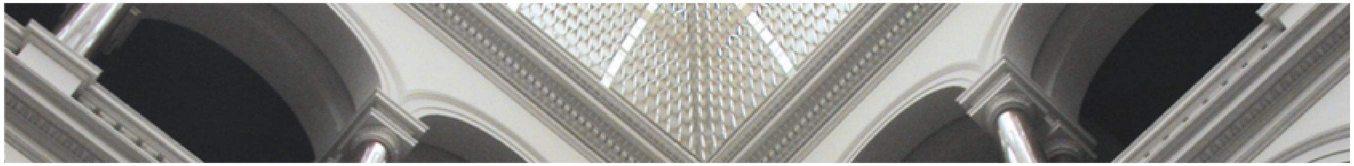
Dimensionality Reduction

Sebastian Schelter | Database Group, TU Berlin | Scalable Data Science: Systems and Methods

Overview

- Representing data as matrices
- Intuition
- Applications
  - Latent Semantic Indexing
  - Estimating the number of triangles in a network
- Singular Value Decomposition at Scale
- Summary



Overview

- **Representing data as matrices**

- Intuition

- Applications
  - Latent Semantic Indexing
  - Estimating the number of triangles in a network

- Singular Value Decomposition at Scale

- Summary

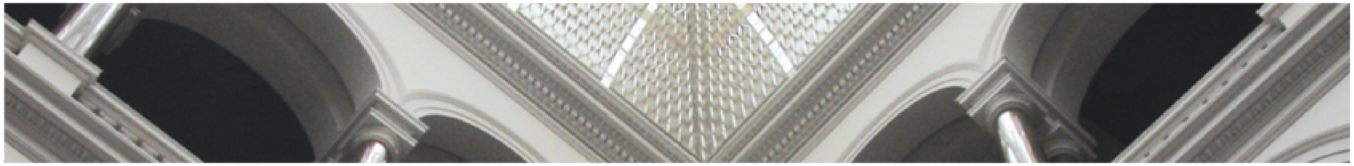
Representing data as matrices

How would you represent the following types of data as a matrix ?

What would be the rows and columns of the matrix?

What would be the contents of a cell?

- **a collection of text documents**
- **a large network**
- **a set of movie ratings**



Representing data as matrices

	idea	rows	columns	cells
text documents	'bag of words'	documents	terms	term occurrence, term frequency, tf-idf scores
network	adjaceny matrix	vertices	vertices	edge occurrences, edge weights
movie ratings	interaction matrix	users	items	occurrence of interaction, rating

- resulting matrices are **high-dimensional** and **extremely sparse** in many cases (problematic!)
- **dimensionality reduction:** reduce data to the **“interesting”** dimensions

Overview

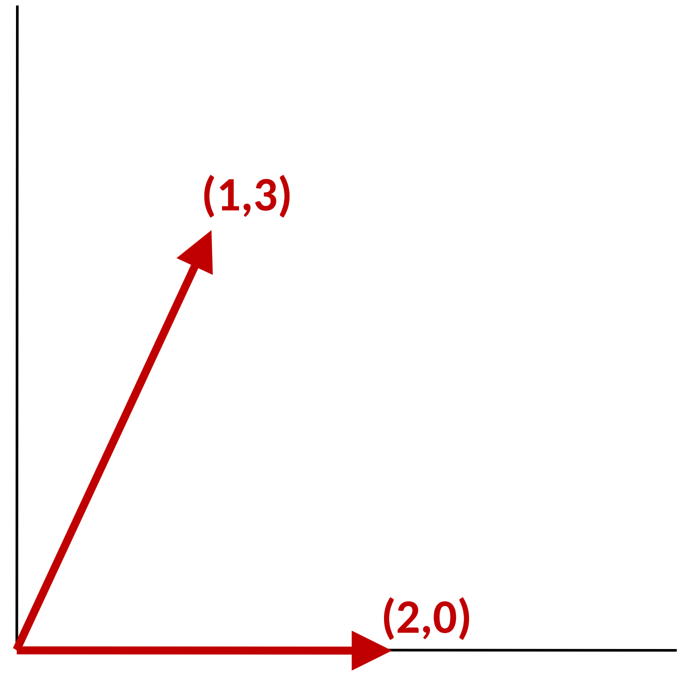
- Representing data as matrices
- **Intuition**
- Applications
  - Latent Semantic Indexing
  - Estimating the number of triangles in a network
- Singular Value Decomposition at Scale
- Summary

Intuition

$$\begin{bmatrix} 2 & 0 \\ 1 & 3 \end{bmatrix}$$

Intuition

$$\begin{bmatrix} 2 & 0 \\ 1 & 3 \end{bmatrix}$$

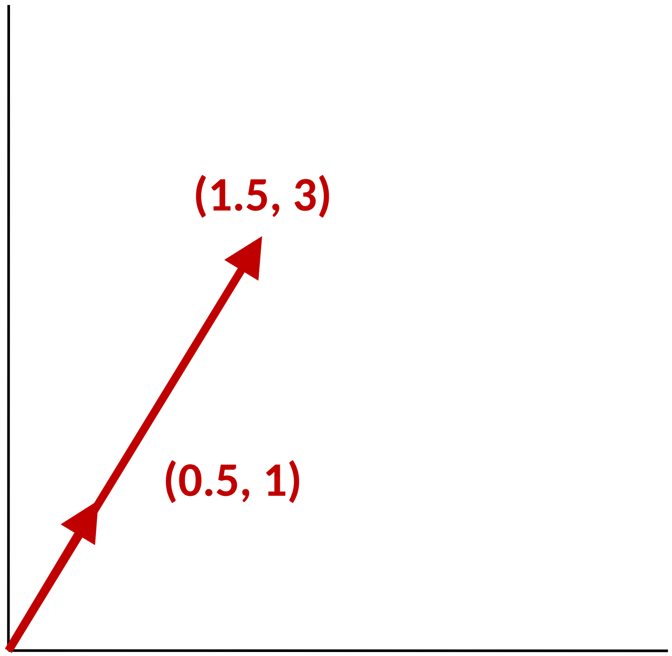


Intuition

$$\begin{bmatrix} 0.5 & 1 \\ 1.5 & 3 \end{bmatrix}$$

Intuition

$$\begin{bmatrix} 0.5 & 1 \\ 1.5 & 3 \end{bmatrix}$$

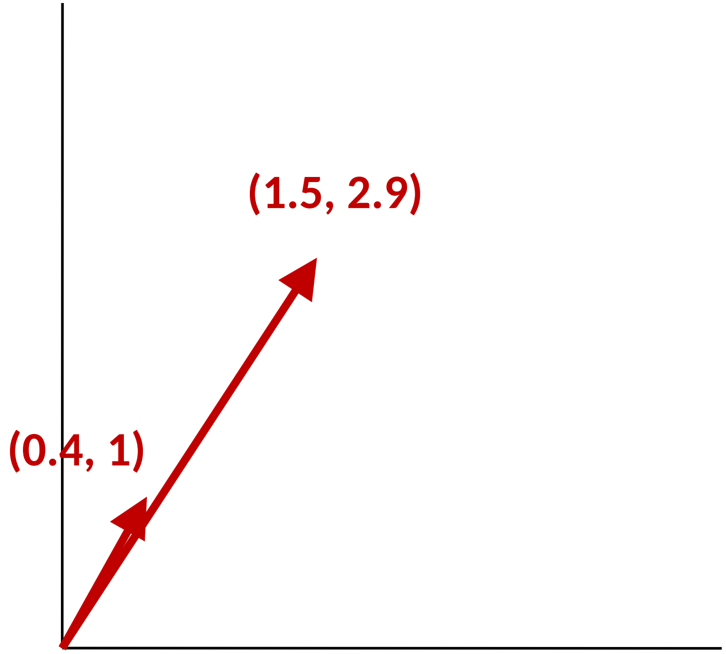


Intuition

$$\begin{bmatrix} 0.6 & 1 \\ 1.5 & 2.9 \end{bmatrix}$$

Intuition

$$\begin{bmatrix} 0.4 & 1 \\ 1.5 & 2.9 \end{bmatrix}$$



Overview

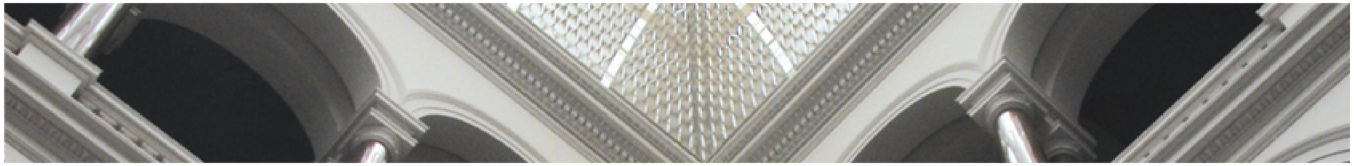
- Representing data as matrices
- Intuition
- Applications
  - **Latent Semantic Indexing**
  - Estimating the number of triangles in a network
- Singular Value Decomposition at Scale
- Summary

Latent Semantic Indexing

- quick **review of search engines**:
  - documents represented as collection of terms
  - search engines operate on **“inverted index”** from terms to documents
  - search: lookup documents for terms contained in query from inverted index

  - mathematically: documents and queries represent vectors in high-dimensional term space (**“vector space model”**)
  - searching means finding the closest document vectors to the query vector

- main **drawback**
  - relies on **lexical matches**, unable to identify synonyms and conceptually close terms



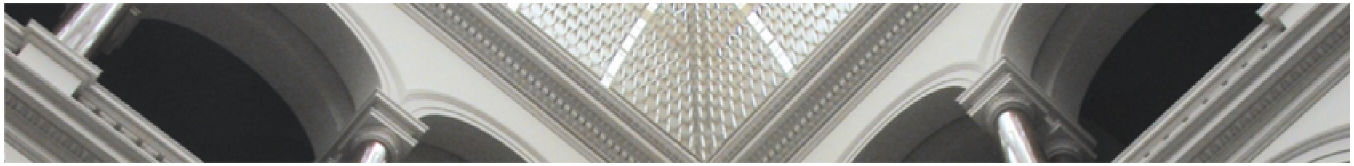
Lexical matching

- imagine a **corpus** with the following three documents

  - document 1: *“bike”*
  - document 2: *“bike harley”*
  - document 3: *“berlin”*

- a query for *“harley”* in a search engine which uses lexical matching only returns document 2
  - yet, document 1 might be relevant as well!

- can we build a search engine that is “smart” enough to also return document 1?



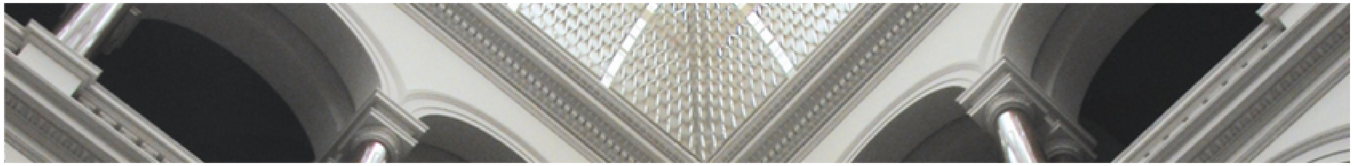
Manual solution: query expansion

- create custom **taxonomies of weighted relations between terms**

  e.g. „harley → bike 0.5“

- **automatically expand queries**
  - query “harley“ becomes „harley bike^0.5“

- **drawbacks**
  - crafting these lists is a lot of work, as they are domain-dependent!
  - might lead to very long queries (expensive!)
  - result quality is hard to predict



Towards Latent Semantic Indexing

- intuition: **structure contained in the corpus** describing relations between terms and documents

- assume **terms and documents** belong to **“latent” concepts**, then:

  - a single term describing a particular concept will occur in documents about that concept
  - terms describing the same concept co-occur in documents about that concept
  - documents about a particular concept share a set of characteristic terms

Technische Universität Berlin

The Linear Algebra view of search

- simplified model:
  - **corpus** is **represented as** *document x term* **matrix**
  - a cell *m,n* is 1 if document *m* contains term *n* and 0 otherwise

$$A = \begin{array}{cccc} & \textit{bike} & \textit{harley} & \textit{berlin} \\ \textit{doc1} & 1 & 0 & 0 \\ \textit{doc2} & 1 & 1 & 0 \\ \textit{doc3} & 0 & 0 & 1 \end{array}$$

- **queries** „harley“ and „harley bike“ **are just vectors in the term space** (analogous to documents)

$$q_1 = \begin{array}{ccc} \textit{bike} & \textit{harley} & \textit{berlin} \\ 0 & 1 & 0 \end{array} \qquad q_2 = \begin{array}{ccc} \textit{bike} & \textit{harley} & \textit{berlin} \\ 1 & 1 & 0 \end{array}$$

Search as matrix-vector multiplication

- use the **number of shared terms as similarity measure** between queries and documents
  → **search becomes matrix-vector multiplication**

$$A\,q^T$$

- examples: search for „harley“ and „harley bike“ in corpus

  doc1 : „bike“
  doc2: „harley bike“
  doc3: „berlin“

Search as matrix-vector multiplication

- example: search for *"harley"*

$$
A = \begin{array}{c|ccc} & \textbf{bike} & \textbf{harley} & \textbf{berlin} \\ \hline \textbf{doc1} & 1 & 0 & 0 \\ \textbf{doc2} & 1 & 1 & 0 \\ \textbf{doc3} & 0 & 0 & 1 \end{array}
\qquad
q_1 = \begin{array}{ccc} bike & harley & berlin \\ 0 & 1 & 0 \end{array}
$$

$$
A\,q_1^T = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}
$$

Search as matrix-vector multiplication

- example: search for *“harley”*

$$A = \begin{array}{c|ccc} & \textbf{bike} & \textbf{harley} & \textbf{berlin} \\ \hline \textbf{doc1} & 1 & 0 & 0 \\ \textbf{doc2} & 1 & 1 & 0 \\ \textbf{doc3} & 0 & 0 & 1 \end{array}$$

$$q_1 = \begin{array}{ccc} \textit{bike} & \textit{harley} & \textit{berlin} \\ 0 & 1 & 0 \end{array}$$

$$A\,q_1^T = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

Search as matrix-vector multiplication

- example: search for *"harley"*

$$
A = \begin{array}{cccc} & \textbf{bike} & \textbf{harley} & \textbf{berlin} \\ \textbf{doc1} & \textbf{1} & \textbf{0} & \textbf{0} \\ \textbf{doc2} & \textbf{1} & \textbf{1} & \textbf{0} \\ \textbf{doc3} & \textbf{0} & \textbf{0} & \textbf{1} \end{array}
\qquad
q_1 = \begin{array}{ccc} \textit{bike} & \textit{harley} & \textit{berlin} \\ 0 & 1 & 0 \end{array}
$$

$$
A\, q_1^T = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}
\begin{array}{l} \text{\# lexical matches in doc1} \\ \text{\# lexical matches in doc2} \\ \text{\# lexical matches in doc3} \end{array}
$$

Search as matrix-vector multiplication

- example: search for *“bike harley”*

$$
A = \begin{array}{c|ccc} & \textbf{bike} & \textbf{harley} & \textbf{berlin} \\ \hline \textbf{doc1} & 1 & 0 & 0 \\ \textbf{doc2} & 1 & 1 & 0 \\ \textbf{doc3} & 0 & 0 & 1 \end{array}
\qquad
\begin{array}{cccc} & \textit{bike} & \textit{harley} & \textit{berlin} \\ q_2 = & 1 & 1 & 0 \end{array}
$$

$$
A\, q_2^T = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}
$$

Search as matrix-vector multiplication

- example: search for *“bike harley”*

$$A = \begin{array}{cccc} & \textbf{bike} & \textbf{harley} & \textbf{berlin} \\ \textbf{doc1} & 1 & 0 & 0 \\ \textbf{doc2} & 1 & 1 & 0 \\ \textbf{doc3} & 0 & 0 & 1 \end{array} \qquad q_2 = \begin{array}{ccc} \textit{bike} & \textit{harley} & \textit{berlin} \\ 1 & 1 & 0 \end{array}$$

$$A\,q_2^T = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix}$$

Search as matrix-vector multiplication

- example: search for *"bike harley"*

$$A = \begin{array}{cccc} & \textbf{bike} & \textbf{harley} & \textbf{berlin} \\ \textbf{doc1} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \textbf{doc2} & \mathbf{1} & \mathbf{1} & \mathbf{0} \\ \textbf{doc3} & \mathbf{0} & \mathbf{0} & \mathbf{1} \end{array} \qquad \begin{array}{cccc} & \boldsymbol{bike} & \boldsymbol{harley} & \boldsymbol{berlin} \\ \boldsymbol{q}_2 = & \mathbf{1} & \mathbf{1} & \mathbf{0} \end{array}$$

$$\boldsymbol{A\,q_2^T} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{1} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} \begin{bmatrix} \mathbf{1} \\ \mathbf{1} \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} \mathbf{1} \\ \mathbf{2} \\ \mathbf{0} \end{bmatrix} \begin{array}{l} \text{\# lexical matches in doc1} \\ \text{\# lexical matches in doc2} \\ \text{\# lexical matches in doc3} \end{array}$$

Exploring our corpus with Linear Algebra

- vector space model of information retrieval
  - a query is just a vector in term space, analogous to a document
  - we can compute the similarity of this vector to all documents

- **how does this help with finding „latent concepts“ ?**

Exploring our corpus with Linear Algebra

- vector space model of information retrieval
  - a query is just a vector in term space, analogous to a document
  - we can compute the similarity of this vector to all documents

- **how does this help with finding „latent concepts“ ?**
  → we can compute similarities between documents!

Exploring our corpus with Linear Algebra

- computing AA^T gives a matrix of document similarities
- cell $A_{m,n}$ holds the number of terms shared between documents m and n

  → documents 1 and 2 are similar!

$$AA^T = \begin{array}{cccc} & doc1 & doc2 & doc3 \\ doc1 & 1 & 1 & 0 \\ doc2 & 1 & 2 & 0 \\ doc3 & 0 & 0 & 1 \end{array}$$

Exploring our corpus with Linear Algebra

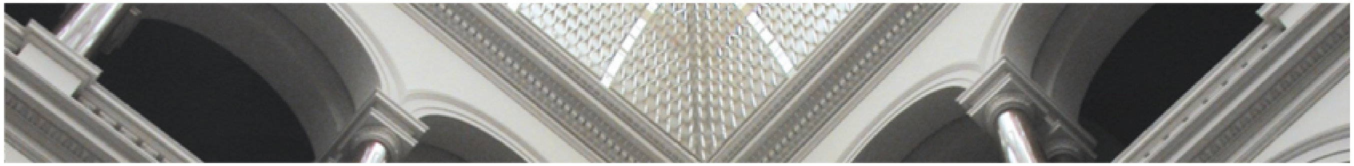
- computing A^TA gives a matrix of term co-occurrences
- cell $A_{m,n}$ holds the number of documents in which terms *m* and *n* occur together
  → *"harley"* and *"bike"* are related!

$$A^T A = \begin{array}{cccc} & bike & harley & berlin \\ bike & 2 & 1 & 0 \\ harley & 1 & 1 & 0 \\ berlin & 0 & 0 & 1 \end{array}$$

Singular Value Decomposition (SVD)

- **Singular Value Decomposition** of a real *m x n* matrix A:
  - U (*m x m*) and V (*n x n*) are orthogonal, ∑ (*m x n*) is diagonal
  - ∑ has the square roots of the eigenvalues of A^TA and AA^T on its diagonal in descending order (**singular values**)
  - columns of U are the corresponding eigenvectors of AA^T (**left singular vectors**)
  - columns of V are the correspondingeigenvectors of A^TA (**right singular vectors**)
  - if we only keep the top *k* singular values of A, we get the optimal rank *k* approximation A_k of A

$$A = U \Sigma V^T \qquad\qquad A_k = U_k \Sigma_k V_k^T$$



Interpreting the SVD

- examine the **rank-2 decomposition of A**
  - rows of A (documents) and columns of A (terms) projected onto a 2-dimensional space, the **latent concept space**
  - „bike“ and „harley“ as well as *doc1* and *doc2* point into the same direction („berlin“ and *doc3* point into perpendicular directions)

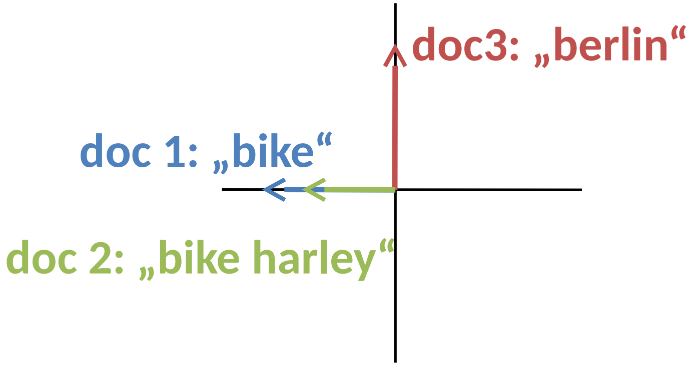
$$U_2 = \begin{array}{c} doc1 \\ doc2 \\ doc3 \end{array} \begin{bmatrix} -.53 & 0 \\ -.85 & 0 \\ 0 & 1 \end{bmatrix} \quad \Sigma_2 = \begin{bmatrix} 1.62 & 0 \\ 0 & 1 \end{bmatrix} \quad V_2 = \begin{array}{c} bike \\ harley \\ berlin \end{array} \begin{bmatrix} -.85 & 0 \\ -.53 & 0 \\ 0 & 1 \end{bmatrix}$$

  - dimensions of the space loosely correspond to concepts („motorcycles“ and „berlin“)
  - replacement documents and terms with vectors that represent their association to the concepts
  - singular values denote the “importance” of the concepts

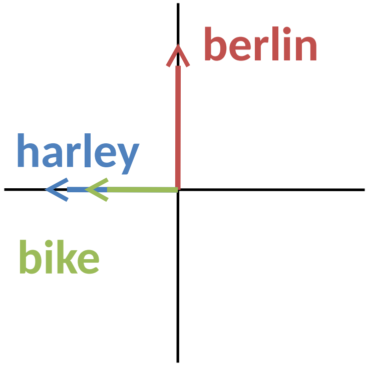
Interpreting the SVD

- **latent concept space**
  - dimensions represent „concepts“ (might be hard to interpret)
  - conceptually similar documents and terms are near to each other

documents



terms



Search in the concept space

- **search in the concept space**
  - project the query into the concept space (**fold-in**)

$$q = \begin{array}{ccc} bike & harley & berlin \\ 0 & 1 & 0 \end{array} \qquad \hat{q} = q\ V\ \Sigma^{-1} = \begin{bmatrix} -.85 & 0 \end{bmatrix}$$

Technische Universität Berlin

Search in the concept space **(the punchline)**

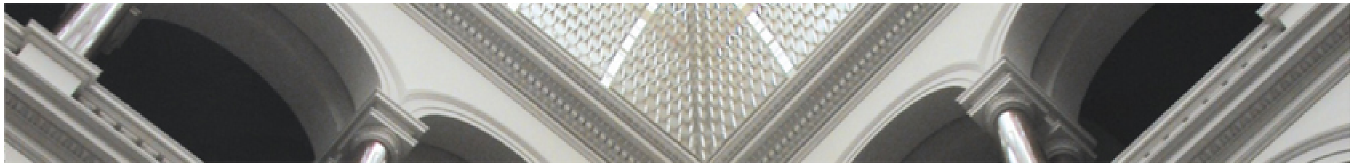
- **search in the concept space**
  - project the query into the concept space (**fold-in**)

$$
\begin{array}{cccc}
 & bike & harley & berlin \\
q = & 0 & 1 & 0
\end{array}
\qquad
\hat{q} = q\ V\ \Sigma^{-1} = \begin{bmatrix} -.85 & 0 \end{bmatrix}
$$

  - compare the projected query to the document concept vectors

$$
U_2\ \hat{q}^T = \begin{bmatrix} -.53 & 0 \\ -.85 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} -.85 \\ 0 \end{bmatrix} = \begin{array}{c} \begin{bmatrix} 0.45 \\ 0.72 \\ 0 \end{bmatrix} \end{array} \begin{array}{l} doc1 \\ doc2 \\ doc3 \end{array}
$$

  - **query matches document 1 although it does not contain the search term *“harley”* !!!**



Drawbacks of Latent Semantic Indexing

- computing the SVD of a large corpus is computationally expensive
  - needs **constant re-computation** for new documents

- **hard to scale**:
  - document concept matrix typically dense
    → every document needs to be inspected at query time!

- works well for synonyms but does not handle polysemy

Overview

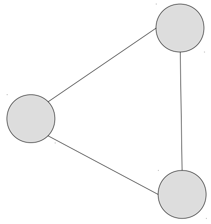
- Representing data as matrices
- Intuition
- Applications
  - Latent Semantic Indexing
  - **Estimating the number of triangles in a network**
- Singular Value Decomposition at Scale
- Summary

Technische Universität Berlin

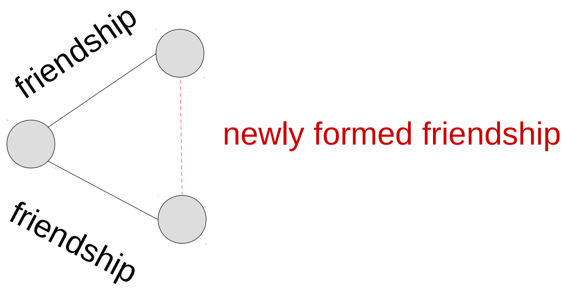
Social Graphs and Triangles

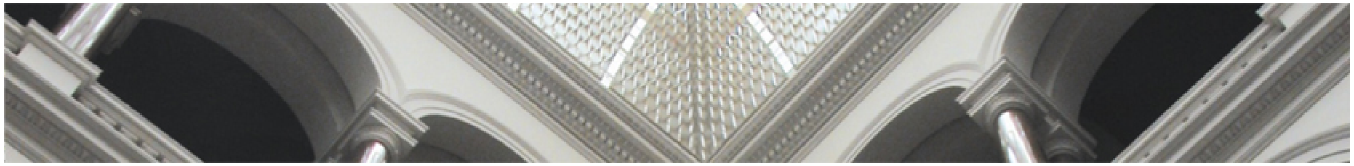
- **social graphs**
  - vertices represent users
  - edges represent connections between users, e.g. friendship, following, etc

- a **triangle** in a graph is a triple of interconnected vertices *(3-clique)*



- social graphs grow by **“closing triangles”**:
  - “becoming friends with a friend of a friend”

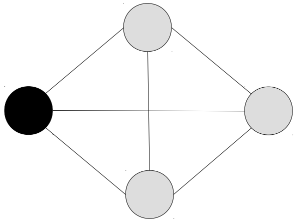




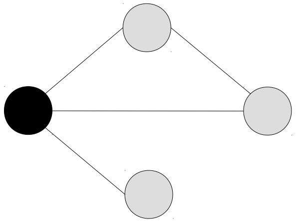
Local clustering coefficient

- a measure of the local “**connectedness”** of a vertex
  - the number of links between the neighbors of a vertex *i* divided by the maximum possible number of links between these neighbors

$$C_i = \frac{2\left|\left\{e_{jk} : v_j, v_k \in N_i, e_{jk} \in E\right\}\right|}{k_i(k_i - 1)}$$



local clustering coefficient of 1

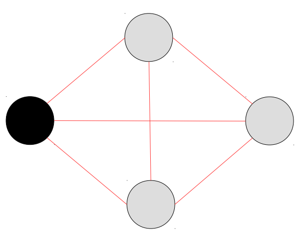


local clustering coefficient of 1/3

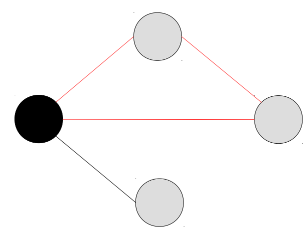
Local clustering coefficient

- local clustering coefficient C_i of a vertex *i* is connected to the **number of triangles t_i** which vertex *i* is a part of

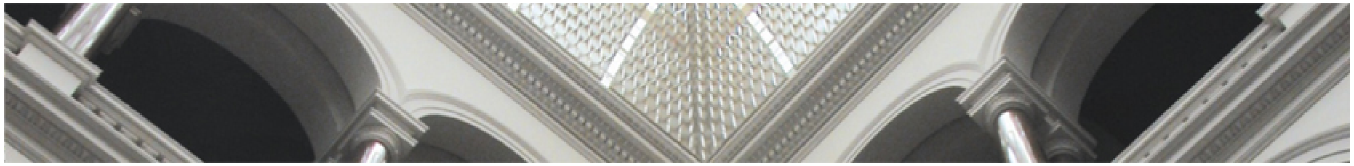
$$C_i = \frac{2\left|\left\{e_{jk} : v_j, v_k \in N_i, e_{jk} \in E\right\}\right|}{k_i(k_i - 1)} = \frac{2t_i}{k_i(k_i - 1)}$$



local clustering coefficient of 1



local clustering coefficient of 1/3



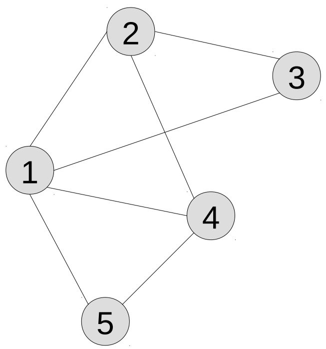
Applications of the local clustering coefficient

- statistic of interest in **network science**

- can be used to **identify spammers amongst high degree vertices in a social network**
  - spammers should have a local clustering coefficient below average
    as they typically randomly connect to other users

Adjacency matrix of a graph

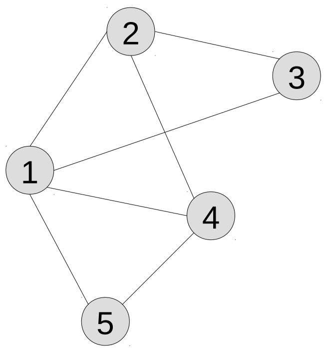
- A_{ij} is 1 if there is an edge between vertices *i* and *j*, 0 otherwise



$$A = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Powers of the adjacency matrix

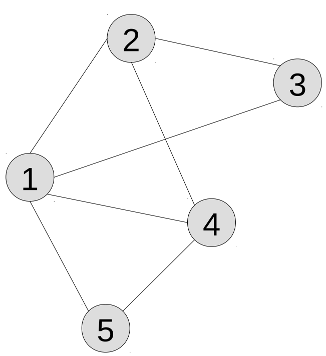
- **powers of the adjacency matrix give information about paths in a network**
- e.g., cell *(i,j)* of A^2 holds the number of paths of length 2 between vertex *i* and vertex *j*



$$A^2 = \begin{bmatrix} 4 & 2 & 1 & 2 & 1 \\ 2 & 3 & 1 & 1 & 2 \\ 1 & 1 & 2 & 2 & 1 \\ 2 & 1 & 2 & 3 & 1 \\ 1 & 2 & 1 & 1 & 2 \end{bmatrix}$$

Powers of the adjacency matrix

- a **triangle is a path of length 3** from a vertex back to itself
  → the diagonal of A^3 holds the number of triangles* for each vertex
- unfortunately, multiplying large matrices quickly becomes infeasible



$$A^3 = \begin{bmatrix} 6 & & & & \\ & 4 & & & \\ & & 2 & & \\ & & & 4 & \\ & & & & 2 \end{bmatrix}$$

*(actually, its twice the number of triangles, as we count each triangle twice)

Diagonalization

- **Diagonalization** of a diagonalizable square *n x n* matrix A:
  - Q (*n x n*) is orthogonal, Δ (*n x n*) is diagonal
  - Δ has the the **eigenvalues** of *A* on its diagonal in descending order
  - Q has the corresponding eigenvectors of A as columns
  - if we only keep the top *k* eigenvalues of A, we get the optimal rank *k* approximation A_k of A

$$A = Q \Delta Q^T$$

Diagonalization and the powers of a matrix

- diagonalization provides an easy way to compute the powers of a matrix

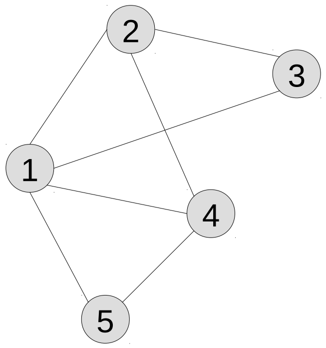
$$A = Q \Delta Q^T$$

- after diagonalization of A, **powers of A can be computed by computing only the powers of Δ** (which is easy since Δ is a diagonal matrix)

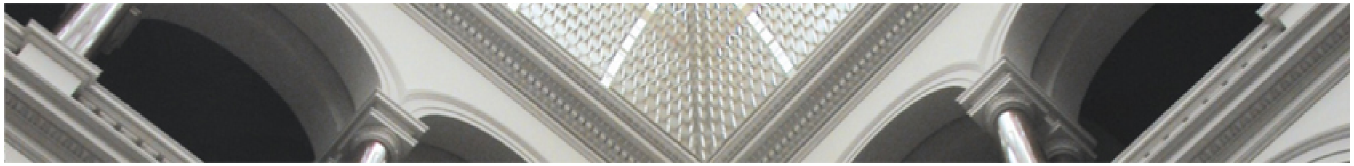
$$A^3 = Q \Delta Q^T Q \Delta Q^T Q \Delta Q^T = Q \Delta^3 Q^T$$

Powers of the adjacency matrix

- using only a few eigenvectors and eigenvalues usually suffices to get a good estimate of the number of triangles



$$A^3 \approx Q_3 \Delta_3^{\,3} Q_3^{\,T} = \begin{bmatrix} \mathbf{6.0093} & & & & \\ & \mathbf{3.9931} & & & \\ & & \mathbf{1.9937} & & \\ & & & \mathbf{3.9931} & \\ & & & & \mathbf{1.99} \end{bmatrix}$$



Overview

- Representing data as matrices
- Intuition
- Applications
  - Latent Semantic Indexing
  - Estimating the number of triangles in a network
- **Singular Value Decomposition at Scale**
- Summary

Singular Value Decomposition

- if matrix A only has a small number of columns:
  - compute $A^T A$ at scale
  - use a fast single machine solver to compute the first k eigenvalues and eigenvectors of of $A^T A$
  - this gives $\sum_k$ and V_k
  - compute U_k at scale as $A\ V_k \sum^{-1}_k$

Lanczos algorithm

- Lanczos algorithm
  - computes eigendecomposition of matrix *A*
  - SVD can be computed from eigendecomposition of A^TA

  - Krylov subspace method, repeatedly multiplies matrix with a random vector
  - creates a tridiagonal matrix *T* which has the same eigenvalues as *A* (and makes them easy to find)

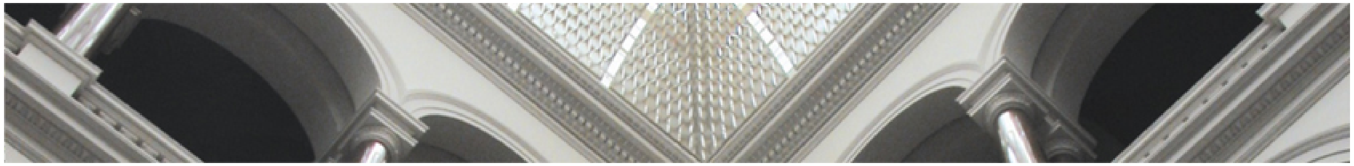
  - easy to scale as it requires only matrix-vector multiplications

$v_1 \leftarrow$ random vector with norm 1
$v_0 \leftarrow 0$
$\beta_1 \leftarrow 0$

for j = 1,2,…,m-1
    $w_j \leftarrow A\, v_j$
    $\alpha_j \leftarrow w_j\, v_j$
    $w_j \leftarrow w_j - \alpha_j\, v_j - \beta_j\, v_{j-1}$
    $\beta_{j+1} \leftarrow |\, w_j \,|$
    $v_{j+1} \leftarrow w_j \,/\, \beta_{j+1}$

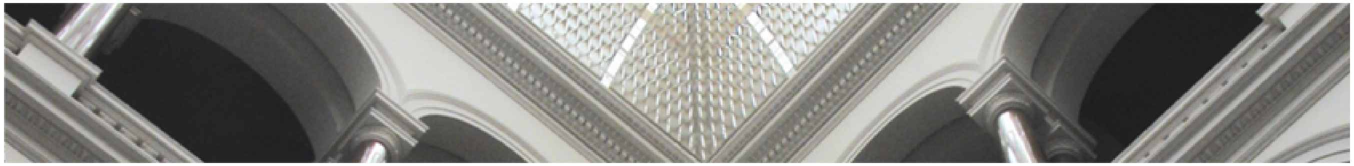
$w_m \leftarrow A\, v_m$
$\alpha_m \leftarrow w_m\, v_m$

$$T = \begin{pmatrix} \alpha_1 & \beta_2 & 0 & 0 & 0 & 0 \\ \beta_2 & \alpha_2 & \beta_3 & 0 & 0 & 0 \\ 0 & \beta_3 & a_3 & \dots & 0 & 0 \\ 0 & 0 & \dots & \dots & \beta_{m-1} & 0 \\ 0 & 0 & 0 & \beta_{m-1} & a_{m-1} & \beta_m \\ 0 & 0 & 0 & 0 & \beta_m & a_m \end{pmatrix}$$



Overview

- Representing data as matrices
- Intuition
- Applications
  - Latent Semantic Indexing
  - Estimating the number of triangles in a network
- Singular Value Decomposition at Scale
- **Summary**



Technische Universität Berlin

Summary

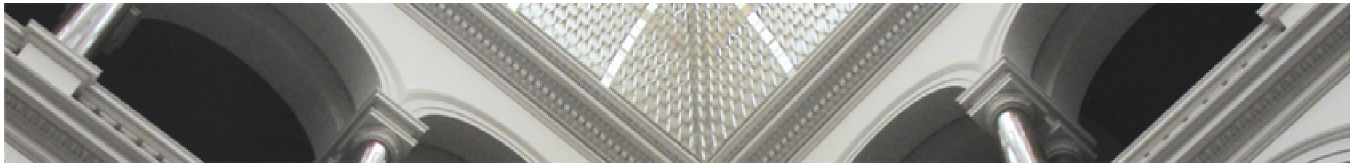
- often: **data looks high dimensional, but has low-rank structure**

- **dimensionality reduction projects data onto a low-rank space**
  - via matrix decomposition techniques, e.g. SVD, Eigendecomposition

- many interesting problems solvable via generalization provided by dimensionality reduction
  - Latent Semantic Indexing
  - Triangle Count Estimation
  - Latent Factor Models for Recommender Systems (coming up)

- SVD at Scale
  - use **Lanczos algorithm** (main operation: matrix-vector multiplications)

- not covered:
  - **Principal Components Analysis (PCA)**, as it is hard to scale



Further Reading

- Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to information retrieval* (Vol. 1, p. 496). Cambridge: Cambridge university press.

- Dumais, S. T. (2004). *Latent semantic analysis.* Annual review of information science and technology, 38(1), 188-230.

- Kang, U., Meeder, B., & Faloutsos, C. (2011). *Spectral analysis for billion-scale graphs: Discoveries and implementation.* In Advances in Knowledge Discovery and Data Mining (pp. 13-5). Springer Berlin Heidelberg.

- Strang, G. (1993). *The fundamental theorem of linear algebra.* American Mathematical Monthly, 848-855.