# Part-II: Applications of Matrix Algorithms

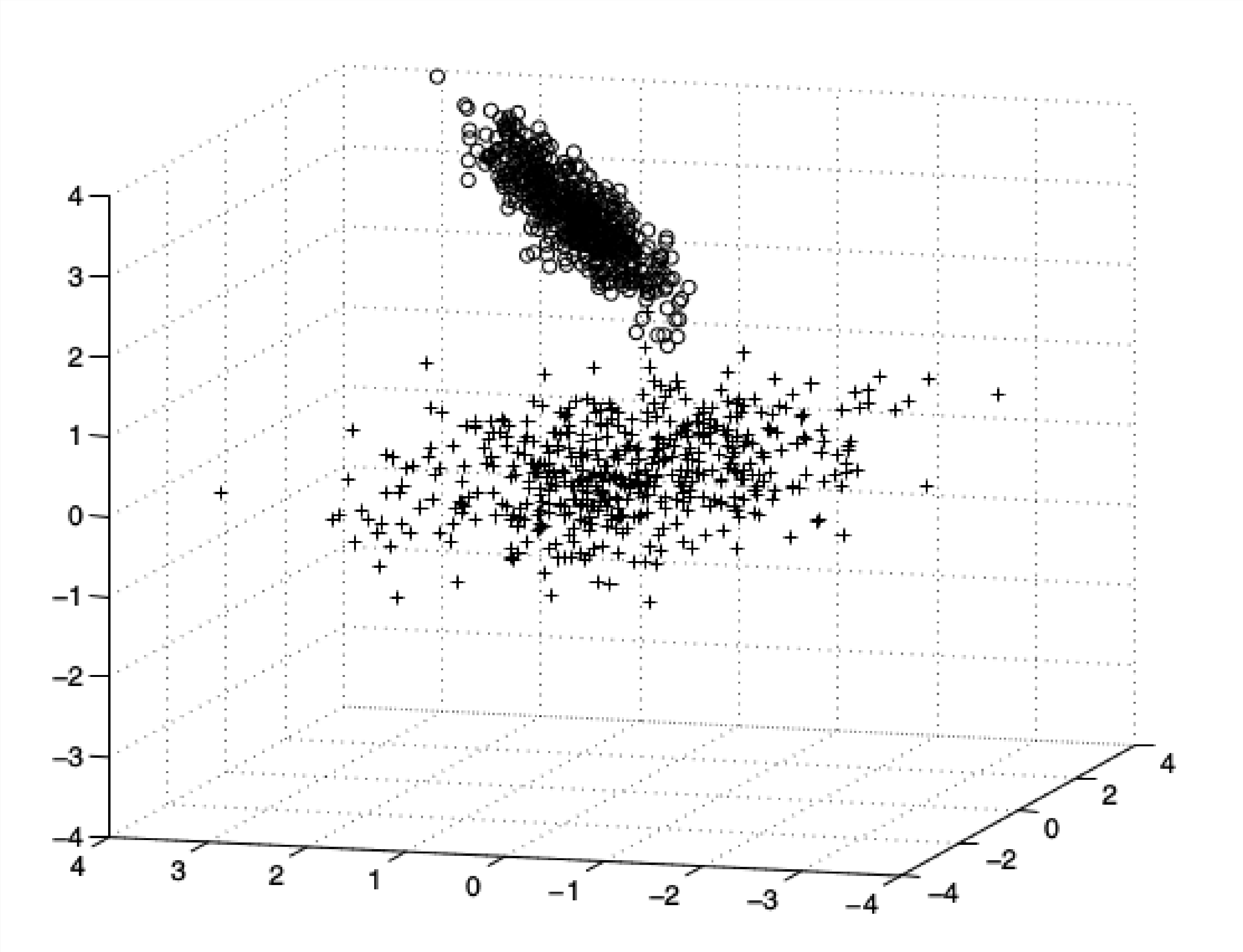
# Motivation

Goal: Data compression and classification

Idea: Organize data points in clusters

- Cluster: Subset of set of data points that are close together in some distance measure.
- Compute the mean of the clusters and use the mean as representative for clusters
- Means can be used as basis vectors, and all data points reprersented in this basis (Why?)

# Example of clusters




- One of the most important algorithm is $k-$means algorithm

# Non-negative Matrices in Data Mining

- In data mining matrix is often non-negative (e.g., user ratings)
- The SVD does not guarantee that singular vectors are non-negative
- Want to compute low-rank approximation:

$$A \approx WH, \quad W, H \geq 0$$

# $k-$Means Algorithm

- Given $n$ data points $(a_j)_{j=1}^{n} \in \mathbb{R}^m$, kept as columns in $A \in \mathbb{R}^{m\times n}$
- Let $\Pi = (\pi_i)_{i=1}^{k}$ denote the partitioning of the vectors into $k$ clusters

$$\pi_j = \{\nu \mid a_\nu belongstoclusterj\}$$

- Let the mean or the centroid of the cluster be

$$m_j = \frac{1}{n_j}\sum_{\nu\in\pi_j} a_\nu$$

# Distance measure, Quality of clustering

Coherence of cluster: The coherence of cluster $\pi_j$ can be measured as

$$q_j = \sum_{\nu \in \pi_j} \|a_\nu - m_j\|_2^2$$

- Closer the vectors to centroid, smaller is $q_j$

Quality of clustering:

Quality of clustering = overall coherence

$$Q(\Pi) = \sum_{j=1}^{k} q_j = \sum_{j=1}^{k} \sum_{\nu \in \pi_j} \|a_\nu - m_j\|_2^2$$

# $k-$means algorithm

$k-$Means Methods: Find a partitioning that has optimal coherence, i.e., solve the minimization problem:

$$\min_{\Pi} Q(\Pi)$$

- Given a provisional partitioning, compute the centroids
- For each data point in the particular cluster, check whether there is another centroid closer to this data point.
    - If yes, then move these data point to the cluster whose centroid is closest to these data points
- Repeat until satisfactory quality

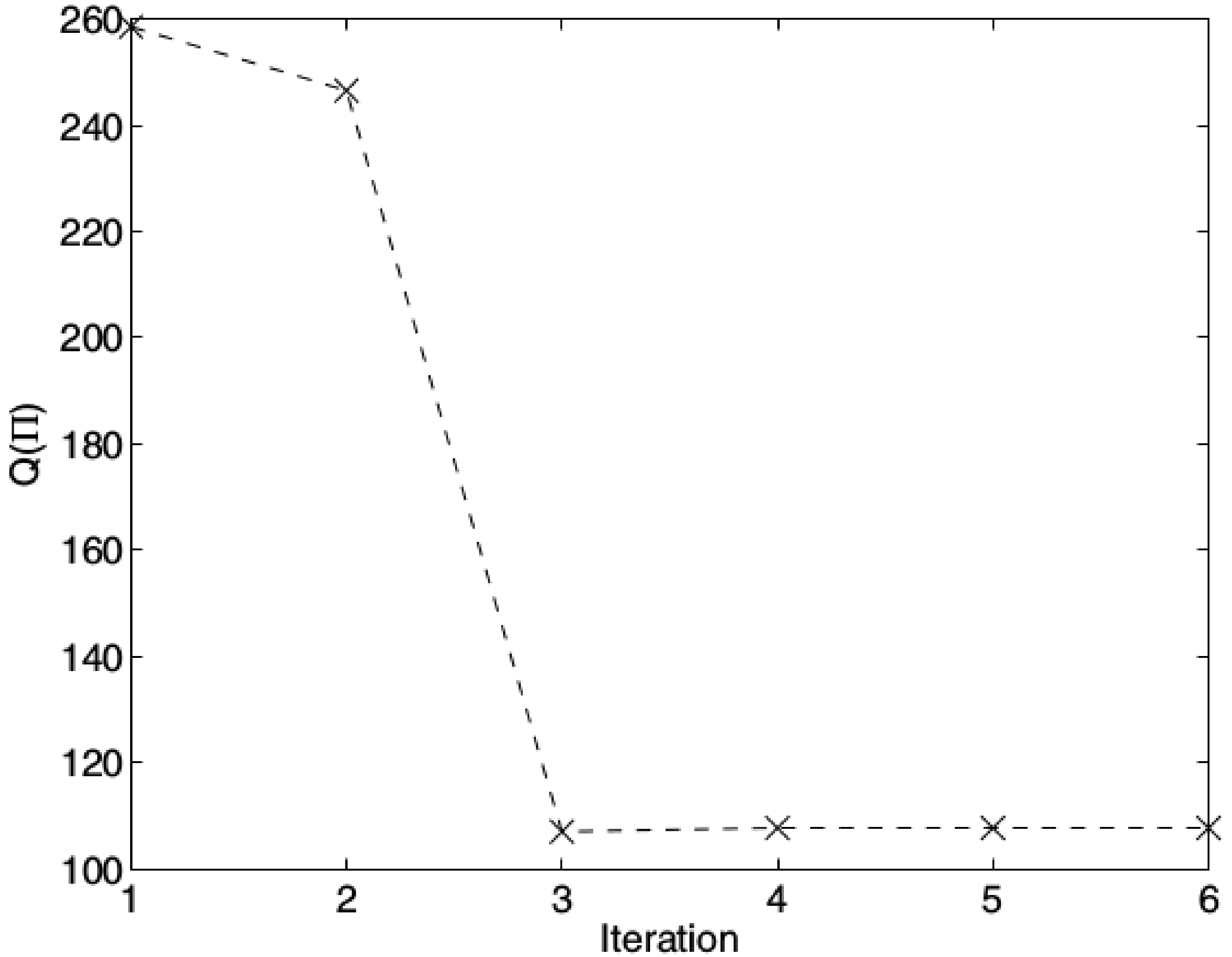
# $k-$means algorithm

1. Start with an initial partitioning $\Pi^0$ and compute the corresponding centroid vectors, $(m_j^{(0)})_{j=1}^k$. Compute $Q(\Pi^0)$. Put $t = 1$
2. For each vector $a_i$, find the closest centroid. If the closest vector is $m_p^{(t-1)}$, assign $a_i$ to $\pi_p^{(t)}$
3. Compute the centroids $((m_j)^{(t)})_{j=1}^k$ of the new partitioning $\Pi^{(t)}$
4. If $|Q(\Pi^{(t-1)}) - Q(\Pi^{(t)})| < tol$, then stop, otherwise increment $t$ by 1 and go to step 2

# An Example Run of $k-$Means

Breat Cancer Diagnosis: The matrix $A \in \mathbb{R}^{9\times 683}$ contains data from breast cytology tests. Out of 683 tests, 444 represent diagnosis of benign, and 239 a diagnosis of malignant. Iterate $k-$means with $k = 2$, until relative difference in $Q(\Pi)$ was $< 10^{-10}$.

- With random initial partitioning, the algorithm converged in 6 steps


260
240
220
200
180
160
140
120
100
Q(Π)
1
2
3
4
5
6
Iteration

# Other algorithms for $k-$means: Motivation

Term document matrix:

Consider the following documents:

- Document 1: The Google matrix P is a model of the Internet.
- Document 2: $P_{ij}$ is nonzero if there is a link from webpage $j$ to $i$
- Document 3: The google matrix is sued to rank all webpages
- Document 4: The ranking is done by solving a matrix eigenvalue problem
- Document 5: England dropped out of the top 10 in FIFA ranking

# Create Term Document Matrix

| Term | Doc 1 | Doc 2 | Doc 3 | Doc 4 | Doc 5 |
|---|---|---|---|---|---|
| eigenvalue | | | | | |
| England | | | | | |
| FIFA | | | | | |
| Google | | | | | |
| Internet | | | | | |
| link | | | | | |
| matrix | | | | | |
| page | | | | | |
| rank | | | | | |
| Web | | | | | |

# Which documents are related to a given query

Find all documents that are relevant to the query: "ranking of webpages". For this, construct query vector:

$$q = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}^T \in \mathbb{R}^{10}$$

Goal: Find the document that is closest to this query. That is, mathematically:

find the columns of A that are close to this query

# Classification of documents

Recall the term document matrix:

| Term | Doc 1 | Doc 2 | Doc 3 | Doc 4 | Doc 5 |
|---|---|---|---|---|---|
| eigenvalue | 0 | 0 | 0 | 1 | 0 |
| England | 0 | 0 | 0 | 0 | 1 |
| FIFA | 0 | 0 | 0 | 0 | 1 |
| Google | 1 | 0 | 1 | 0 | 0 |
| Internet | 1 | 0 | 0 | 0 | 0 |
| link | 0 | 1 | 0 | 0 | 0 |
| matrix | 1 | 0 | 1 | 1 | 0 |
| page | 0 | 1 | 1 | 0 | 0 |
| rank | 0 | 0 | 1 | 1 | 1 |
| Web | 0 | 1 | 1 | 0 | 0 |

- First four document deal with: Google, page ranking, etc. The last document deal with football.

# Towards Another Way to do Clustering!

- Create inital partitioning. Let first 4 docs be in one partition and the the last doc in another partition
- Compute normalized centroids for these partitions

$$C = \begin{pmatrix} 0.1443 & 0 \\ 0 & 0.5774 \\ 0 & 0.5774 \\ 0.2561 & 0 \\ 0.1443 & 0 \\ 0.1443 & 0 \\ 0.4005 & 0 \\ 0.2561 & 0 \\ 0.2561 & 0.5774 \\ 0.2561 & 0 \end{pmatrix}$$

# Motivation for Non-Negative Matrix Factorization

- Write coordinates of columns of $A$ in terms of the "centroid basis" by solving

$$\min_{D} \|A - CH\|$$

  - Do QR of $H$ to get: $C = QR$
  - Form the solution: $H = R^{-1}Q^T A$, where

$$H = \begin{pmatrix} 1.7283 & 1.4168 & 2.8907 & 1.5440 & 0.0000 \\ -0.2556 & -0.2095 & 0.1499 & 0.3490 & 1.7321 \end{pmatrix}$$

- $H$ has negative entries: difficult to intrepret ...

# Motivation for Non-Negative Matrix Factorization

- Due to negative entries in $H$, the first column of $A$ is approximated as:

  $a_1 \approx Ch_1$

  $= (0.2495, -0.1476, -0.1476, 0.4427, 0.2495, 0.2495, 0.6921, 0.4427, 0.2591, 0.4427)$

- This approximate document has negative entries for England and FIFA
- Need a way to keep the coordinates and basis positive!

# Another Algorithm for Clustering

Assuming that the approximate documents has positive entries, how to do clustering?

- Read out the component of approximate document along the centroid vector, say, for first document $a_1$

$$a_1 = H(1,1)C(:,1) + H(2,1)C(:,2)$$

- If $H(1,1)$ is the largest, then $a_1$ goes to 1st cluster, otherwise to 2nd cluster! Voila! You have new algorithm!
- For intuition why this works, see explaination done on board

# Non-Negative Matrix Factorization

- Having negative entries in $H$ creates difficulty in assigning cluster.
  - Need a non-negative matrix factorization!

Non-Negative Matrix Factorization: Given $A \in \mathbb{R}^{m \times n}$, want to compute rank-$k$ approximation that is constrained to have nonnegative factors. The optimization problem is:

$$\min_{W \geq 0, H \geq 0} \|A - WH\|_F, \quad W \in \mathbb{R}^{m \times k}, H \in \mathbb{R}^{k \times n}$$

- Optimization problem in both $W$ and $H$, hence non-linear!
- If either of $W$ or $H$ is known then it is usual linear least squares problem

# Alternating Least Squares

1. Guess an initial value of $W^{(1)}$
2. for $k = 1, 2, \ldots$ until convergence
   2.1 Solve $\min_{H \geq 0} \|A - W^{(k)}H\|_F$, giving $H^{(k)}$
   2.2 Solve $\min_{W \geq 0} \|A - WH^{(k)}\|_F$ giving $W^{(k+1)}$

- The factorization is not unique:
- One factor may grow, and other may decay! Need to column scale $W$

$$WH = (WD)(D^{-1}H)$$

# Algorithm: Non-Negative Matrix Factorization

```
while (not converged)
    [W] = normalize(W)
    for i=1:n
        H(:,i) = nonnegls(W,A(:,i)) %-ve in H made zero
    end
    for i=1:n
        w = nonnegls(H',A(:,i)') % -ve in W made zero
        W(i,:) = w';
    end
end
```

# Matrix Algorithms in Text Mining

Text Mining: Methods for extracting useful information from large and often unstructured collection of texts.

- For example, searching databases of abstracts in scientific papers.
- In a medical application one may want to find all the abstracts in the database that deal with a particular syndrome

We are interested in vector space model

# Processing of Texts

To create a vector space model:

- Create a term document matrix, where each document is represented by a column vector.
- The column has nonzero entries in the positions that correspond to terms that can be found in the document.
- Consequently, each row represents a term and has nonzero entries in those positions that correspond to the documents where the term can be found

# How to create a term document matrix?

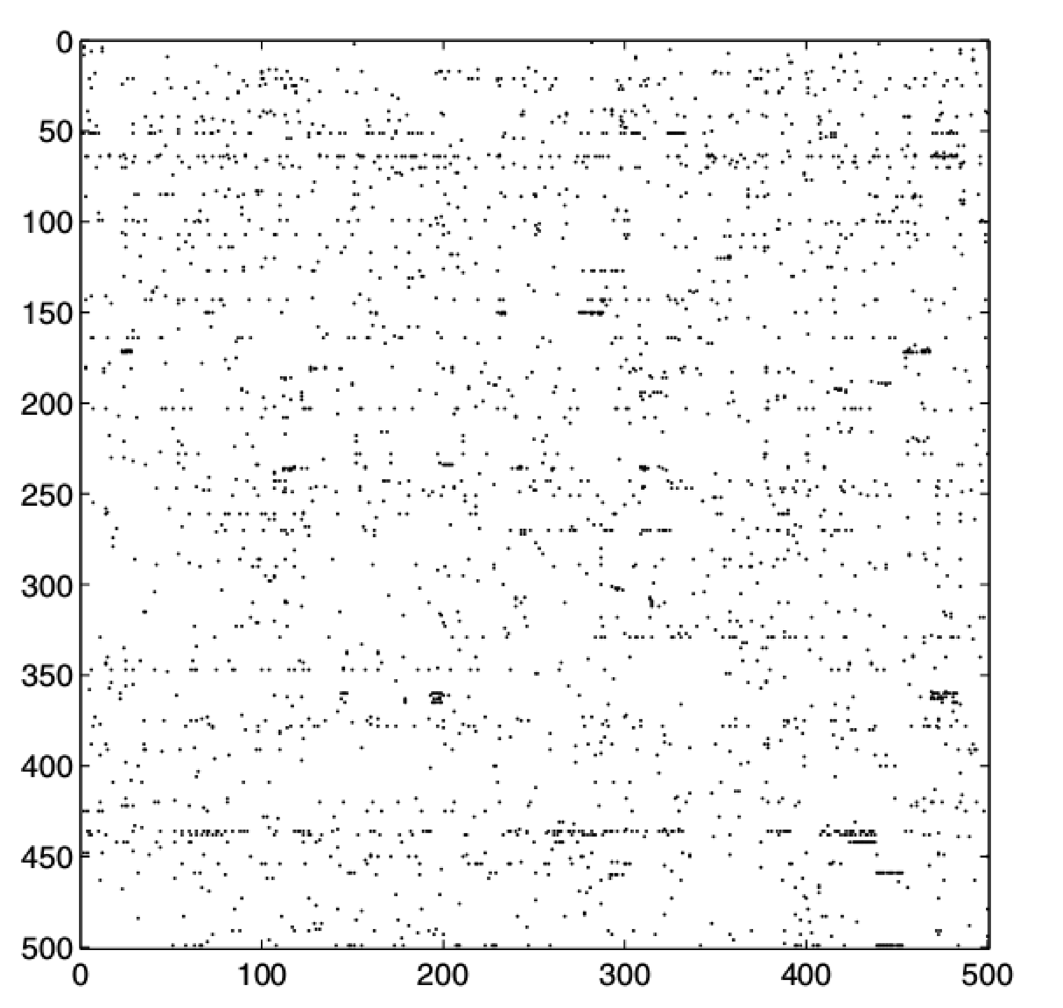
- Manual: In previous example, we created it manually
- Automated: For realistic, i.e., large size texts, one uses a text parser
  - Text parsers usually include both a stemmer and an option to remove stop words
  - Additionally, parsers have filters for reomving formatting code in the documents, e.g., HTML or XML.
  - It is common not only to count the occurrence of terms in documents but also to apply a term weighting scheme, where the elements of A are weighted depending on the characteristics of the document collection.
- For example, elements of the term document matrix $A(i,j)$ is defined by $a_{ij} = f_{ij} \log(n/n_i)$

# How to create a term document matrix?

- $f_{ij}$ is term frequency, the number of times term $i$ appears in document $j$
- $n_i$ is the number of documnets that contain term $i$
- If a term occurs frequently only in a few documents then both the factors are large
  - The term discriminates well between different groups of documents, and the log-factor in (11.1) gives it a large weight in the documents where it appears
- Normally, term document matrix is sparse

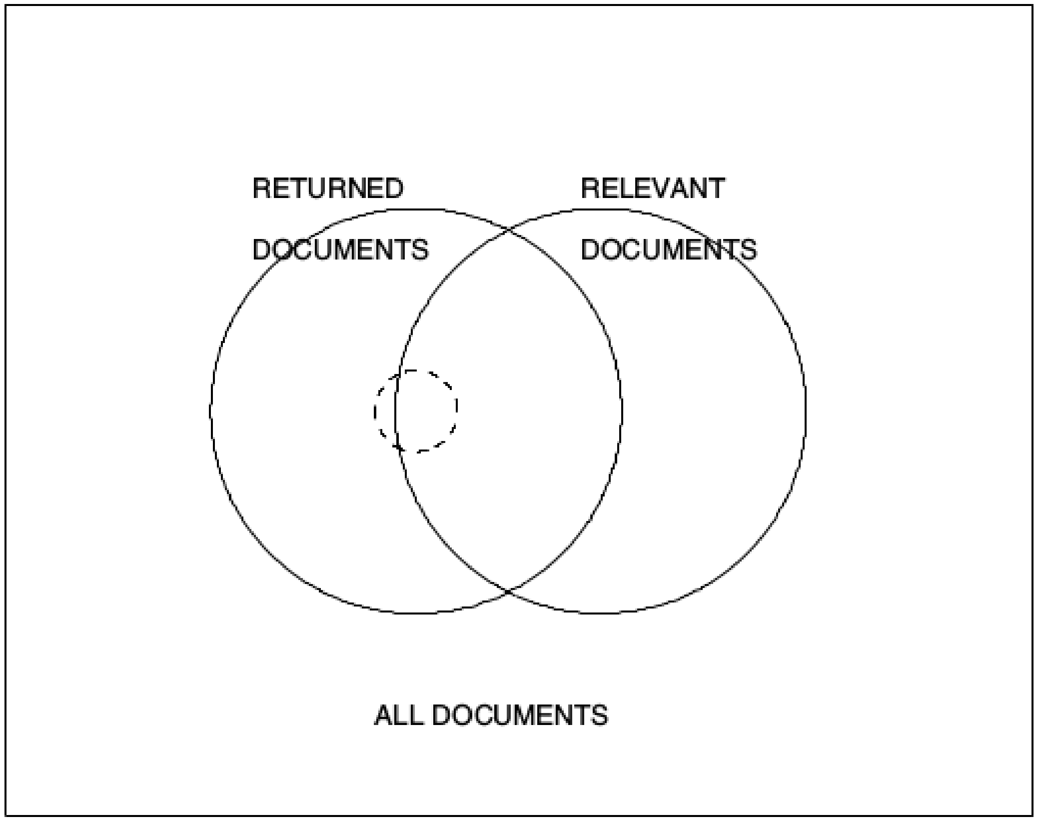
# Sparseness in term document matrix.

**Figure 1:** Medline Term Document Matrix. Each dot is non-zero



# When does query matching produces good result?

- The query matching produces a good result when the intersection between the two sets of returned and relevant documents is as large as possible and the number of returned irrelevant documents is small.

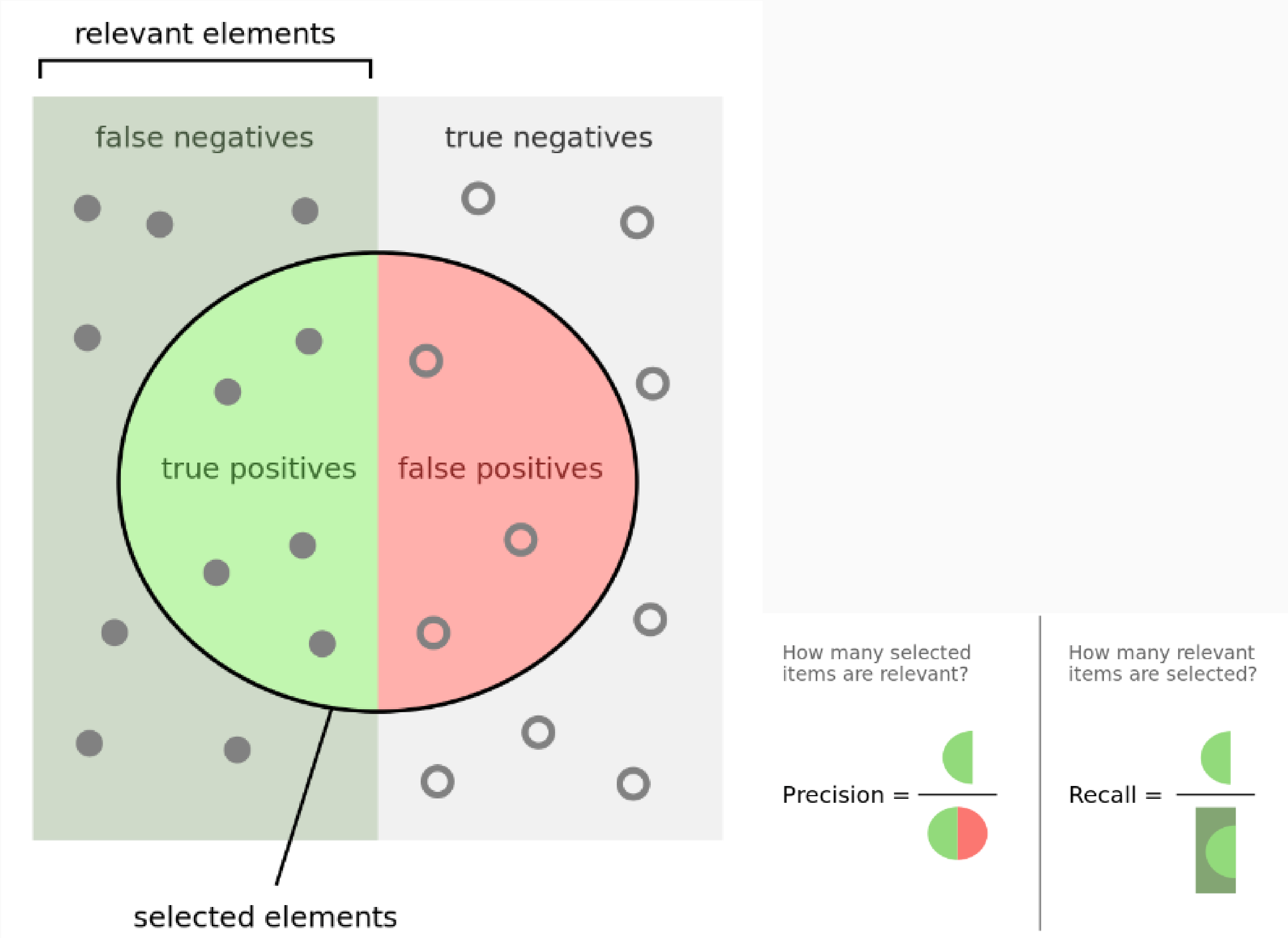


# Measures for Performance Modelling

Define the following measure:

1. Precision: $P = \dfrac{D_r}{D_t}$
   - $D_r$ is the number of relevant documents
   - $D_t$ is the total number of documents retrieved
2. Recall: $R = \dfrac{D_r}{N_r}$
   - $N_r$ is the number of relevant documents in the database

# Precision and Recall: Graphically



# Latent Semantic Indexing

- Based on the assumption that there is some underlying latent semantic structure in the data that is corrupted by the wide variety of words used
  - The semantic structure can be discovered and enhanced by projecting the data onto a low-dimensional space using SVD
- Let $A = U\Sigma V^T$ be the SVD of the temr document matrix
- Consider approximating $A$ by a matrix of low-rank $k$ :

$$A \approx U_k \Sigma_k V_k^T = U_k H_k$$

- Since $a_j \approx U_k h_j$, the column $j$ of $H_k$ holds the coordinates of document $j$ in terms of the orthogonal basis

# Rank-$k$ approximation and query matching

The rank $k$ approximation of term document matrix

$$A_k = U_k H_k$$

For query matching, we compute

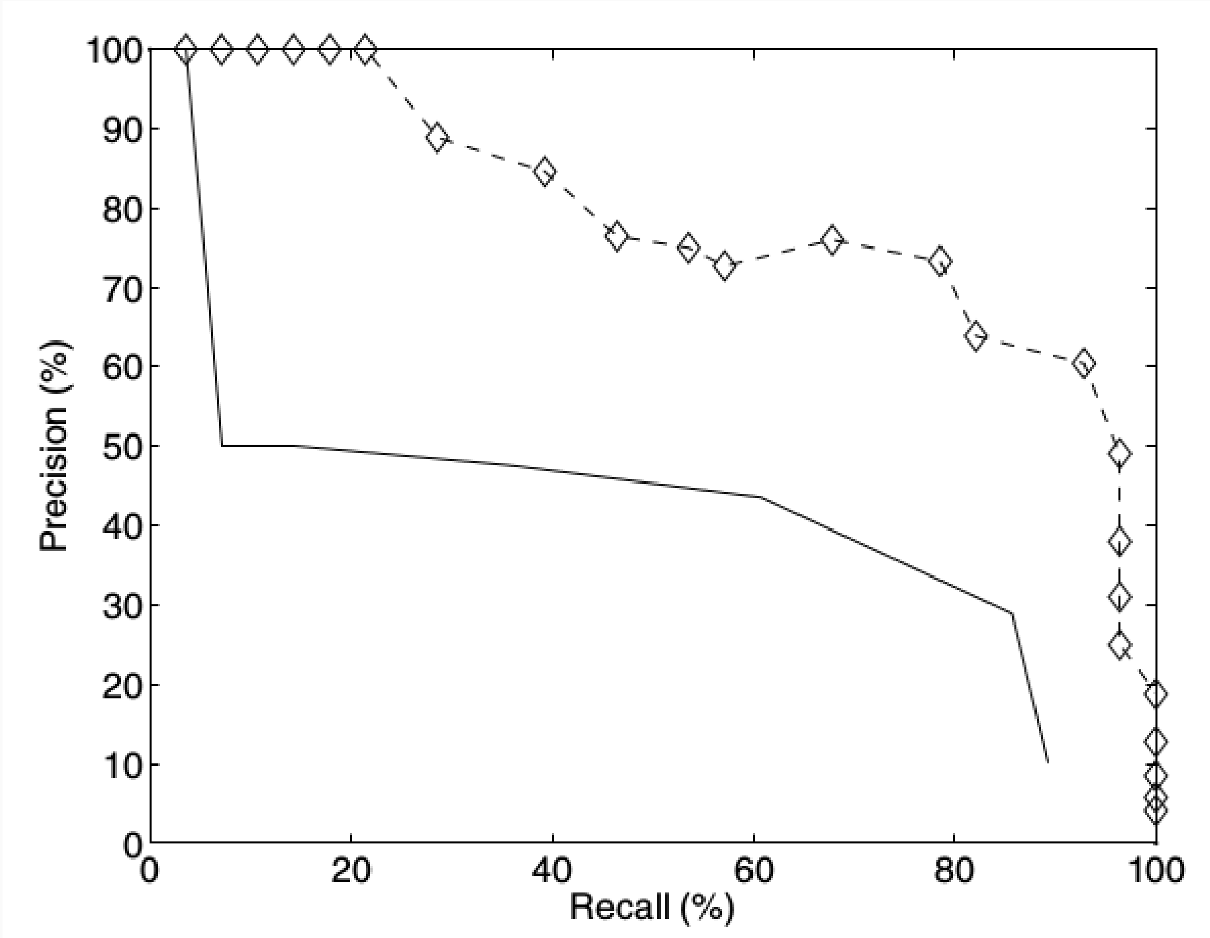
$$q^T A_k = q^T U_k H_k = (U_k^T q) H_k$$

In other words, we compute the coordinates of the query in terms of new document basis. The cosines are computed as:

$$\cos\theta_j = \frac{q_k^T h_j}{\|q_k\|_2 \|h_j\|_2}, \quad q_k = U_k^T q$$

- Query matching is performed in $k-$dimensional space

# Query Matching Using Latent Semantic Embedding




- We have better results when we project query only on significant directions $\Longrightarrow$ dimension reduction helps remove corrupt info

# Example of LSI

- Consider the term document matrix as before, and the query "**ranking** of **webpages**".
- The documents 1-4 are relevant to the query, while document 5 is irrelevant.
- However, we obtain the cosines for the query as:

$$(00.66670.77460.33330.3333)$$

  - Document 5 is as relevant to the query as document 4
  - Document 1 is orthogonal to the query: none of the words of the query occurs in document 1

# Effect of LSI

We now use LSI:

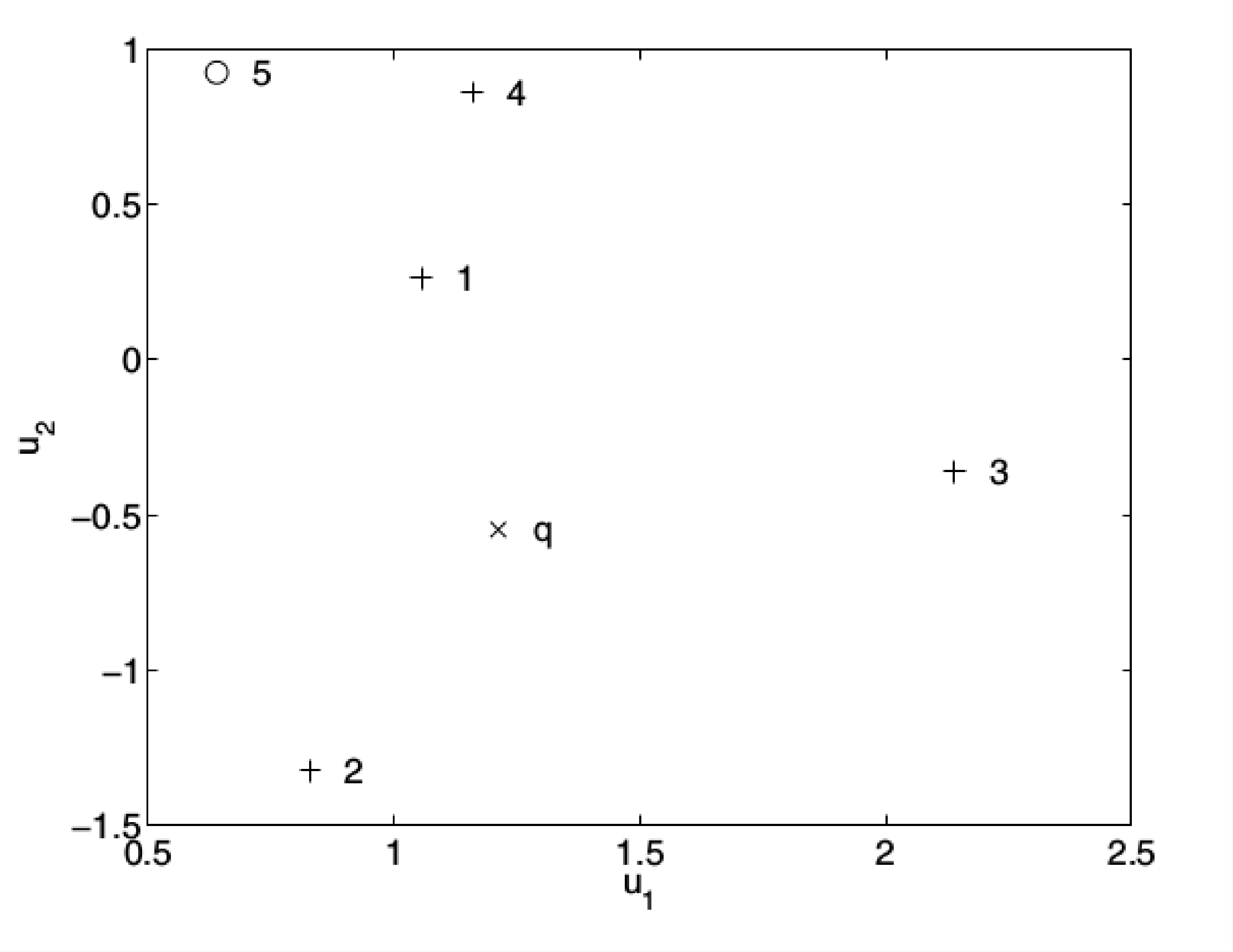
- Compute SVD of the term document matrix, and use rank-2 approximation
- Project queries to the 2-dimensional space
- Compute the cosines:

$$(0.78570.83320.96700.48730.1819)$$

  - Document 1 is now becomes highly relevant
  - Cosines for relevant documents 2-4 have been reinforced
  - Cosines for document 5 has been significantly reduced

Conclusion: In this example, dimension reduction enhanced the retrieval performance

# Plot Query




- The fifth document looks farthest!

Goal: Develop automatic procedure for text summarization from a vast amount of textual information

- Websearch engine (for example, G****e) presents a small amount of text from each document that matches a query
- Summarization of news articles

Automatic Text Summarization is related to:

- Information Retrieval
- Natural Language Processing
- Machine Learning

# Saliency Score

Consider the text of Chapter 12 of the book "Matrix Methods in Data Mining". Our goal is to extract key words and key sentences.

The preprocessing steps needed are:

- Perform stemming
- Remove Stop words
- Remove special symbols, e.g., matyhematics or mark-up languages such as HTML, LATEX, etc

# Term-Sentence Matrix

As before, we need to create a term document matrix using same type of parser as in information retrieval.
We have:

- Matrix $A \in \mathbb{R}^{m \times n}$
  - $m$ the number of different terms
  - $n$ the number of sentences
  - $a_{ij}$ frequency of the term $i$ in sentence $j$
  - Col vector $A(1:m, j)$ is non-zeros in positions corresp. to the terms occuring in sentence $j$
  - Row vector $A(i, 1:n)$ is non-zeros in positions corresp. to the sentences containing term $j$
  - We want to simultaneously rank term and sentences!

# Assignment of Salience Scores

Assignment of saliency scores is made based on the mutual reinforcement principle:
A term should have a high saliency score if it appears in many sentences with high saliency scores. A sentence should have a high saliency score if it contains many words with high saliency scores.

We assert that the saliency score of term $i$ is proportional to the sum of the scores of the sentences where it appears; in addition, each term is weighted by the corresponding matrix element:

$$u_i \propto \sum_{j=1}^{n} a_{ij} v_j, \quad i = 1, 2, \ldots, m$$

Similarly, the saliency score of sentence $j$ is defined to be proportional to the scores of its words, weighted by the corresponding $a_{ij}$ ,

$$v_j \propto \sum_{i=1}^{m} a_{ij} u_i, \quad j = 1, 2, \ldots, n$$

Collecting the saliency scores in two vectors:

$$\sigma_u u = Av, \quad \sigma_v v = A^T u$$

where $\sigma_u$ and $\sigma_v$ are proportionality constants.

- $\sigma_u$ and $\sigma_v$ must be same

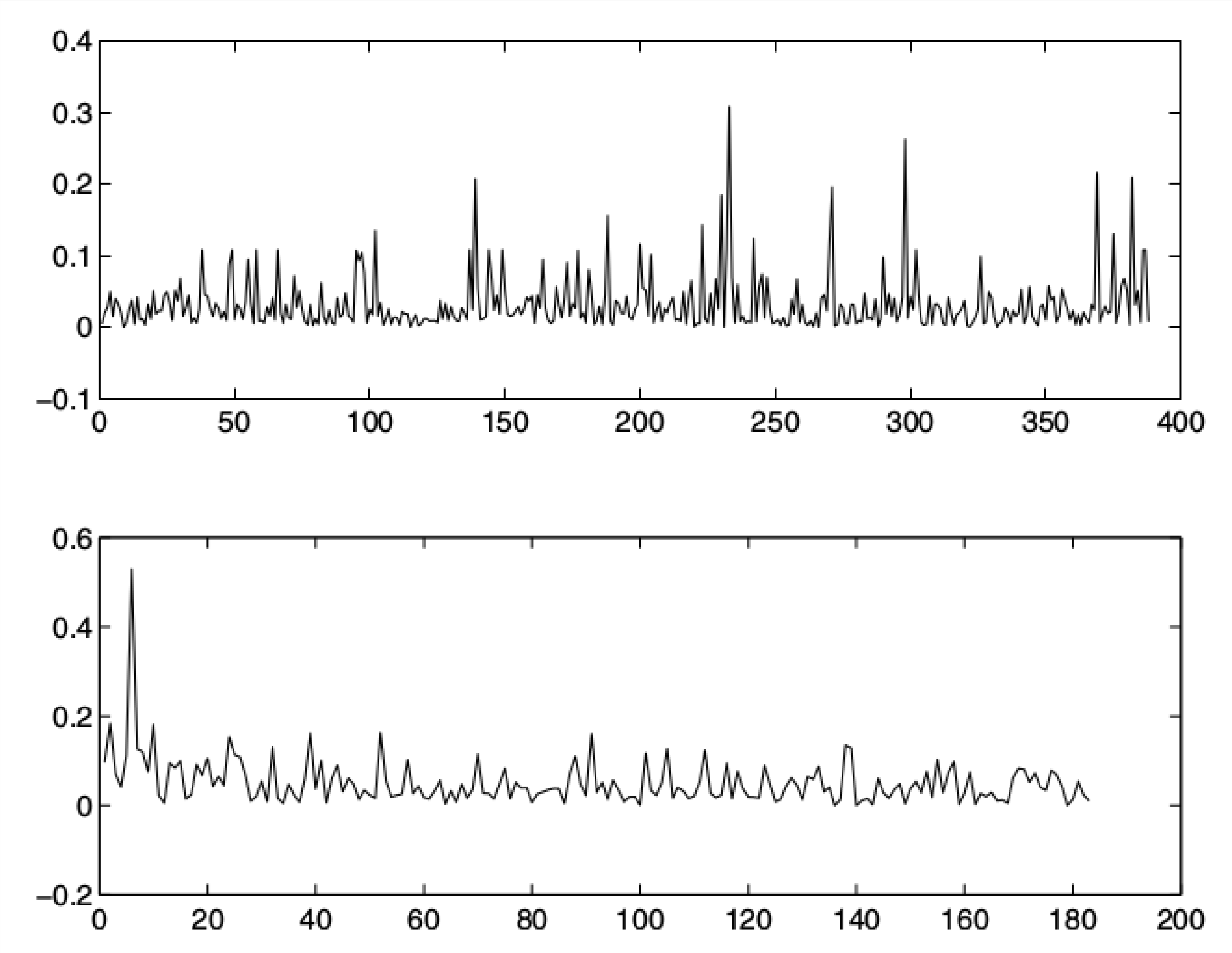
Inserting one into the other:

$$\sigma_u u = \frac{1}{\sigma_v} AA^T u$$
$$\sigma_v v = \frac{1}{\sigma_u} A^T A v$$

- $u$ and $v$ are eigenvectors of $AA^T$ and $A^T A$ respectively.

- Here $u$ and $v$ are singular vectors corresponding to the same singular values
- When do we guarantee that the components of $u$ and $v$ are nonnegative?
    - When we choose the largest singular value

# Saliency scores: Examples



- For term-sentence matrix based on Ch. 12: 388 terms in 183 sentences
- Compute first singluar matrix:

  ```
  [u,s,v] = svds(A,1)
  ```
- locate 10 largest components of $u_1$, the following words are most important: page, search, university, web, Google, rank, outlink, link, number, equal

Following are the six most important sentences:

- A Google search conducted on September 29, 2005, using the search phrase university, gave as a result links to the following well-known universities: Har- vard, Stanford, Cambridge, Yale, Cornell, Oxford.
- When a search is made on the Internet using a search engine, there is first a traditional text processing part, where the aim is to find all the Web pages containing the words of the query.
- Loosely speaking, Google assign a high rank to a Web page if it has inlinks from other pages that have a high rank.

- Assume that a surfer visiting a Web page chooses the next page from among the outlinks with equal probability.
- Similarly, column $j$ has nonzero elements equal to $N_j$ in those positions that correspond to the outlinks of $j$, and, provided that the page has outlinks, the sum of all the elements in column $j$ is equal to one.
- The random walk interpretation of the additional rank-1 term is that in each time step the surfer visiting a page will jump to a random page with probability $1\alpha$ (sometimes referred to as teleportation)

# Drawback of method based on Saliency Scores

If there are, say, two top sentences that contain the same high- saliency terms, then their coordinates will be approximately the same, and both sentences will be extracted as key sentences, which is unnecessary.

- It can be avoided if we base the key sentence extraction on rank-$k$ approximation.

# Key Sentence Extraction from Rank-$k$ Approximation

- Assume rank-$k$ approximation of term-sentence matrix:

$$A \approx CD, \quad C \in \mathbb{R}^{m \times k}, D \in \mathbb{R}^{k \times n}$$

  This can be done using either:
  - clustering, SVD, or non-negative factorization
- $k \geq$ number of key sentences to be extracted
- $C$ rank-$k$ matrix of basis vectors
- columns of $D$ holds the coordinates of corresponding cols in $A$ in terms of the basis vectors (cols of $C$)

# Key Sentence Extraction from Rank-$k$ Approx.

Low rank approximation does not immediately give most important sentences!

- Most important sentences are the ones for which the columns of $A$ is the heaviest in terms of the basis:
  - A column is heaviest if the norm of its coordinate in the basis of $C$ is largest!
- Idea: Find the column of $A$ which is heaviest, then find the column of $A$ which is next heaviest in terms of the remaining $k-1$ basis vectors, and so on

# Key Sentence Extraction from Rank-$k$ Approx.

We have

$$AP \approx CDP = (CT)(T^{-1}DP), \quad T \in \mathbb{R}^{k \times k}$$

Steps above are:

- find the column of largest norm in $D$
- permute it by $P_1$ to the first column
- similarly permute the corresponding column in $A$ to first column

# Key Sentence Extraction from Rank-$k$ Approx.

- determine Householder transformation $Q_1$ that zeros the elements in the first column below the $(1, 1)$ entry of $DP_1$
- Apply the transformation $Q_1$ to both $C$ and $D$ as follows

$$AP_1 \approx (CQ_1)(Q_1^T DP_1)$$

- Recall: this is the first step in QR with col pivoting

# Key Sentence Extraction

Consider an example with: $m = 6, n = 5, k = 3$.

How does all this help in identifying similar sentences?

To illustrate, let us assume that:

The 4th col of $D$ is similar to col that was moved to 1st col

After the first step, the matrices have the structure:

$$(CQ_1)(Q_1^T DP_1) = \begin{pmatrix} x & x & x \\ x & x & x \\ x & x & x \\ x & x & x \\ x & x & x \\ x & x & x \\ x & x & x \end{pmatrix} \begin{pmatrix} \kappa_1 & x & x & x & x \\ 0 & x & x & \epsilon_1 & x \\ 0 & x & x & \epsilon_2 & x \end{pmatrix}$$

# Key Sentence Extraction

$$(CQ_1)(Q_1^T DP_1) = \begin{pmatrix} x & x & x \\ x & x & x \\ x & x & x \\ x & x & x \\ x & x & x \\ x & x & x \\ x & x & x \end{pmatrix} \begin{pmatrix} \kappa_1 & x & x & x & x \\ 0 & x & x & \epsilon_1 & x \\ 0 & x & x & \epsilon_2 & x \end{pmatrix}$$

- $\kappa_1$ is the 1st col of $DP_1$
- since col 4 was similar to col 1 that is now 1st col, the entries $\epsilon_1$ and $\epsilon_2$ are small. (why?)

# Key Sentence Extraction

- Introduce the diagonal matrix

$$T_1 = \begin{pmatrix} \kappa_1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

between factors: $C_1 D_1 = (CQ_1 T_1)(T_1^{-1} Q_1^T DP_1)$

$$= \begin{pmatrix} * & x & x \\ * & x & x \\ * & x & x \\ * & x & x \\ * & x & x \\ * & x & x \\ * & x & x \end{pmatrix} \begin{pmatrix} 1 & x & x & x & x \\ 0 & x & x & \epsilon_1 & x \\ 0 & x & x & \epsilon_2 & x \end{pmatrix}$$

# Key Sentence Extraction

- Only changes: 1st col in left and 1st col in right factor
- Form the relation: $AP_1 \approx C_1 D_1$
  - first col in $AP_1$ is approx. equal to 1st col in $C_1$
  - Since the cols of $C$ are dominating directions, we have identified dominating col of $A$
- Observation: If one col of $D$ is similar to the first one, then it will have small entries below the first row, and it will not play a role in the selection of second most dominating col of $A$

# Key Sentence Extraction

- We determine the second most imp col of $A$ : As before, we compute the norms of cols of $D_1$, excluding the first row.
- col with the largest norm is moved to position 2
- the second col below the first row is now reduced by Householder: $C_2 D_2 = (C_1 Q_2 T_2)(T_2^{-1} Q_2^T D_1 P_2)$

$$= \begin{pmatrix} * & * & x \\ * & * & x \\ * & * & x \\ * & * & x \\ * & * & x \\ * & * & x \\ * & * & x \end{pmatrix} \begin{pmatrix} 1 & x & x & x & x \\ 0 & 1 & * & \epsilon_1 & * \\ 0 & x & x & \epsilon_2 & x \end{pmatrix}$$

# Key Sentence Extraction

- Second col of $AP_1P_2 \approx C_2D_2$ holds the second most dominating column
- Continuing $AP \approx C_kD_k, \quad D_k(R \quad S)$,
  - $R$ is upp. triang.
  - $P$ product of permutations
  - first $k$ cols of $AP$ hold the dominating cols
  - rank $k$ approx is in $C_k$

We have

$$AP \approx C_kRR^{-1}D_k = \hat{C}(I, \hat{S})$$

where, $\hat{C} = C_kR$ and $\hat{S} = R^{-1}S$.

# Key Sentence Extraction

Since,

$$\hat{C} = CQ_1T_1Q_2T_2 \cdots Q_kT_kR$$

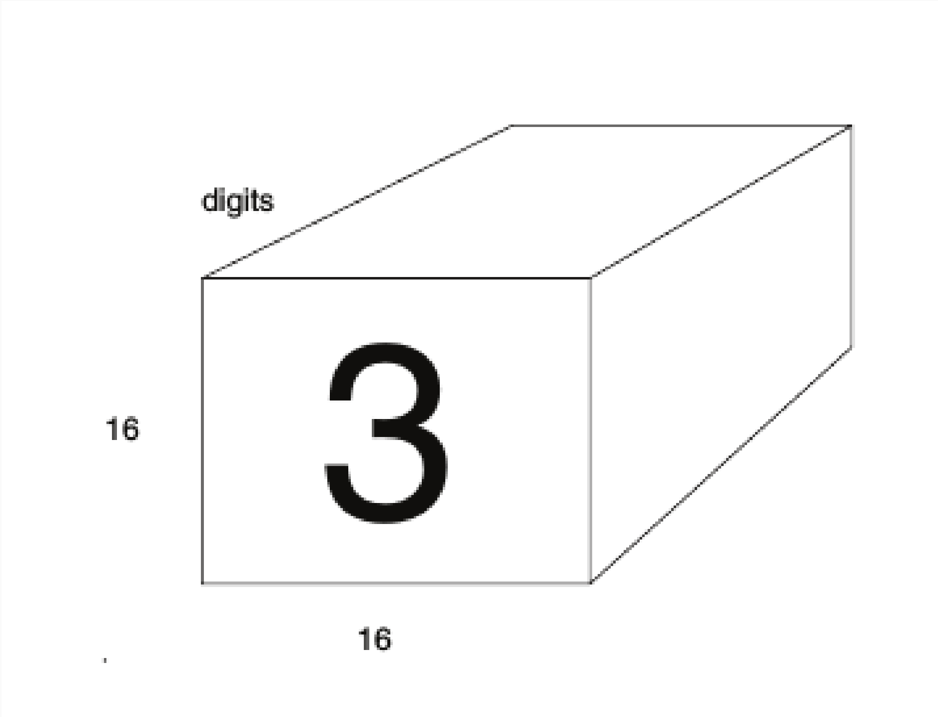
- $\hat{C}$ is rotated and scaled version of $C$
- hence, $\text{span}(\hat{C}) = \text{span}(C)$
- It still holds the dominating directions of $A$
- Note: If we were interested only in top $k$ sentences, then we need not had to apply transformations described above!
    - But in that case without above transformation, you may report very similar sentences many times

# Introduction to Tensor Decompositions

- We have seen: vectors and matrices which can be seen as one-dim or two dim arrays of data
  - For example in a term-document matrix, every element is associated with one term and one document
- In many apps, data is organized in more than two categories leading to tensors
- Multilinear algebra: area of mathematics dealing with tensors
- For simplicity, we restrict to: $\mathcal{A} \in \mathbb{R}^{\ell \times m \times n}$, i.e., array of data with three subscripts

# Example of Tensor

- Each digit is $16 \times 16$ matrix of pixels on grey scale, then a set of $n$ digits is organized as tensor: $\mathbb{R}^{16 \times 16 \times n}$
- Refer to $\mathcal{A} \in \mathbb{R}^{\ell \times m \times n}$ as three mode array
  - different dimensions of the array are called modes
- The dimensions of the array $\mathcal{A} \in \mathbb{R}^{\ell \times m \times n}$ are $\ell$, $m$,and $n$.
- For example, a matrix is called 2-mode array



# Basic Tensor Concepts: inner product and norm

- Define inner product of two tensors:

$$\langle A, B \rangle = \sum_{i,j,k} a_{ijk} b_{ijk}$$

- The norm is defined as

$$\|\mathcal{A}\|_F = \langle A, A \rangle_F = \left( \sum_{i,j,k} a_{ijk}^2 \right)^{1/2}$$

- For matrices, this is called Frobenius norm

# Basic Tensor Concepts: $i$ mode multiplication

1-mode multiplication of tensor by a matrix

$$(\mathcal{A} \times_1 U)(j, i_2, i_3) = \sum_{k=1}^{\ell} u_{j,k} a_{k,i_2,i_3}$$

$$A \times_1 U = UA$$

2-mode multiplication of tensor by a matrix

$$(\mathcal{A} \times_2 U)(i_1, j, i_3) = \sum_{k=1}^{\ell} v_{j,k} a_{i_1,k,i_3}$$

$$A \times_1 U = AV^T$$

3-mode is analogous

# Tensor Folding and Unfolding

Sometimes convenient to unfold a Tensor into a matrix. The unfolding of tensor $\mathcal{A}$ along the three modes:

$$
\begin{aligned}
\mathbb{R}^{\ell\times mn} &\ni \text{unfold}_1(\mathcal{A}) := A_{(1)} := \begin{pmatrix}\mathcal{A}(:,1,:) & \mathcal{A}(:,2,:) \cdots & \mathcal{A}(:,m,:)\end{pmatrix},\\
\mathbb{R}^{mn\times \ell n} &\ni \text{unfold}_2(\mathcal{A}) := A_{(2)} := \begin{pmatrix}\mathcal{A}(:,:,1)^T & \mathcal{A}(:,:,2)^T \cdots & \mathcal{A}(:,:,n)^T\end{pmatrix}\\
\mathbb{R}^{n\times \ell m} &\ni \text{unfold}_3(\mathcal{A}) := A_{(3)} := \begin{pmatrix}\mathcal{A}(1,:,:)^T & \mathcal{A}(2,:,:)^T \cdots & \mathcal{A}(\ell,:,:)^T\end{pmatrix}
\end{aligned}
$$

# Tensor Folding and Unfolding

Let $\mathcal{B} \in \mathbb{R}^{3\times3\times3}$ be a tensor defined as:

```
B(:,:,1) =                 B(:,:,2) =                 B(:,:,3) =
     1     2     3              11    12    13              21    22    23
     4     5     6              14    15    16              24    25    26
     7     8     9              17    18    19              27    28    29
```

The unfolding along the third mode gives:

```
>> B3 = unfold(B,3)

b3 =      1     2     3     4     5     6     7     8     9
         11    12    13    14    15    16    17    18    19
         21    22    23    24    25    26    27    28    29
```

The inverse of the unfolding operation is written:

$$\text{fold}_i(\text{unfold}_i(\mathcal{A})) = \mathcal{A}$$

# Tensor Folding and Unfolding

Note: For the folding operation to be well-defined, the information about target tensor must be specified

Using folding-unfolding, the $i$ mode tensor-matrix multiplication is defined as

$$\mathcal{A} \times_i U = \text{fold}_i(U \, \text{unfold}_i(\mathcal{A})) = \text{fold}_i(UA_{(i)})$$

# Tensor SVD

The matrix SVD can be generalized to tensors in many ways.
We show one possible, called HOSVD

**Theorem 8.3 (HOSVD).** *The tensor $\mathcal{A} \in \mathbb{R}^{l \times m \times n}$ can be written as*

$$\mathcal{A} = \mathcal{S} \times_1 U^{(1)} \times_2 U^{(2)} \times_3 U^{(3)}, \tag{8.6}$$

*where $U^{(1)} \in \mathbb{R}^{l \times l}$, $U^{(2)} \in \mathbb{R}^{m \times m}$, and $U^{(3)} \in \mathbb{R}^{n \times n}$ are orthogonal matrices. $\mathcal{S}$ is a tensor of the same dimensions as $\mathcal{A}$; it has the property of* all-orthogonality*: any two slices of $\mathcal{S}$ are orthogonal in the sense of the scalar product* (8.1)*:*

$$\langle \mathcal{S}(i,:,:), \mathcal{S}(j,:,:) \rangle = \langle \mathcal{S}(:,i,:), \mathcal{S}(:,j,:) \rangle = \langle \mathcal{S}(:,:,i), \mathcal{S}(:,:,j) \rangle = 0$$

*for $i \neq j$. The* 1-mode singular values *are defined by*

$$\sigma_j^{(1)} = \|\mathcal{S}(i,:,:)\|_F, \qquad j = 1, \ldots, l,$$

*and they are ordered*

$$\sigma_1^{(1)} \geq \sigma_2^{(1)} \geq \cdots \geq \sigma_l^{(1)}. \tag{8.7}$$

*The singular values in other modes and their ordering are analogous.*

# Algorithm to compute HOSVD

***Proof.*** We give only the recipe for computing the orthogonal factors and the tensor $\mathcal{S}$; for a full proof, see [60]. Compute the SVDs,

$$A_{(i)} = U^{(i)}\Sigma^{(i)}(V^{(i)})^T, \quad i = 1, 2, 3, \tag{8.8}$$

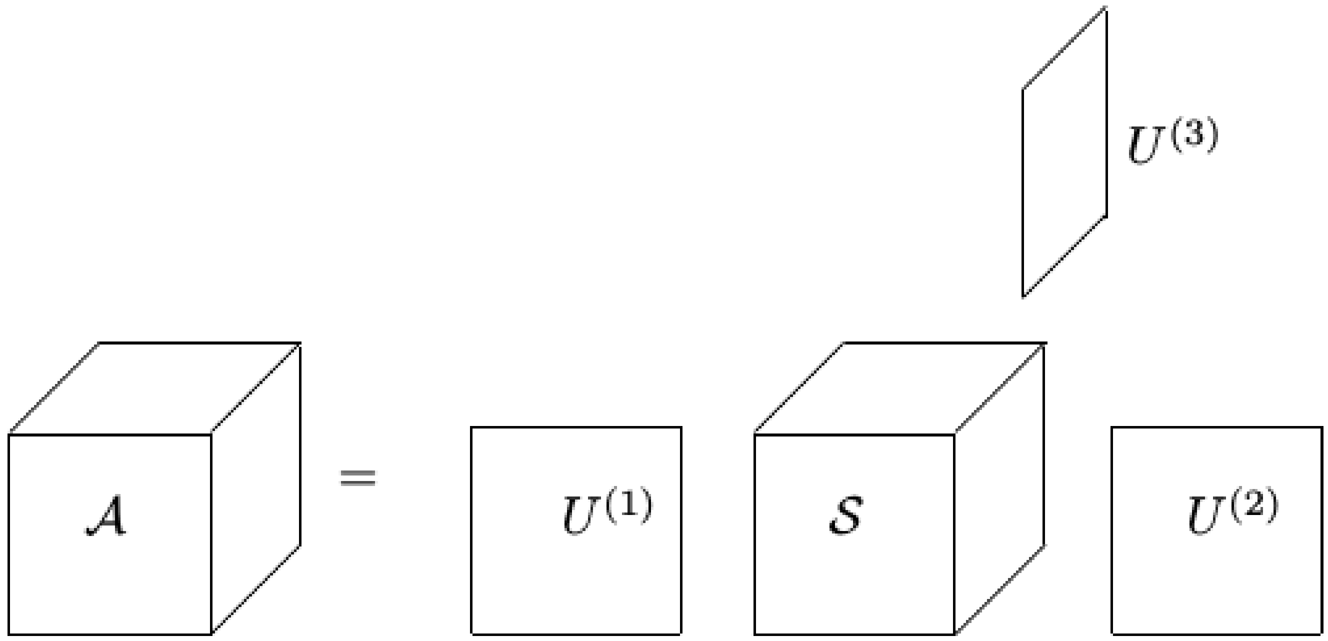
and put

$$\mathcal{S} = \mathcal{A} \times_1 (U^{(1)})^T \times_2 (U^{(2)})^T \times_3 (U^{(3)})^T.$$

It remains to show that the slices of $\mathcal{S}$ are orthogonal and that the $i$-mode singular values are decreasingly ordered. $\square$

# Picture of HOSVD

The all-orthogonal tensor $\mathcal{S}$ is usually referred to as the *core tensor*. The HOSVD is visualized in Figure 8.2.



# Computation of HOSVD in MATLAB

The computation of the HOSVD is straightforward and is implemented by the following MATLAB code, although somewhat inefficiently:[16]

```
function [U1,U2,U3,S,s1,s2,s3]=svd3(A);
% Compute the HOSVD of a 3-way tensor A

[U1,s1,v]=svd(unfold(A,1));
[U2,s2,v]=svd(unfold(A,2));
[U3,s3,v]=svd(unfold(A,3));

S=tmul(tmul(tmul(A,U1',1),U2',2),U3',3);
```

The function `tmul(A,X,i)` is assumed to multiply the tensor `A` by the matrix `X` in mode `i`, $\mathcal{A} \times_i X$.

# Tensor Compression

A tensor $\mathcal{A} \in \mathbb{R}^{\ell \times m \times n}$ can be expressed as a sum of matrix times singular vectors:

$$\mathcal{A} = \sum_{i=1}^{n} A_i \times_3 u_i^{(3)}, \quad A_i = S(:,:,i) \times_1 U^{(1)} \times_2 U^{(2)}$$

where $u_i^{(3)}$ are column vectors in $U^{(3)}$. The tensor compression is obtained by:

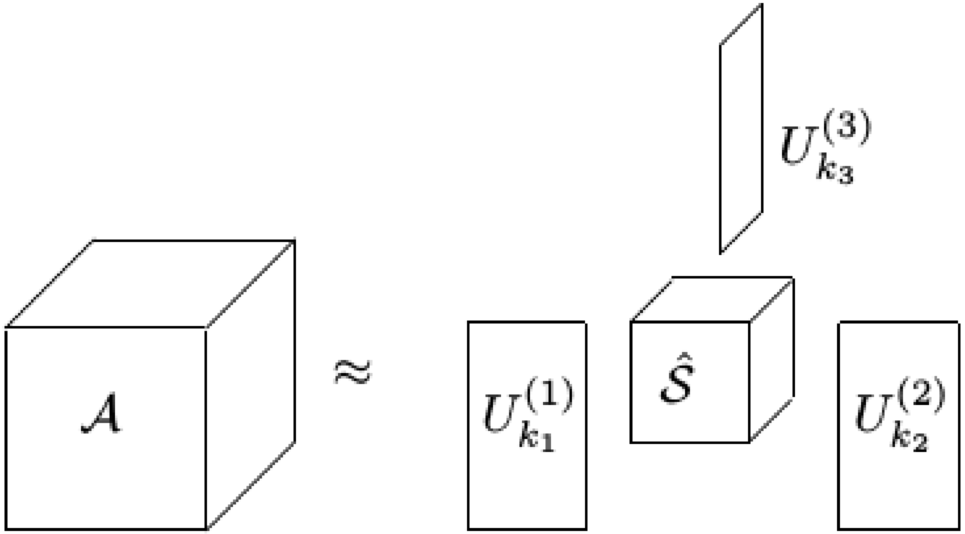
$$\mathcal{A} = \sum_{i=1}^{k} A_i \times_3 U_i^{(3)}$$

Note: This is a $k$-approximation of the tensor in 3rd mode

# Tensor Compression: another approach

$$\mathcal{A} \approx \hat{\mathcal{A}} = \hat{\mathcal{S}} \times_1 U_{k_1}^{(i)} \times_2 U_{k_2}^{(2)} \times_3 U_{k_3}^{(3)}.$$

We illustrate this as follows:




where

- $U_{k_i}^{(i)} = U^{(i)}(:, 1 : k_i)$. Here $U^{(i)}$ compressed to $k_i$
- $\hat{S} = S(1 : k_1, 1 : k_2, 1 : k_3)$