DATA MINING
LECTURE 8

Sequence Segmentation

Dimensionality Reduction

SEQUENCE SEGMENTATION

Why deal with sequential data?

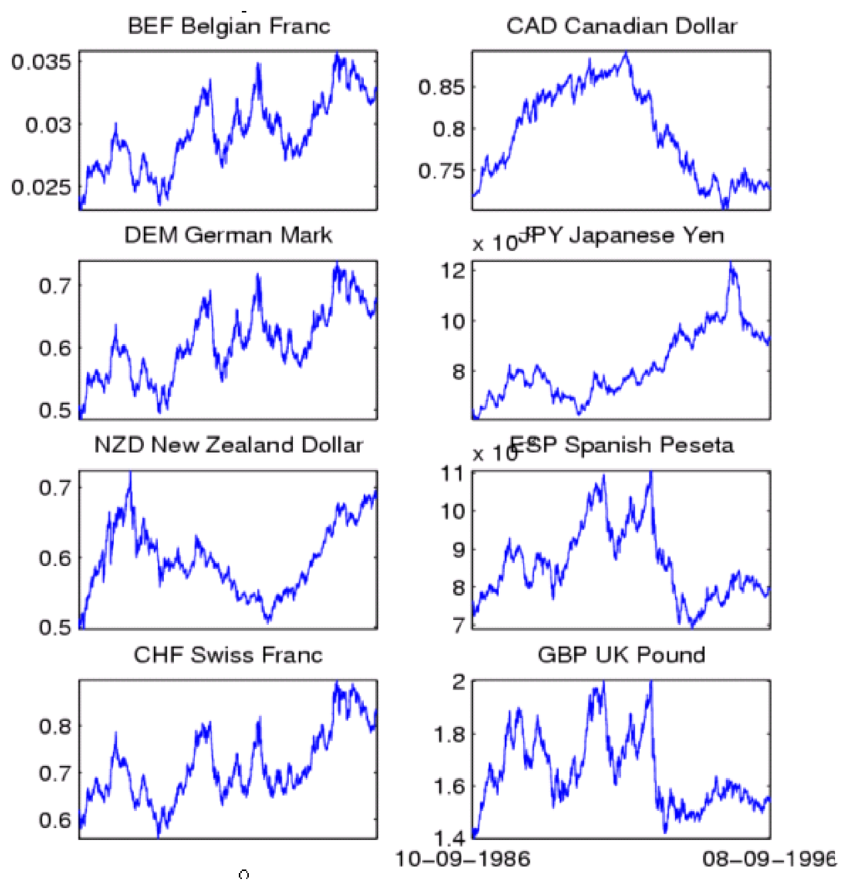
- Because all data is sequential ☺
  - All data items arrive in the data store in some order

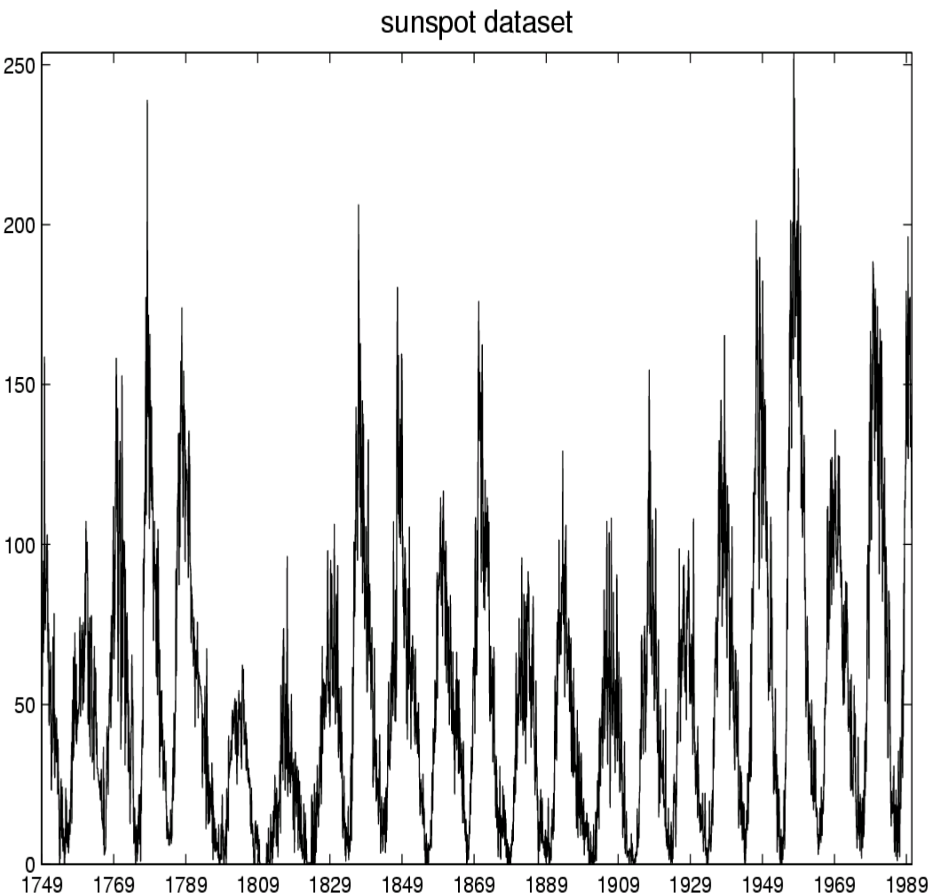
- Examples
  - transaction data
  - documents and words

- In some (many) cases the order does not matter

- In many cases the order is of interest

Time-series data





- Financial time series, process monitoring…

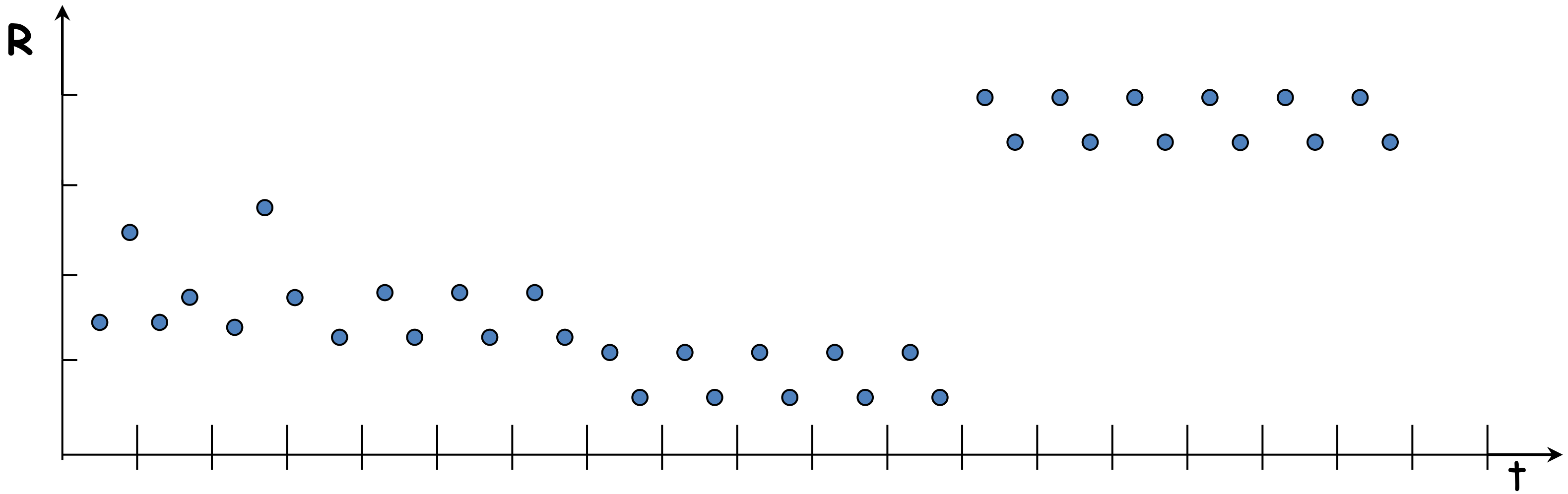
Sequence Segmentation

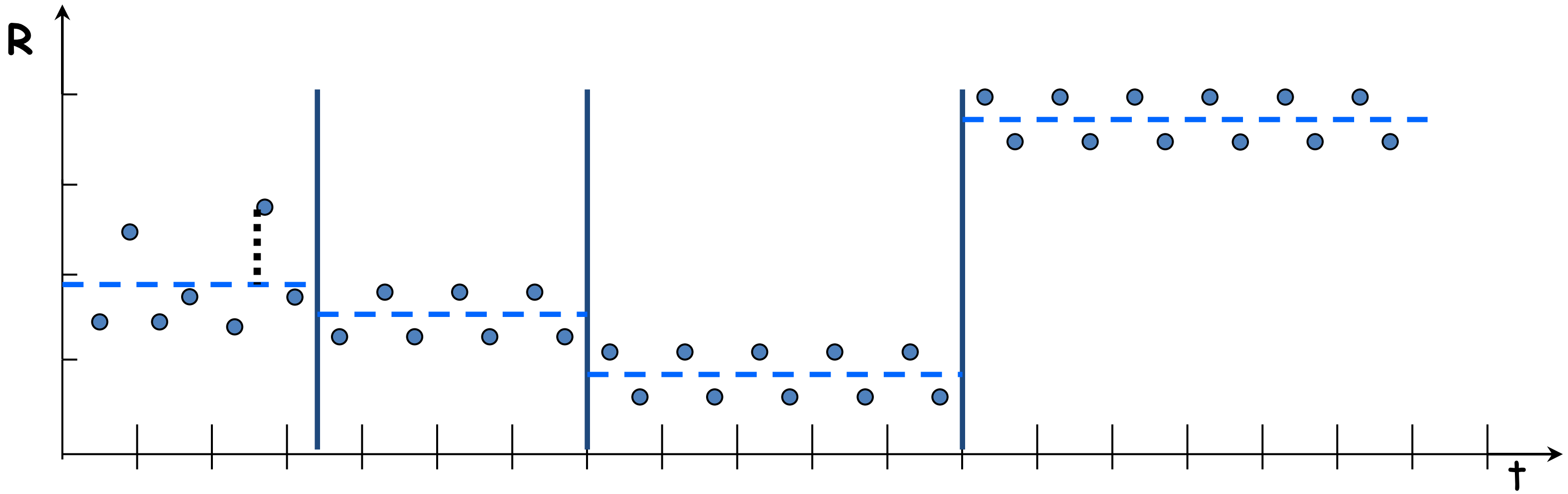
- Goal: discover structure in the sequence and provide a concise summary

- Given a sequence T, segment it into K contiguous segments that are as homogeneous as possible

- Similar to clustering but now we require the points in the cluster to be contiguous

Example





Basic definitions

- Sequence $T = \{t_1,t_2,\dots,t_N\}$: an ordered set of N d-dimensional real points $t_i \in R^d$
- A k-segmentation S: a partition of T into K contiguous segments $\{s_1,s_2,\dots,s_K\}$.
  - Each segment $s \in S$ is represented by a single value $\mu_s \in R^d$ (the representative of the segment)
- Error $E(S)$: The error of replacing individual points with representatives
  - Sum of Squares Error (SSE):

$$E(S) = \sum_{s \in S} \sum_{t \in s} (t - \mu_s)^2$$

  - Representative for SSE: mean $\mu_s = \frac{1}{|s|} \sum_{t \in s} t$

The K-segmentation problem

Given a sequence T of length N and a value K, find a K-segmentation $S = \{s_1, s_2, \ldots, s_K\}$ of T such that the SSE error E is minimized.

- Similar to K-means clustering, but now we need the points in the clusters to respect the order of the sequence.
  - This actually makes the problem easier.

Optimal solution for the k-segmentation problem

- Bellman'61: The k-segmentation problem can be solved optimally using a standard dynamic-programming algorithm

- Dynamic Programming:
  - Construct the solution of the problem by using solutions to problems of smaller size
    - Define the dynamic programming recursion
  - Build the solution bottom up from smaller to larger instances
    - Define the dynamic programming table that stores the solutions to the sub-problems

Dynamic Programming Solution

- Terminology:
  - $E(S[1,n],k)$: error of optimal segmentation of subsequence $T[1,n]$ with k segments

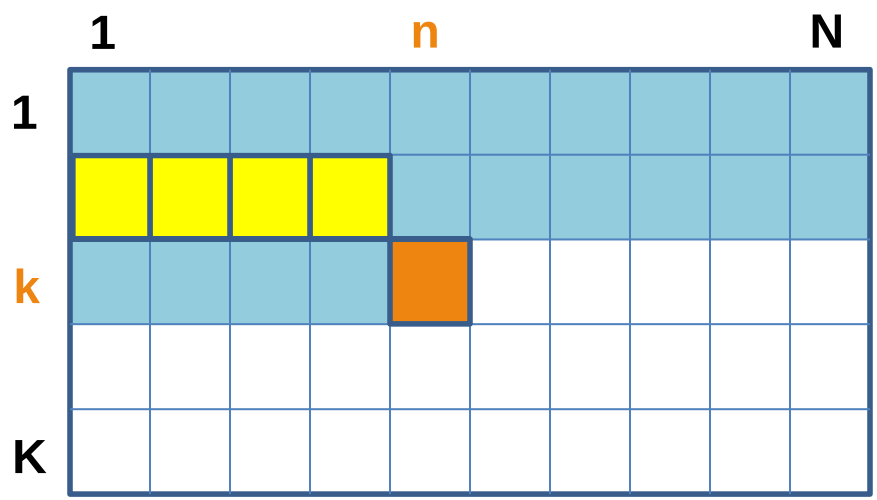
- Dynamic Programming Recursion:

$$E(S[1,n],k) = \min_{1 \le j < n} \left\{ E(S[1,j],k-1) + \sum_{j+1 \le t \le n} \left(t - \mu_{[j+1,n]}\right)^2 \right\}$$

- Dynamic programming table:
  - Two-dimensional table A[1…K,1…N]
  - A[k,n] = $E(S[1,n],k)$

Dynamic programming solution

$$E(S[1,n],k) = \min_{1\leq j<n} \left\{ E(S[1,j],k-1) + \sum_{j+1\leq t\leq n} \left(t - \mu_{[j+1,n]}\right)^2 \right\}$$



- Fill the table row to tow from smaller to larger values of k

Algorithm Complexity

- What is the complexity?
  - NK cells to fill
  - $E(S[1,n],k) = \min_{1 \le j < n} \left\{ E(S[1,j],k-1) + \sum_{j+1 \le t \le n} \left(t - \mu_{[j+1,n]}\right)^2 \right\}$
    - O(N) cells to check for each of the cells
    - O(N) to compute the second term
  - $O(N^3K)$ in the naïve computation
- $\sum_{j+1 \le t \le n} \left(t - \mu_{[j+1,n]}\right)^2 = \sum_{j+1 \le t \le n} t^2 + \frac{1}{n-j+2} \sum_{j+1 \le t \le n} t$
- We can compute in constant time by precomputing partial sums
  - $O(N^2K)$ complexity

Heuristics

- Bottom-up greedy (BU): O(NlogN)
  - Merge adjacent points each time selecting the two points that cause the smallest increase in the error until K segments
  - [Keogh and Smyth'97, Keogh and Pazzani'98]
- Top-down greedy (TD): O(NK)
  - Introduce breakpoints so that you get the largest decrease in error, until K segments are created.
  - [Douglas and Peucker'73, Shatkay and Zdonik'96, Lavrenko et. al'00]
- Local Search Heuristics: O(NKI)
  - Assign the breakpoints randomly and then move them so that you reduce the error
  - [Himberg et. al '01]

DIMENSIONALITY REDUCTION

The curse of dimensionality

- Real data usually have thousands, or millions of dimensions
  - E.g., web documents, where the dimensionality is the vocabulary of words
  - Facebook graph, where the dimensionality is the number of users
- Huge number of dimensions causes many problems
  - Data becomes very sparse, some algorithms become meaningless (e.g. density based clustering)
  - The complexity of several algorithms depends on the dimensionality and they become infeasible.

Dimensionality Reduction

- Usually the data can be described with fewer dimensions, without losing much of the meaning of the data.
  - The data reside in a space of lower dimensionality

- Essentially, we assume that some of the data is noise, and we can approximate the useful part with a lower dimensionality space.
  - Dimensionality reduction does not just reduce the amount of data, it often brings out the useful part of the data

Data in the form of a matrix

- We are given n objects and d attributes describing the objects. Each object has d numeric values describing it.

- We will represent the data as a $n \times d$ real matrix A.
  - We can now use tools from linear algebra to process the data matrix

- Our goal is to produce a new $n \times k$ matrix B such that
  - It preserves as much of the information in the original matrix A as possible
  - It reveals something about the structure of the data in A

Example: 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 subsets of terms that bring documents together

Example: Recommendation systems

$$\begin{array}{cc} & \textbf{d} \text{ movies} \\ \begin{array}{l}\textbf{n} \\ \text{customers}\end{array} & \left(\begin{array}{c} \\ \\ A \\ \\ \end{array}\right) \end{array}$$

A$_{ij}$ = rating of **j**-th product by the **i**-th customer

Find subsets of movies that capture the behavior or the customers

Some linear algebra basics

- We assume that vectors are column vectors. We use v^T for the transpose of vector v (row vector)
  - Dot product: $u^T v$ $(1\times n, n\times 1 \;\rightarrow\; 1\times 1)$
    - The dot product is the projection of vector u on v
  - External product: uv^T $(n\times 1\,, 1\times m \;\rightarrow\; n\times m)$
    - The resulting $n\times m$ has rank 1: all rows (or columns) are linearly dependent
  - Rank of matrix A: The number of linearly independent vectors (column or row) in the matrix.
- Eigenvector of matrix A: a vector v such that $Av = \lambda v$

Singular Value Decomposition

$$A = U \quad \Sigma \quad V^T = \begin{bmatrix} \vec{u}_1 & \vec{u}_2 & \cdots & \vec{u}_r \end{bmatrix} \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_r \end{bmatrix} \begin{bmatrix} \vec{v}_1 \\ \vec{v}_2 \\ \vdots \\ \vec{v}_r \end{bmatrix}$$

$$[n \times r]\,[r \times r]\,[r \times n]$$

- **r** : rank of matrix A
- $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r$: singular values (square roots of eig-vals AA^T, A^TA)
- $\vec{u}_1, \vec{u}_2, \cdots, \vec{u}_r$: left singular vectors (eig-vectors of AA^T)
- $\vec{v}_1, \vec{v}_2, \cdots, \vec{v}_r$: right singular vectors (eig-vectors of A^TA)
- $$A = \sigma_1 \vec{u}_1 \vec{v}_1^T + \sigma_2 \vec{u}_2 \vec{v}_2^T + \cdots + \sigma_r \vec{u}_r \vec{v}_r^T$$

Singular Value Decomposition

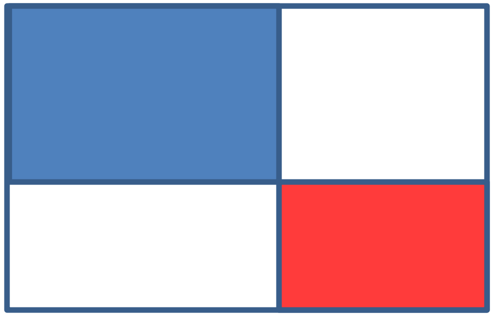
- What does it mean?

- If A has rank r, then A can be written as the sum of r rank-1 matrices

- There are r linear trends in A.
  - Linear trend: the tendency of the row vectors of A to align with vector **v**
  - Strength of the i-th linear trend: $||A\boldsymbol{v}_i|| = \boldsymbol{\sigma}_i$

An (extreme) example

- Document-term matrix
  - Blue and Red rows (colums) are linearly depedent

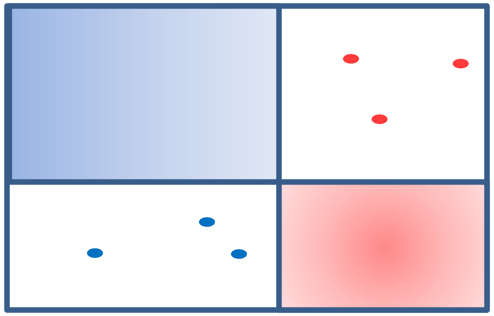
A = 

- There are two types of documents (words): blue and red
  - To describe the data is enough to describe the two types, and the projection weights for each row
  - A is a rank-2 matrix

An (more realistic) example

- Document-term matrix

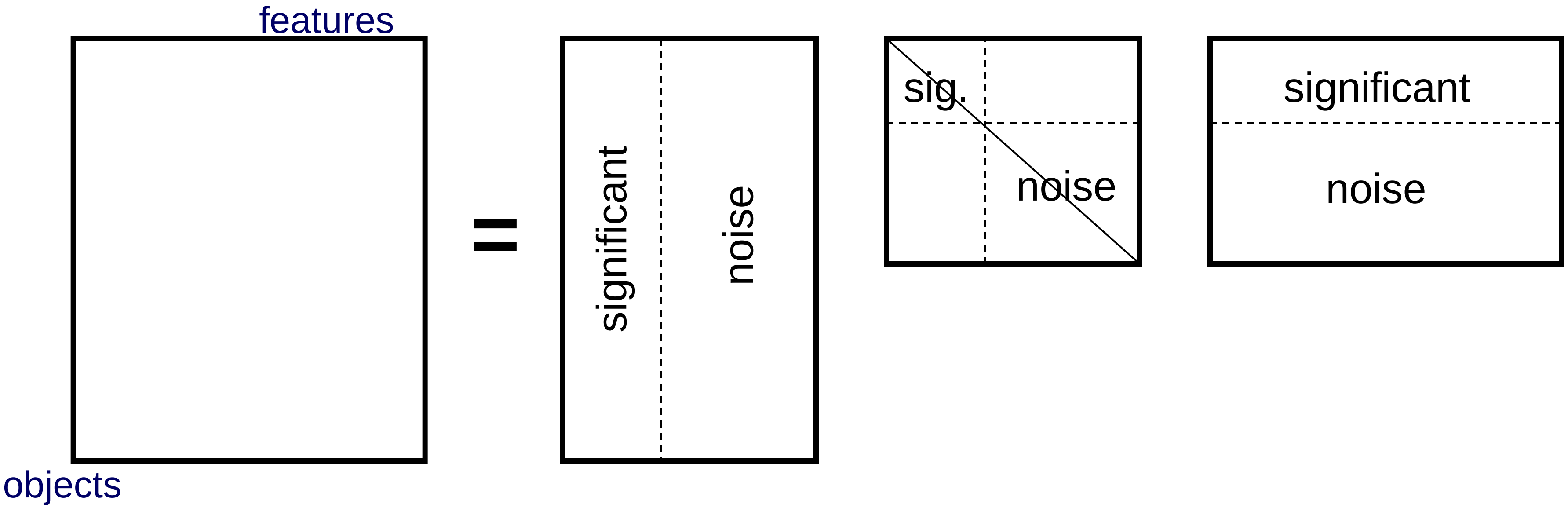
A =



- There are two types of documents and words but they are mixed
  - We now have more than two singular vectors, but the strongest ones are still about the two types.
  - By keeping the two strongest singular vectors we obtain most of the information in the data.
    - This is a rank-2 approximation of the matrix A

SVD and Rank-k approximations

$$\mathbf{A} = \mathbf{U} \quad \mathbf{\Sigma} \quad \mathbf{V}^T$$



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 k** **k x d**

U_k (V_k): orthogonal matrix containing the top ***k*** left (right) singular vectors of **A**.
Σ_k: diagonal matrix containing the top ***k*** singular values of **A**

A_k is an approximation of **A**

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

SVD as an optimization

- The rank-k approximation matrix A_k produced by the top-k singular vectors of A minimizes the Frobenious norm of the difference with the matrix A

$$A_k = \arg \max_{B:rank(B)=k} ||A - B||_F^2$$

$$||A - B||_F^2 = \sum_{i,j} \left(A_{ij} - B_{ij}\right)^2$$

What does this mean?

- We can project the row (and column) vectors of the matrix A into a k-dimensional space and preserve most of the information
- (Ideally) The k dimensions reveal latent features/aspects/topics of the term (document) space.
- (Ideally) The A_k approximation of matrix A, contains all the useful information, and what is discarded is noise

Two applications

- Latent Semantic Indexing (LSI):
  - Apply SVD on the document-term space, and index the k-dimensional vectors
  - When a query comes, project it onto the low dimensional space and compute similarity cosine similarity in this space
  - Singular vectors capture main topics, and enrich the document representation
- Recommender systems and collaborative filtering
  - In a movie-rating system there are just a few types of users.
  - What we observe is an incomplete and noisy version of the true data
  - The rank-k approximation reconstructs the “true” matrix and we can provide ratings for movies that are not rated.

SVD and PCA

- PCA is a special case of SVD on the centered covariance matrix.

Covariance matrix

- Goal: reduce the dimensionality while preserving the "information in the data"
- Information in the data: variability in the data
  - We measure variability using the covariance matrix.
  - Sample covariance of variables X and Y

$$\sum_i (x_i - \mu_X)^T (y_i - \mu_Y)$$

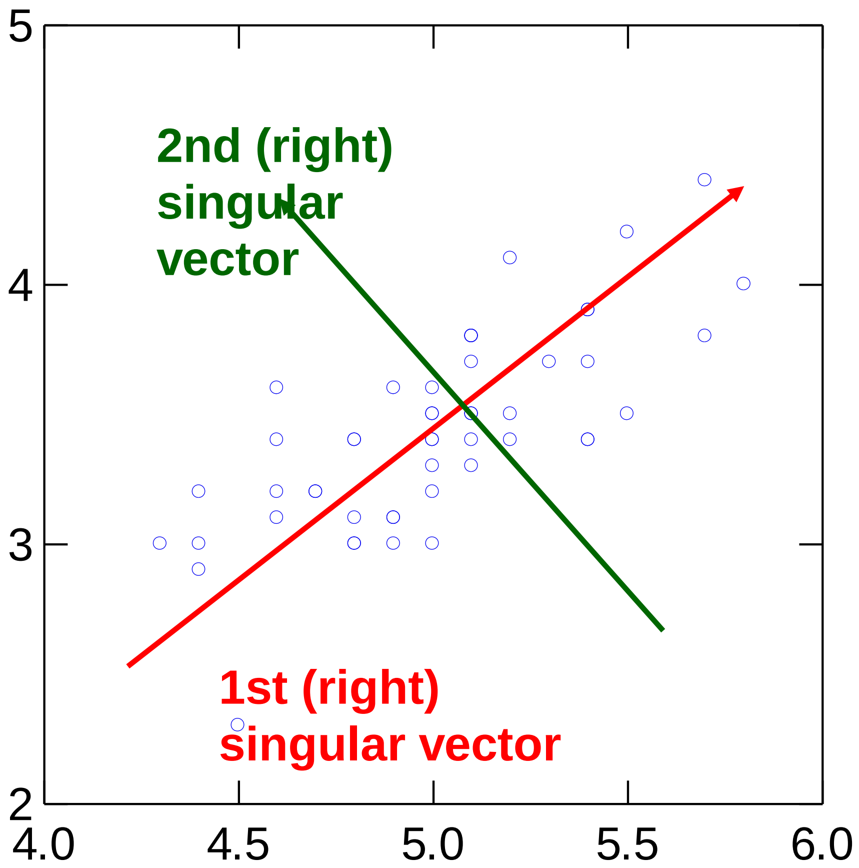
- Given matrix A, remove the mean of each column from the column vectors to get the centered matrix C
- The matrix $V = C^T C$ is the covariance matrix of the row vectors of A.

PCA: Principal Component Analysis

- We will project the rows of matrix A into a new set of attributes such that:
  - The attributes have zero covariance to each other (they are orthogonal)
  - Each attribute captures the most remaining variance in the data, while orthogonal to the existing attributes
    - The first attribute should capture the most variance in the data

- For matrix C, the variance of the rows of C when projected to vector x is given by $\sigma^2 = ||Cx||^2$
  - The right singular vector of C maximizes σ^2!

PCA



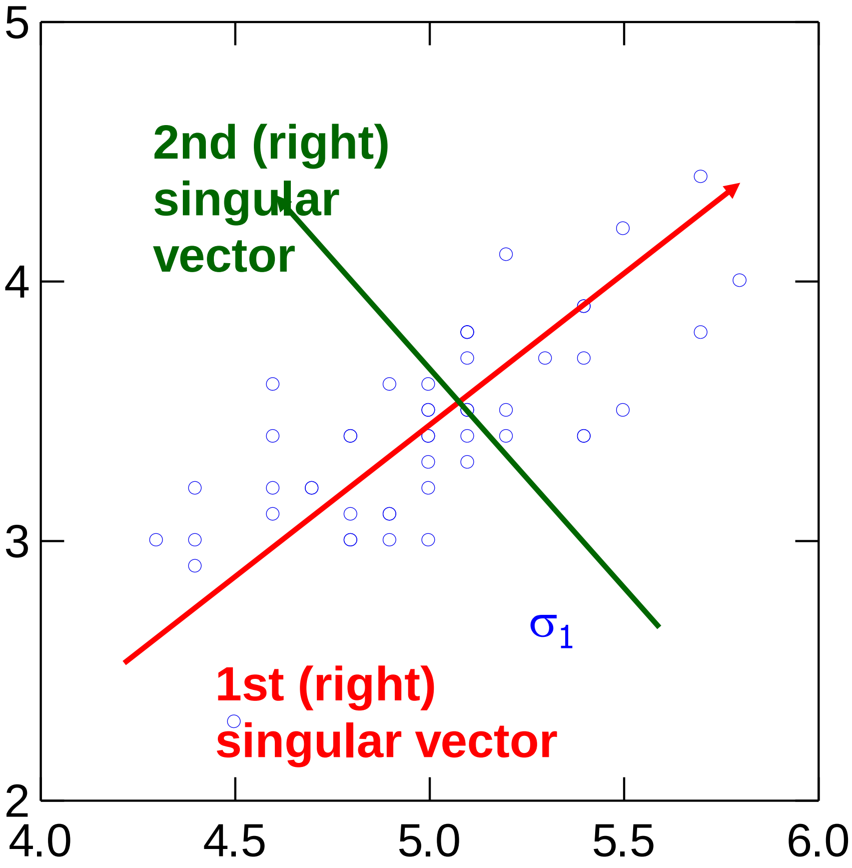
Input: **2-d** dimensional points

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

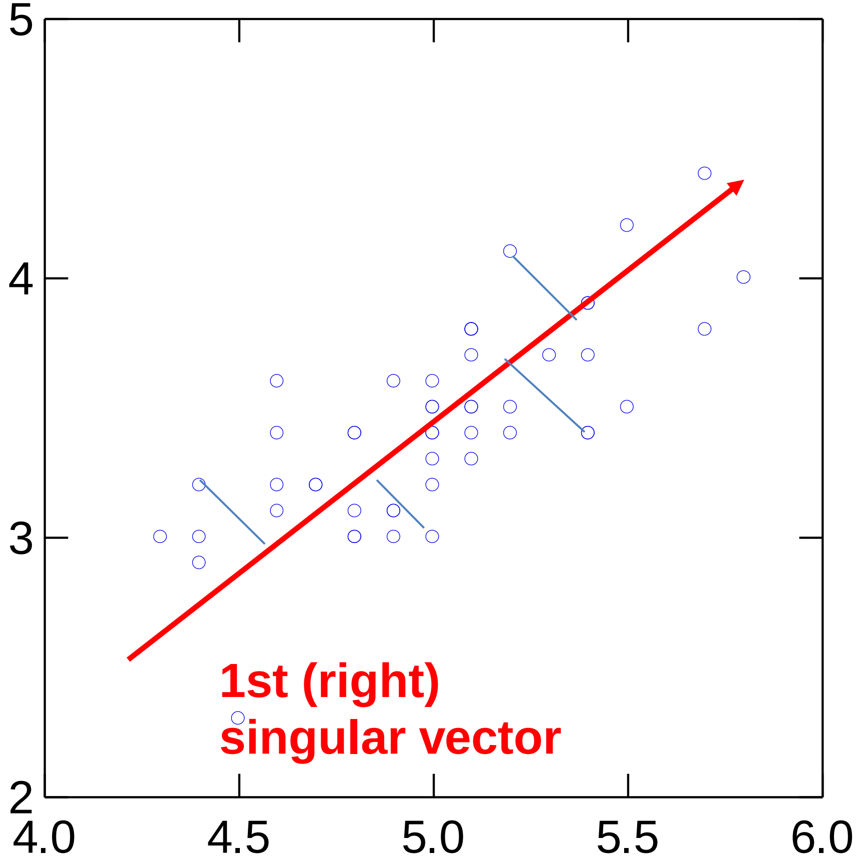


σ_1: measures how much of the data variance is explained by the first singular vector.

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

Another property of PCA/SVD

- The chosen vectors are such that minimize the sum of square differences between the data vectors and the low-dimensional projections



SVD is “**the Rolls-Royce and the Swiss Army Knife of Numerical Linear Algebra.”***

*Dianne O’Leary, MMDS ’06