Dimensionality reduction

Outline

- From distances to points :
  - MultiDimensional Scaling (MDS)
- Dimensionality Reductions or data projections
- Random projections
- Singular Value Decomposition and Principal Component Analysis (PCA)

Multi-Dimensional Scaling (MDS)

- So far we assumed that we know both data points **X** and distance matrix **D** between these points

- What if the original points **X** are not known but only distance matrix **D** is known?

- Can we reconstruct **X** or some approximation of **X**?

Problem

- Given distance matrix **D** between **n** points
- Find a **k**-dimensional representation of every $\mathbf{x_i}$ point **i**
- So that $\mathbf{d(x_i,x_j)}$ is as close as possible to **D(i,j)**

Why do we want to do that?

How can we do that? (Algorithm)

High-level view of the MDS algorithm

- Randomly initialize the positions of **n** points in a **k**-dimensional space
- Compute pairwise distances **D'** for this placement
- Compare **D'** to **D**
- Move points to better adjust their pairwise distances (make **D'** closer to **D**)
- Repeat until **D'** is close to **D**

The MDS algorithm

- ***Input:*** **nxn** distance matrix **D**
- Random **n** points in the **k**-dimensional space **(x_1,…,x_n)**
- **stop = false**
- **while not stop**
  - **totalerror = 0.0**
  - For every **i,j** compute
    - **D'(i,j)=d(x_i,x_j)**
    - **error = (D(i,j)-D'(i,j))/D(i,j)**
    - **totalerror +=error**
    - **For** every dimension **m**: **$grad_{im}$ = (x_{im}-x_{jm})/D'(i,j)*error**
  - **If totalerror** small enough, **stop = true**
  - **If(!stop)**
    - **For every point i and every dimension m: x_{im}= x_{im} - rate*$grad_{im}$**

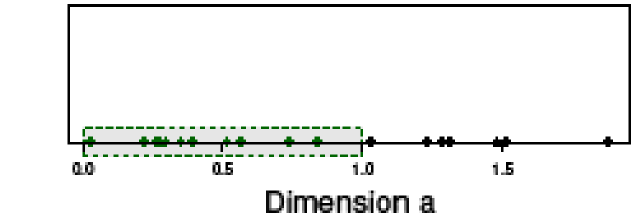
Questions about MDS

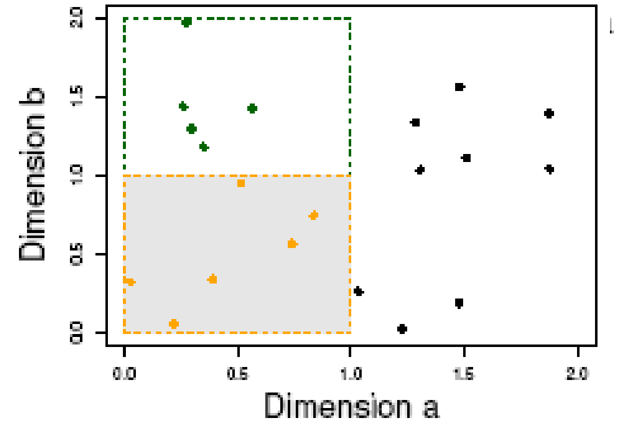
- Running time of the MDS algorithm
  - **$O(n^2 I)$,** where **I** is the number of iterations of the algorithm

- MDS does not guarantee that metric property is maintained in **D’**

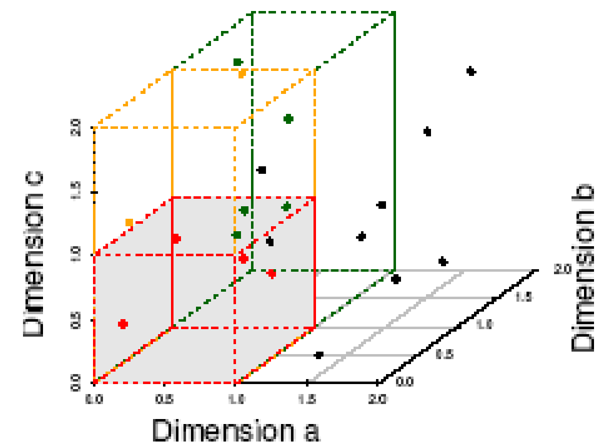
The Curse of Dimensionality

- Data in only one dimension is relatively packed
- Adding a dimension "stretches" the points across that dimension, making them further apart
- Adding more dimensions will make the points further apart—high dimensional data is extremely sparse
- Distance measure becomes meaningless





(b) 6 Objects in One Unit Bin



(c) 4 Objects in One Unit Bin

(graphs from Parsons et al. KDD Explorations 2004)

The curse of dimensionality

- The efficiency of many algorithms depends on the number of dimensions **d**
  - Distance/similarity computations are at least linear to the number of dimensions
  - Index structures fail as the dimensionality of the data increases

Goals

- Reduce dimensionality of the data
- Maintain the meaningfulness of the data

Dimensionality reduction

- Dataset **X** consisting of **n** points in a **d**-dimensional space
- Data point $\mathbf{x_i \epsilon R^d}$ (**d**-dimensional real vector):

  $\mathbf{x_i = [x_{i1}, x_{i2},\ldots, x_{id}]}$
- Dimensionality reduction methods:
  - **Feature selection:** choose a subset of the features
  - **Feature extraction:** create new features by combining new ones

Dimensionality reduction

- Dimensionality reduction methods:
  - **Feature selection:** choose a subset of the features
  - **Feature extraction:** create new features by combining new ones
- Both methods map vector $x_i \epsilon R^d$, to vector $y_i \epsilon R^k$, $(k<<d)$
- $F : R^d \rightarrow R^k$

Linear dimensionality reduction

- Function **F** is a ***linear*** projection
- $y_i = A\ x_i$
- $Y = A\ X$
- **Goal:** **Y** is as ***close*** to **X** as possible

Closeness: Pairwise distances

- **Johnson-Lindenstrauss lemma:** Given **$\varepsilon>0$**, and an integer **n**, let **k** be a positive integer such that **$k \geq k_0 = O(\varepsilon^{-2} \log n)$**. For every set **$X$** of **$n$** points in **$R^d$** there exists **$F: R^d \rightarrow R^k$** such that for all **$x_i, x_j \in X$**

$$(1-\varepsilon)||x_i - x_j||^2 \leq ||F(x_i) - F(x_j)||^2 \leq (1+\varepsilon)||x_i - x_j||^2$$

What is the intuitive interpretation of this statement?

JL Lemma: Intuition

- Vectors **$x_i \epsilon R^d$**, are projected onto a **k**-dimensional space (**$k<<d$**): **$y_i = x_i A$**
- If **$||x_i||=1$** for all **i**, then,

  $||x_i-x_j||^2$ is approximated by **$(d/k)||x_i-x_j||^2$**
- **Intuition:**
  - The expected squared norm of a projection of a unit vector onto a random subspace through the origin is **k/d**
  - The probability that it deviates from expectation is very small

Finding random projections

- Vectors $\mathbf{x_i} \in \mathbf{R^d}$, are projected onto a **k**-dimensional space (**k<<d**)
- Random projections can be represented by linear transformation matrix **A**
- $\mathbf{y_i = x_i\ A}$

- What is the matrix **A**?

Finding random projections

- Vectors $x_i \epsilon R^d$, are projected onto a **k**-dimensional space (**k<<d**)
- Random projections can be represented by linear transformation matrix **A**
- $y_i = x_i A$

- What is the matrix **A**?

Finding matrix A

- Elements **A(i,j)** can be Gaussian distributed
- Achlioptas* has shown that the Gaussian distribution can be replaced by

$$A(i,j) = \begin{cases} +1 \text{ with } & \text{prob } \frac{1}{6} \\ 0 \text{ with } & \text{prob } \frac{2}{3} \\ -1 \text{ with } & \text{prob } \frac{1}{6} \end{cases}$$

- All zero mean, unit variance distributions for **A(i,j)** would give a mapping that satisfies the ***JL*** lemma

- **Why is Achlioptas result useful?**

Datasets in the form of matrices

We are given **n** objects and **d** features describing the objects. (Each object has **d** numeric values describing it.)

Dataset

An **n-by-d** matrix **A**, A_{ij} shows the "***importance***" of feature **j** for object **i**.

Every row of **A** represents an object.

Goal

1. **Understand** the structure of the data, e.g., the underlying process generating the data.
2. **Reduce the number of features** representing the data

Market basket matrices

d products

(e.g., milk, bread, wine, etc.)

n customers

$$A$$

A_{ij} = quantity of **j**-th product purchased by the **i**-th customer

Find a subset of the products that characterize customer behavior

Social-network matrices

d groups
(e.g., BU group, opera, etc.)

n users

$$A$$

A_{ij} = partitipication of the **i**-th user in the **j**-th group

Find a subset of the groups that accurately clusters social-network users

Document matrices

d terms
(e.g., theorem, proof, etc.)

n documents

$$\begin{pmatrix} \\ A \\ \\ \end{pmatrix}$$

A_{ij} = frequency of the **j**-th term in the **i**-th document

Find a subset of the terms that accurately clusters the documents

Recommendation systems

d products

n customers

$$A$$

A_{ij} = frequency of the **j**-th product is bought by the **i**-th customer

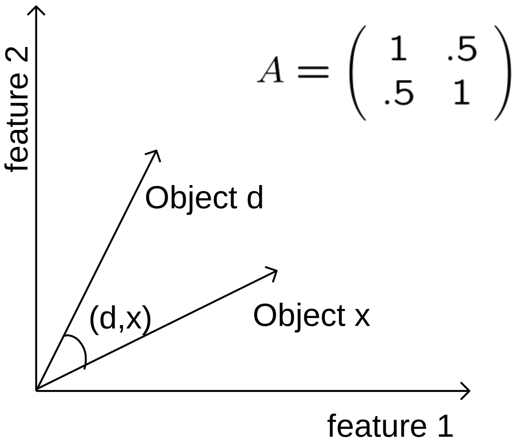
Find a subset of the products that accurately describe the behavior or the customers

The Singular Value Decomposition (SVD)

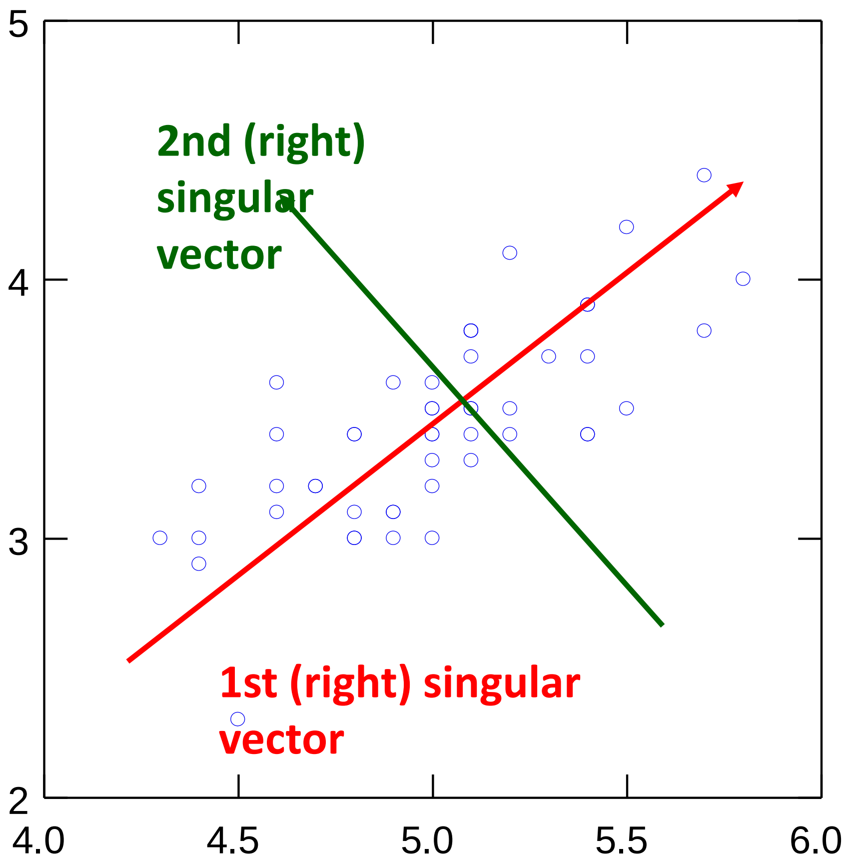
Data matrices have **n** rows (one for each object) and **d** columns (one for each feature).

Rows: vectors in a Euclidean space,

Two objects are "***close***" if the angle between their corresponding vectors is small.



SVD: Example



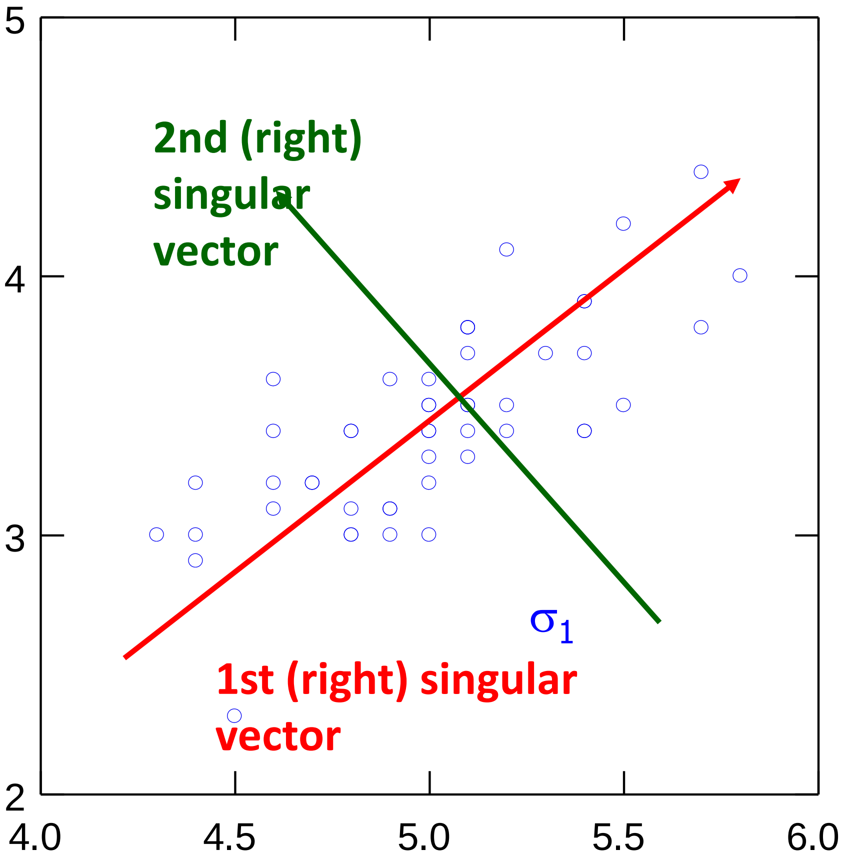
Input: **2-d** dimensional points

Output:

1st (right) singular vector:
direction of maximal variance,

2nd (right) singular vector:
direction of maximal variance, after removing the projection of the data along the first singular vector.

Singular values



σ_1: measures how much of the data variance is explained by the first singular vector.

σ_2: measures how much of the data variance is explained by the second singular vector.

SVD decomposition

$$\begin{pmatrix} & & \\ & A & \\ & & \end{pmatrix} = \begin{pmatrix} & & \\ & U & \\ & & \end{pmatrix} \cdot \begin{pmatrix} \Sigma & \\ 0 & \end{pmatrix} \cdot \begin{pmatrix} & V & \end{pmatrix}^T$$

n x d **n x ℓ** **ℓ x ℓ** **ℓ x d**

U (V): orthogonal matrix containing the left (right) singular vectors of **A.**

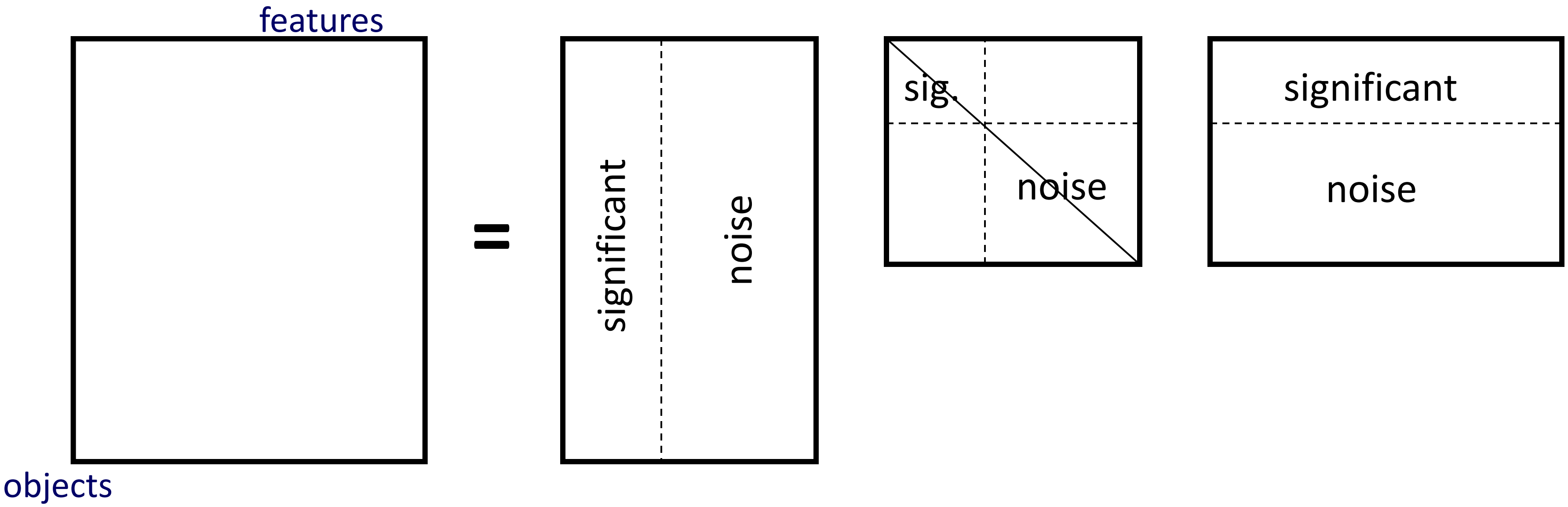
Σ: diagonal matrix containing the **singular values** of **A:**

($\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_\ell$)

Exact computation of the SVD takes **$O(\min\{mn^2, m^2n\})$** time. The top *k* left/right singular vectors/values can be ***computed faster*** using Lanczos/Arnoldi methods.

SVD and Rank-**k** approximations

A **=** **U** **Σ** **V**^{T}



Rank-k approximations (A_k)

$$\begin{pmatrix} \\ \\ A_k \\ \\ \\ \end{pmatrix} = \begin{pmatrix} \\ \\ U_k \\ \\ \\ \end{pmatrix} \cdot \begin{pmatrix} \\ \Sigma_k \\ \\ \end{pmatrix} \cdot \begin{pmatrix} \\ V_k^T \\ \\ \end{pmatrix}$$

n x d **n x k** **k x d**

A_k is the **best** approximation of **A**

U_k (V_k): orthog

singular vecto

Σ_k: diagonal ma

A_k is an approximation of A

SVD as an optimization problem

Find **C** to minimize:

$$\min_{C} \left\| \underset{n\times d}{A} - \underset{n\times k}{C} \; \underset{k\times d}{X} \right\|_F^2$$

Frobenius norm:

$$\|A\|_F^2 = \sum_{i,j} A_{ij}^2$$

Given **C** it is easy to find **X** from standard least squares. However, the fact that we can find the optimal **C** is fascinating!

PCA and SVD

- PCA is SVD done on **centered** data
- PCA looks for such a direction that the data projected to it has the maximal variance
- PCA/SVD continues by seeking the next direction that is orthogonal to all previously found directions
- All directions are orthogonal

How to compute the PCA

- Data matrix **A**, ***rows = data points***, ***columns = variables*** (attributes, features, parameters)

1. Center the data by subtracting the mean of each column
2. Compute the SVD of the centered matrix **A'** (i.e., find the first **k** singular values/vectors) **A' = UΣV^T**
3. The principal components are the columns of **V**, the coordinates of the data in the basis defined by the principal components are **UΣ**

Singular values tell us something about the variance

- The variance in the direction of the **k**-th principal component is given by the corresponding singular value σ_k^2

- Singular values can be used to estimate how many components to keep

- ***Rule of thumb:*** keep enough to explain ***85%*** of the variation:

$$\frac{\sum_{j=1}^{k} \sigma_j^2}{\sum_{j=1}^{n} \sigma_j^2} \approx 0.85$$

SVD is “**the Rolls-Royce and the Swiss Army Knife of Numerical Linear Algebra.**”*
*Dianne O’Leary, MMDS ’06

SVD as an optimization problem

Find C to minimize:

$$\min_C \left\| \underset{n\times d}{A} - \underset{n\times k}{C} \; \underset{k\times d}{X} \right\|_F^2$$

Frobenius norm:

$$\|A\|_F^2 = \sum_{i,j} A_{ij}^2$$

Given C it is easy to find X from standard least squares. However, the fact that we can find the optimal C is fascinating!