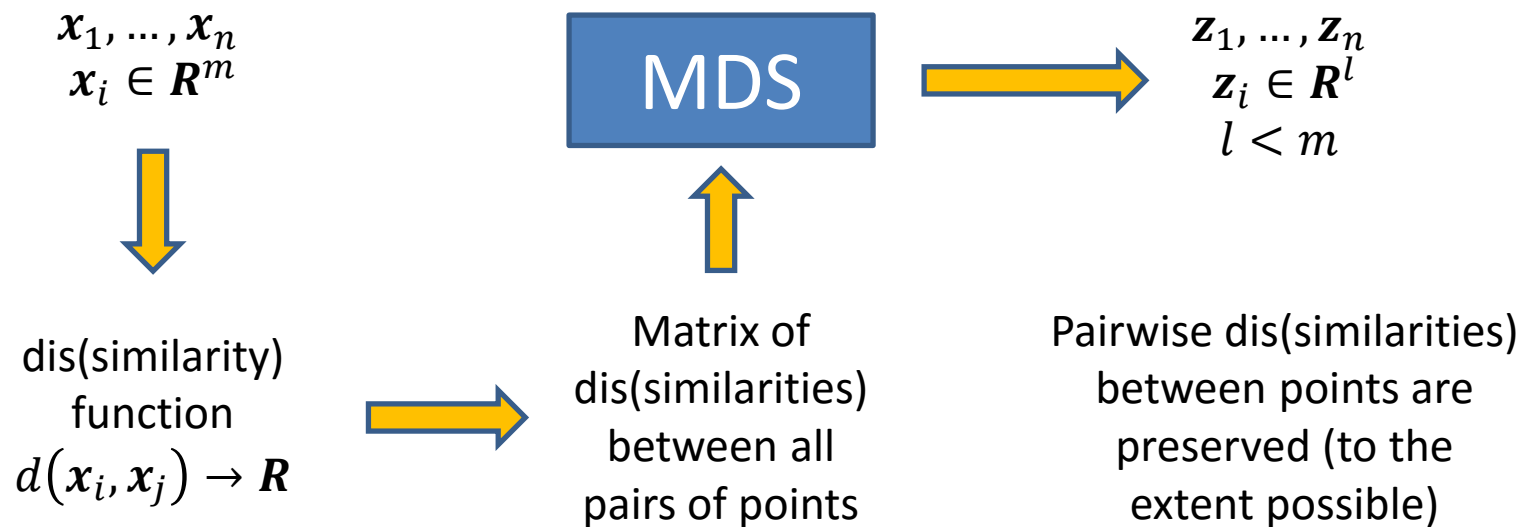
# Multidimensional scaling

- Another common approach to address non-linear data is multidimensional scaling (MDS)
- MDS is a common name for a group of related methods
- MDS aims to map data to a lower-dimensional space, such that pairwise dis(similarities) are preserved



# Types of MDS

- There are two “parameters” of the MDS approach
  - \* How to measure dis(similarity)
  - \* How to measure preservation of dis(similarity)
- Different types of MDS differ in these “parameters”



# MDS as an optimization problem

- One natural choice is to measure dissimilarity between the mapped points using Euclidean distance  $d(\mathbf{z}_i, \mathbf{z}_j) = \|\mathbf{z}_i - \mathbf{z}_j\|$

- The preservation can be measured using a function such as

$$S(\mathbf{z}_1, \dots, \mathbf{z}_n) = \frac{\sum_{i,j} \left( d(\mathbf{x}_i, \mathbf{x}_j) - d(\mathbf{z}_i, \mathbf{z}_j) \right)^2}{\sum_{i,j} d(\mathbf{z}_i, \mathbf{z}_j)^2}$$

\* In MDS, such a function is called the stress function

- The aim of such MDS is to

find  $\mathbf{z}_1, \dots, \mathbf{z}_n$  that minimise  $S_M(\mathbf{z}_1, \dots, \mathbf{z}_n)$

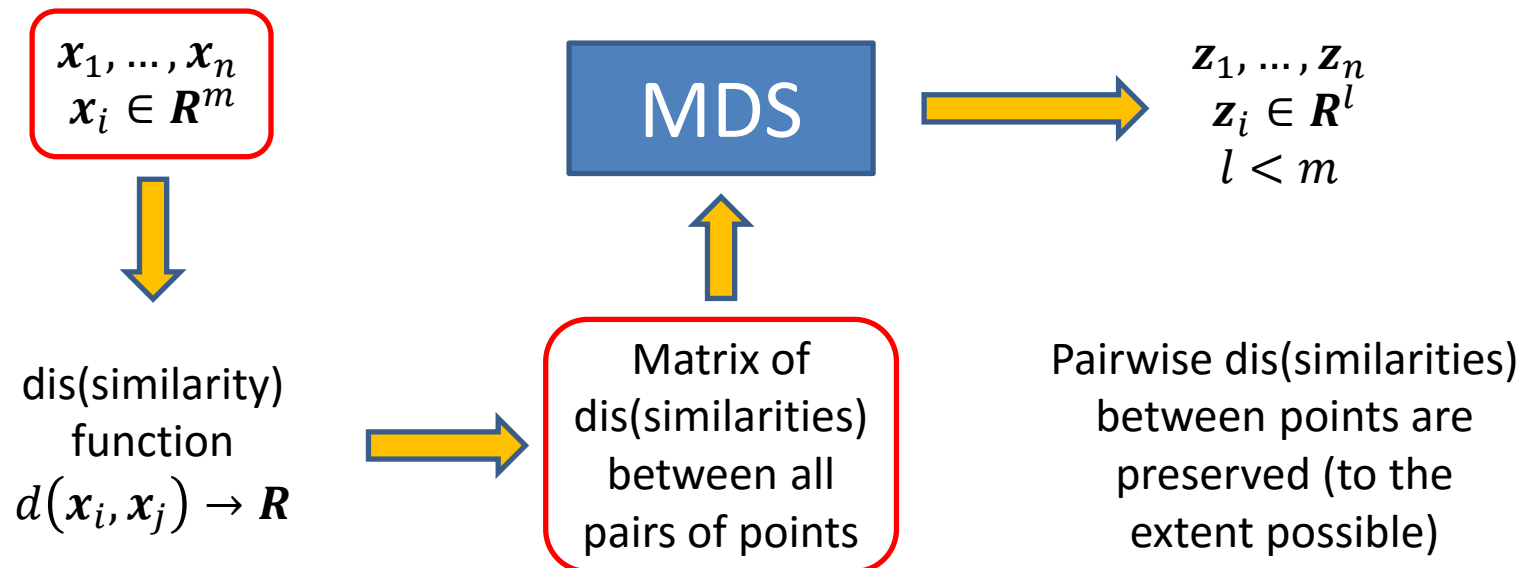
- This can be solved using gradient descent

# Intuition behind MDS

- Suppose that there are genuine clusters in high dimensional data
- Points within clusters are close to each other, points from different clusters are far away
- MDS attempts to preserve this distance structure, so that clusters are (hopefully) preserved in the low dimensional map

# MDS and data representation

- Note interplay between different ways to represent (almost) the same information
- This workflow can be tweaked to fit different applications

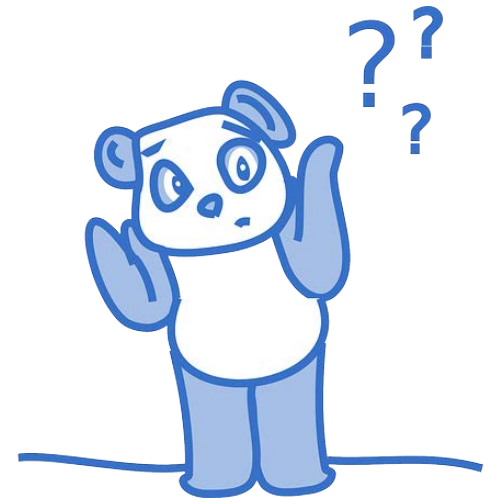


# Exercise: Mapping cities in Uzbekistan

- Reconstruct locations of cities based on distances

	Tashkent	Samarkand	Khiva
Tashkent		270	740
Samarkand	270		600
Khiva	740	600	

art: OpenClipartVectors at  
pixabay.com (CC0)



Hint: Samarkand is the south most of these three. Tashkent is the east most.

# Data representation

- Switching between data representations

Compute distances



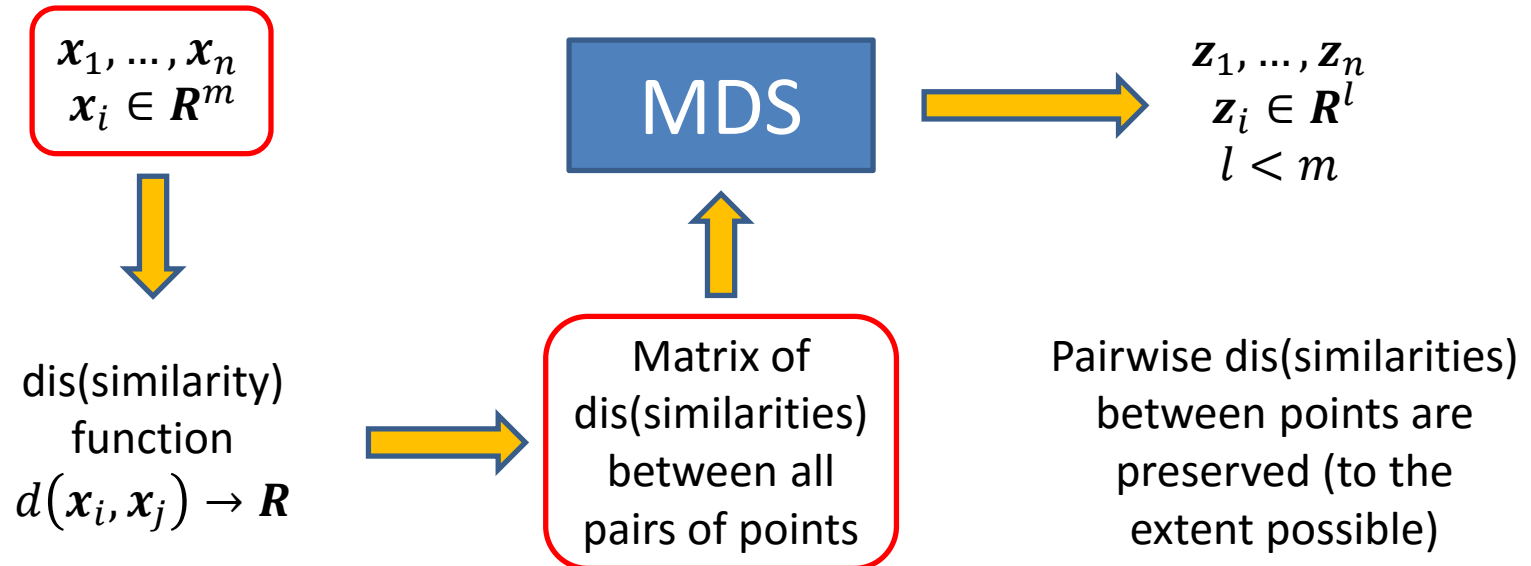
- \* Coordinates for each point

- \* Matrix of pairwise distances

MDS



- \* (Reconstructed) coordinates for each point



# MDS for recovering locations

- Switching between data representations

- \* Unknown coordinates for each point

- \* Matrix of pairwise distances

- \* Reconstructed coordinates for each point

No dimensionality reduction

MDS



$x_1, \dots, x_n$   
 $x_i \in \mathbb{R}^m$



dis(similarity)  
function  
 $d(x_i, x_j) \rightarrow \mathbb{R}$



Matrix of  
dis(similarities)  
between all  
pairs of points



MDS



$z_1, \dots, z_n$   
 $z_i \in \mathbb{R}^m$

Pairwise dis(similarities)  
between points are  
preserved (to the  
extent possible)



# MDS for dimensionality reduction

- Switching between data representations

Compute  
distances



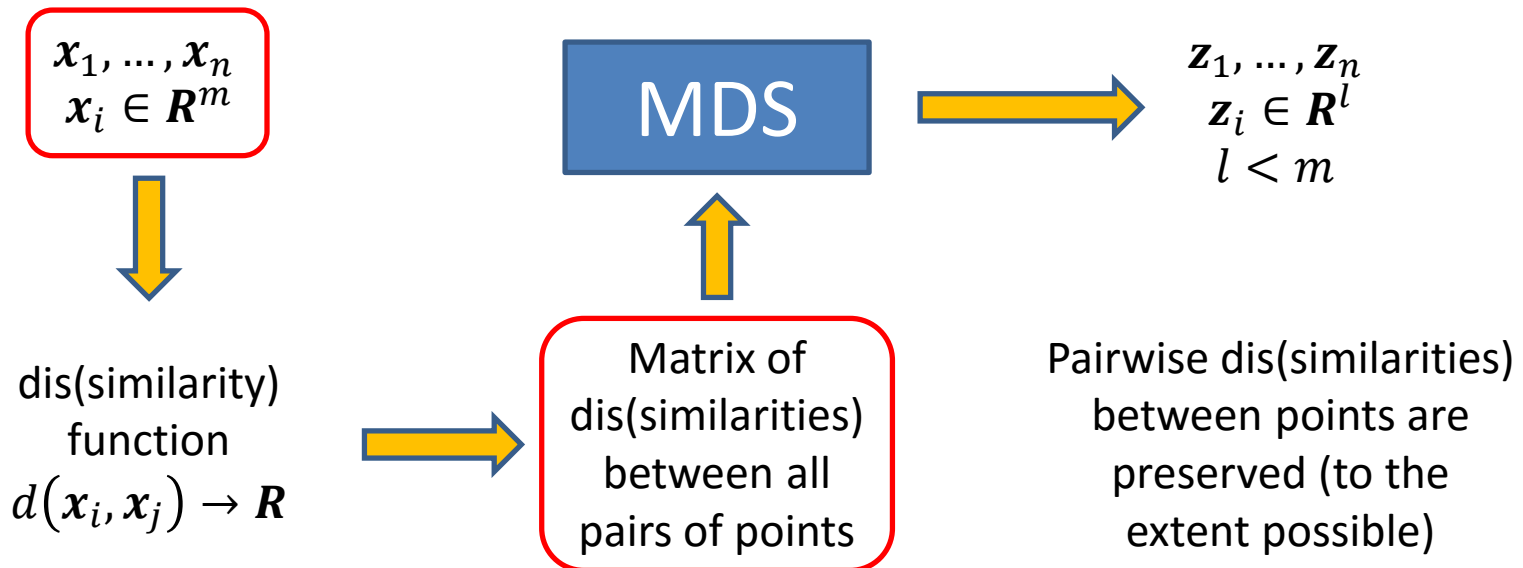
- \* High dimensional coordinates

MDS



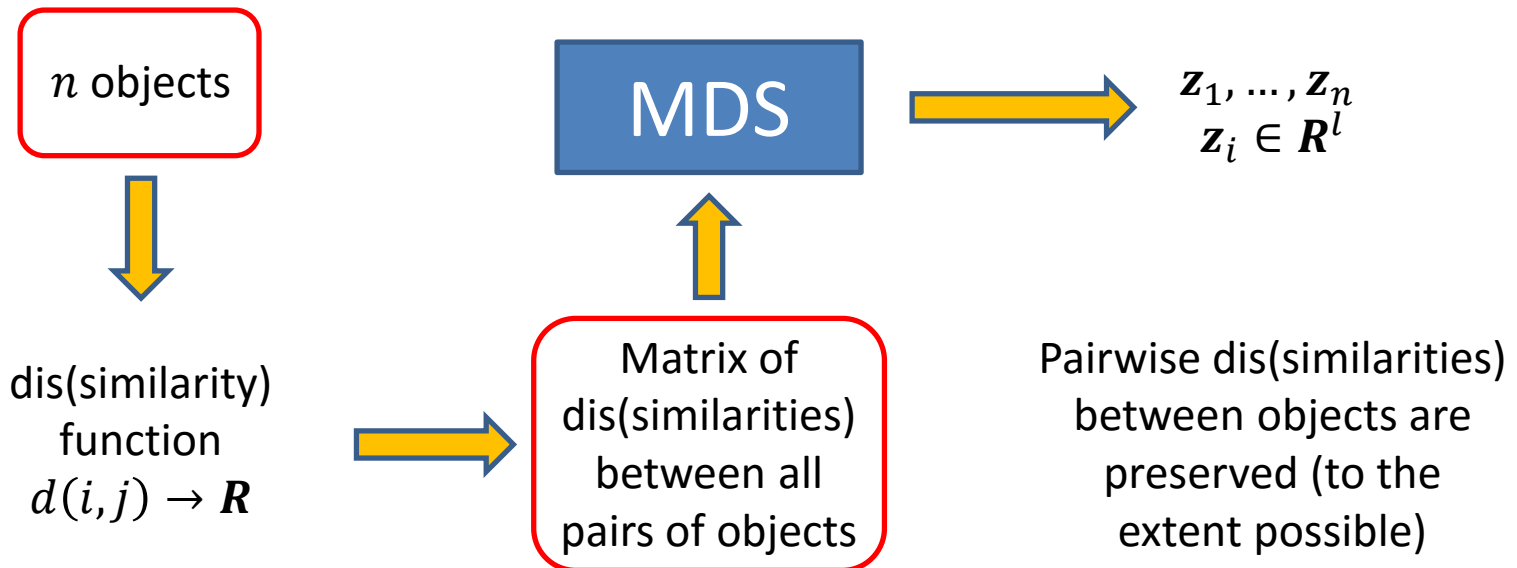
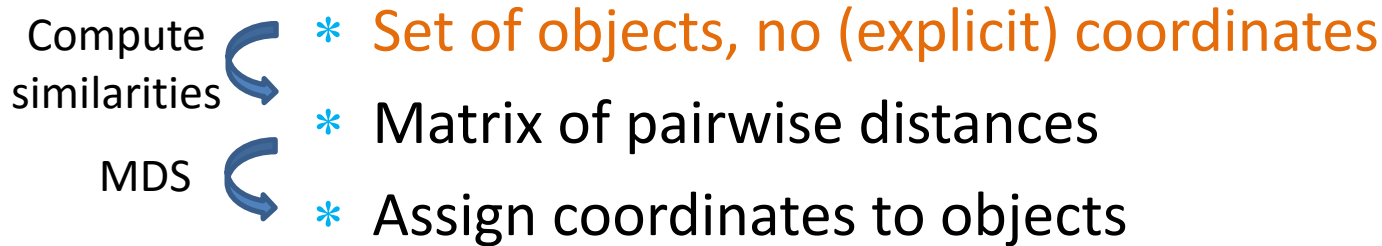
- \* Matrix of pairwise distances

- \* Approximate low dimensional coordinates



# MDS for finding a meaningful map

- Switching between data representations




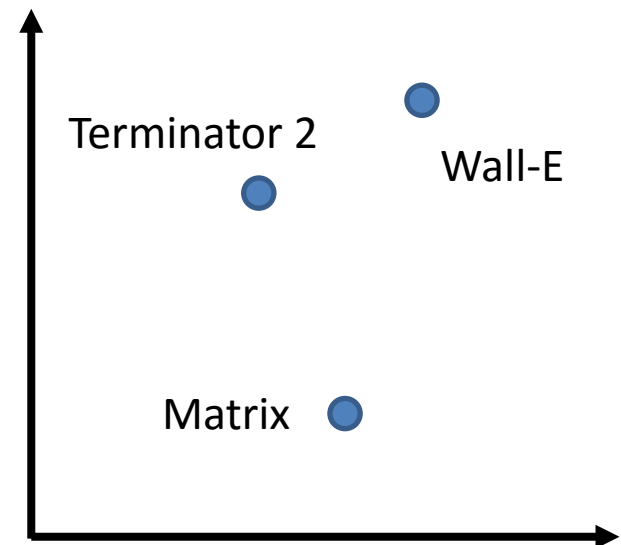
# Examples: Mapping movies

- Ask 100 people: “do you think movies X and Y are similar?”
- Similarity score = proportion of positive answers

distance =  $1 - \text{similarity}$

	Terminator 2	Matrix	Wall-E
Terminator 2		0.5	0.3
Matrix	0.5		0.7
Wall-E	0.3	0.7	

MDS  




# This lecture

- Principal components analysis
  - \* Linear dimensionality reduction method
  - \* Diagonalising covariance matrix
- Multidimensional scaling
  - \* Non-linear dimensionality reduction methods
  - \* Explicit search for a low-dimensional configuration