AIM3 – Scalable Data Analysis and Data Mining

06 – Dimensionality Reduction

Sebastian Schelter, Christoph Boden, Volker Markl

Fachgebiet Datenbanksysteme und Informationsmanagement
Technische Universität Berlin

http://www.dima.tu-berlin.de/

think of data in terms of vectors and matrices

Representing data as matrices

- think of data in terms of vectors and matrices
  - Text
    - documents x terms
  - Ratings
    - users x items
  - Graphs
    - vertices x vertices
- in a lot of use cases this data is extremely high dimensional
  → „*Curse of dimensionality*"
  - *extremely sparse matrices*
  - *nearest neighbor search*
- goal: reduce the data to the „interesting" dimensions

A concrete example:
Latent Semantic Indexing

The problems with lexical matching

- Imagine a corpus with only three very simple documents:
  - *doc1* : „bike“
  - *doc2*: „harley bike“
  - *doc3*: „berlin“

- Now we search for „harley“:
  - only *doc2* is found, although *doc1* might be relevant too!

- General drawbacks of lexical matching
  - **Synonymy**: huge diversity in the words people use for describing a document (think of reformulating Google queries...)
  - **Polysemy**: words with multiple meanings might match irrelevant documents (query about the planet mars like „size of mars“ might return documents about the chocolate bar)

Possible solutions: query expansion

- Create custom taxonomies of weighted relations
  - „harley-> bike 0.5"
- Manually 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

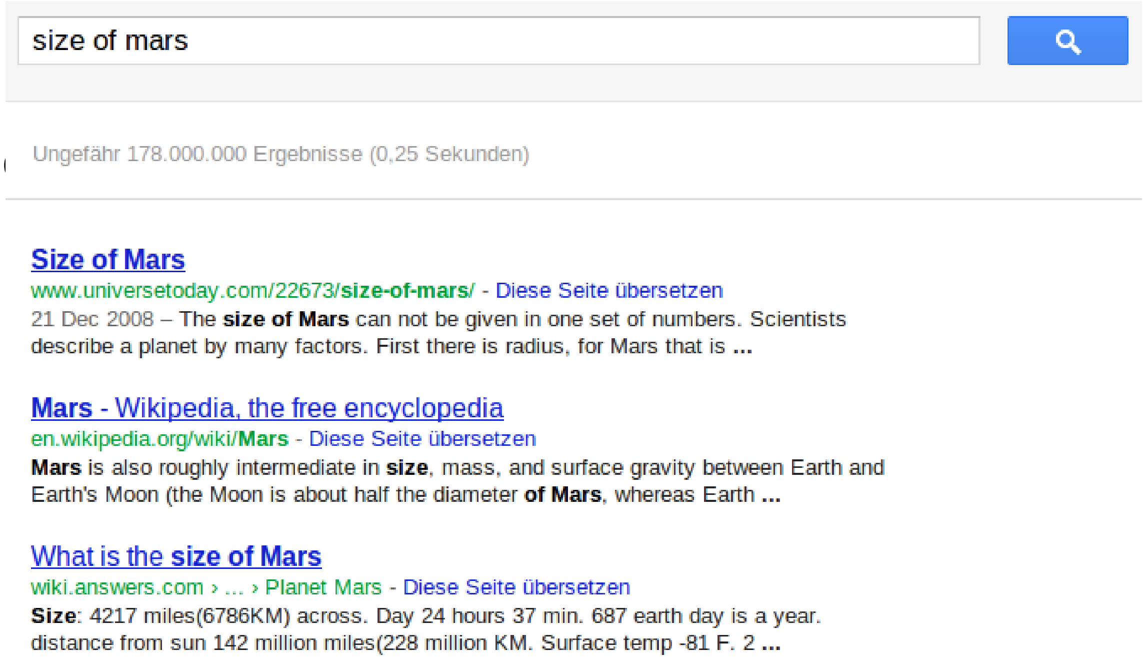
Possible solutions: query suggestions

- Detect queries that can profit from reformulation and suggest refinements

- Drawbacks
  - bad user experience: more clicks and decisions necessary
  - needs a sufficient amount of training data
  - generates a lot of queries

Possible solutions: learn to rank

- Use additional data to learn an optimal ranking of the search results
  - user feedback (relevance feedba
  - link structure of the documents (PageRank)



- Drawbacks
  - complex
  - might need a sufficient amount of training data

Can‘t we do better?

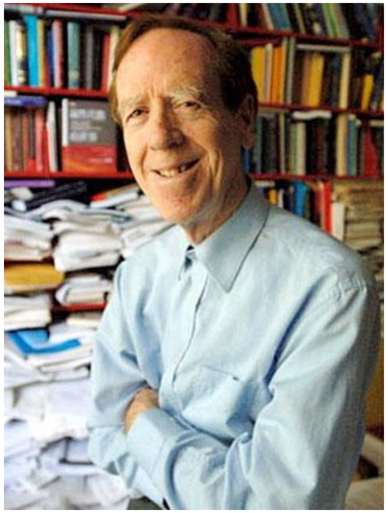
- Intuition suggests:
  - there is already some kind of structure contained in the corpus that describes the relations among terms and documents
  - we just can‘t see it!

- Say terms and documents belong to „concepts“, then:
  - a single term describing a particular „concept“ will occur in documents about that „concept“
  - terms describing the same concept will co-occur in documents about that „concept“
  - documents about a particular „concept“ will share a set of characteristic terms

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

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



  - queries „harley“ and „harley bike“ are just vectors in the term space

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

Search as matrix-vector multiplication

- Let's use the number of shared terms as similarity measure between queries and documents
  - searching becomes matrix-vector multiplication! $$A\,q^T$$
  - examples: search for „harley" and „harley bike"

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

$$A\,q_1^T = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{array}{cc} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} & \begin{array}{c} doc1 \\ doc2 \\ doc3 \end{array} \end{array}$$

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

$$A\,q_2^T = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} = \begin{array}{cc} \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix} & \begin{array}{c} doc1 \\ doc2 \\ doc3 \end{array} \end{array}$$

- AA^T document similarities
  - a cell m,n holds the number of terms shared by documents *m* and *n*

$$AA^T = \begin{array}{c|ccc} & doc1 & doc2 & doc3 \\ \hline doc1 & 1 & 1 & 0 \\ doc2 & 1 & 2 & 0 \\ doc3 & 0 & 0 & 1 \end{array}$$

  → *doc1* and *doc2* are similar

- $A^T A$ term co-occurrences
  - a cell m,n holds the number of documents in which terms *m* and *n* occur together

$$A^T A = \begin{array}{c|ccc} & bike & harley & berlin \\ \hline bike & 2 & 1 & 0 \\ harley & 1 & 1 & 0 \\ berlin & 0 & 0 & 1 \end{array}$$

  → „harley“ and „bike“ related

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, Σ (*m x n*) is diagonal
  - Σ 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 corresponding eigenvectors 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

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

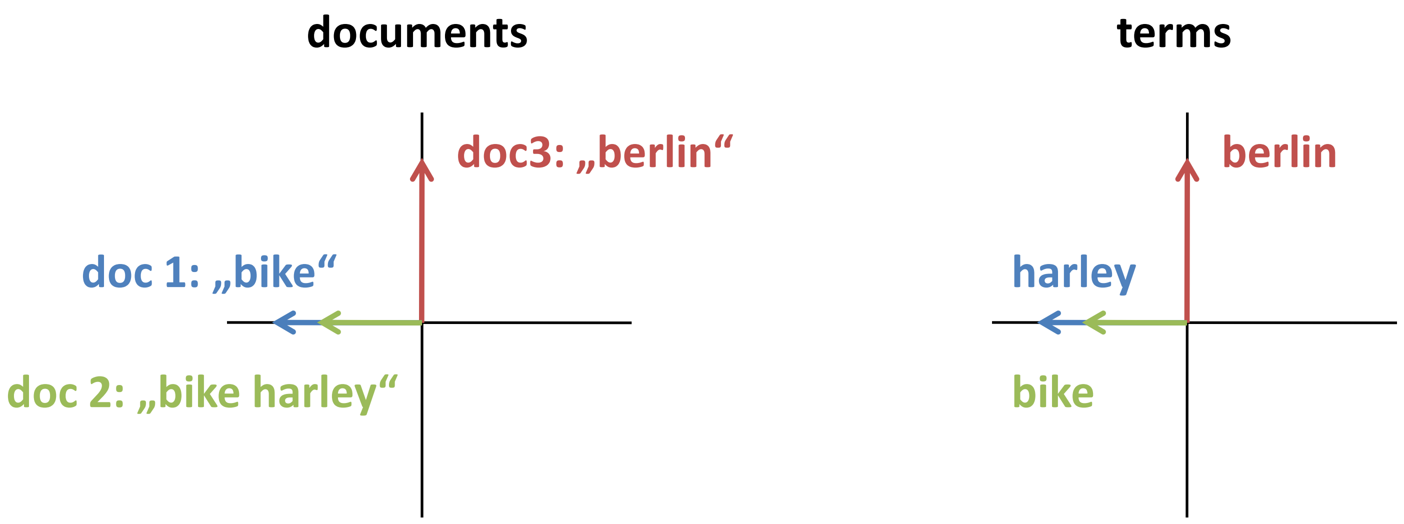
Interpreting the SVD

- Let's have a look at the rank-2 decomposition of A
  - rows of A (documents) and columns of A (terms) are projected onto a 2-dimensional space, the **concept space**
  - notice that „bike" and „harley" as well as *doc1* and *doc2* point into the same direction (and „berlin" and *doc3* point into a perpendicular direction)

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

  - the dimensions of the space correspond to concepts hidden in the corpus („motorcycles" and „berlin" in our example)
  - documents and terms are replaced with vectors that represent their association to the concepts
  - the singular values denote the importance of the concepts

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



Latent Semantic Indexing (LSI)

- Search in the concept space
  - project the query onto 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

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

$\rightarrow$ query matches *doc1* although it does not contain the term „harley"

Drawbacks of LSI

- Lack of solid statistical foundation
  - assumes Gaussian distribution of terms (wrong!)
  - has led to the development of *Probabilistic Latent Semantic Indexing* (*pLSI*) and *Latent Dirichlet Allocation* (*LDA*)
- Computing the SVD of a large corpus is computationally expensive
  - but an interesting research problem ☺
  - needs updating for new documents
- Hard to scale
  - at query time each document needs to be inspected
- mainly a solution for synonymy not polysemy

Estimating the number of triangles in a graph

Counting triangles in social graphs

- social graphs
  - vertices are users
  - edges are connections between users such as friendships, followings, etc

- triangle
  - a triple of completely inter-connected users
  - social graphs often grow by closing triangles
- local clustering coefficient
  - number of existing triangles around a vertex divided by the number of possible vertices around it
  - a measure of its local „connectedness“

Counting triangles in (undirected) social graphs

- create the adjacency matrix A of a graph
  - a_{ij} is 1 if there is an edge between vertices i and j, 0 otherwise
- multiplying A by itself reveals information about the connectivity of the graph
  - each entry a_{ij}^{3} of A^3 holds the number of paths of length 3 from vertex i to vertex j
  - that means the diagonal of A^3 holds the number of triangles for each vertex!
  - unfortunately multiplying large matrices is unfeasible...
- but we can use the diagonalization of A to estimate the number of triangles!

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

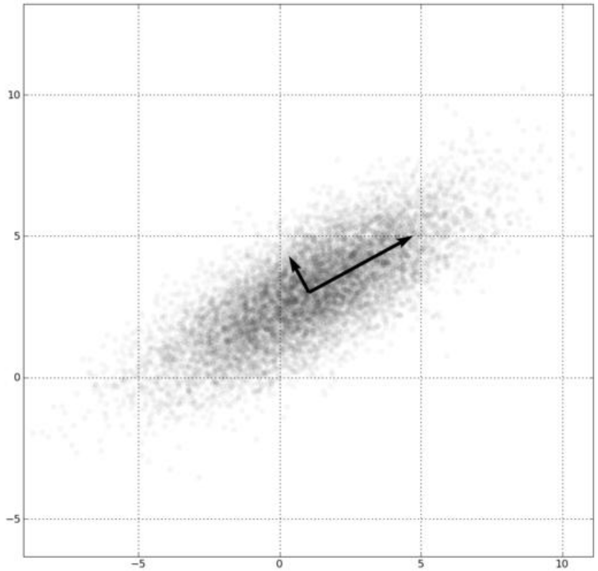
$$A^3 = Q \Delta Q^T Q \Delta Q^T Q \Delta Q^T = Q \Delta^3 Q^T$$

Principal Component Analysis (PCA)

Principal Component Analysis

- mathematical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called **principal components**

- Algorithm
  - center the data
  - compute the covariance matrix
  - compute an eigenvalue decomposition of the covariance matrix



- PCA is extremely hard to scale because of the density of the original matrix after centering!
  - unfeasible to compute the covariances of large dense matrices

Decomposing large matrices

Lanczos algorithm for computing the SVD

- basic idea
  - iteratively multiply the matrix A with a random initial basis vector
  - reorthogonalize the resulting vectors to a basis of the so called Krylov subspace of A
  - use these to create tridiagonal matrix T_{mm} whose eigenvalues/eigenvectors are a good approximation of the eigenvalues/eigenvectors of A

$$
\begin{aligned}
&\mathrm{v}_1 \leftarrow \text{random vector with norm } 1\\
&v_0 \leftarrow 0\\
&ß_0 \leftarrow 0\\
&\text{for } j = 1 \text{ to } m\\
&\quad w_j \leftarrow Av_j - \beta_j v_{j-1}\\
&\quad \alpha_j \leftarrow (w_j, v_j)\\
&\quad w_j \leftarrow w_j - \alpha_j v_j\\
&\quad ß_{j+1} \leftarrow \|w_j\|\\
&\quad v_{j+1} \leftarrow w_j / ß_{j+1}
\end{aligned}
$$

- main operation that needs to be parallelized:

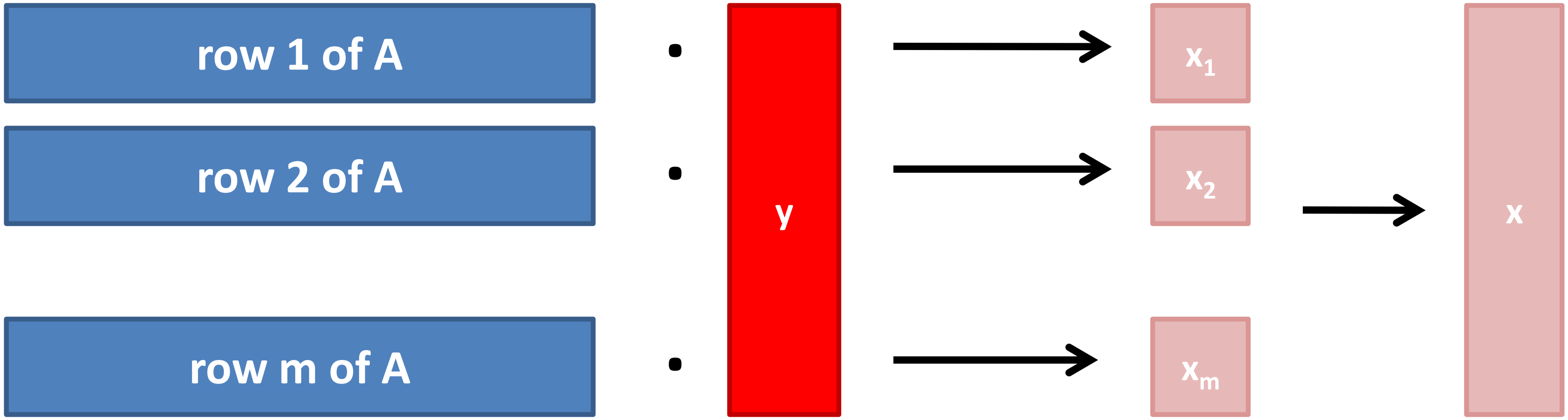
  matrix vector multiplication

$$
T_{mm} = \begin{bmatrix} \alpha_1 & ß_2 & & 0 \\ ß_2 & \alpha_2 & ß_3 & \\ & ß_3 & \dots & ß_m \\ 0 & & ß_m & \alpha_m \end{bmatrix}
$$

Parallelizing Ay = x

- Matrix A is partitioned by rows (in distributed filesystem)

- **Hadoop**: broadcast y to all machines, *MAP* computes the m-th component of x by multiplying the m-th row of A with y, *REDUCE* collects all components of x
- **Stratosphere**: row-wise multiplication in a *CROSS* between rows of A and y, *REDUCE* again collects all components of x



Stochastic SVD

- a **randomized, non-iterative** algorithm for computing the SVD of large matrices

$$A = U \Sigma V^T$$

- draw a random *n* x *k* matrix
- compute an *n* x *k* random sample matrix Y, whose columns form a basis for the range of A
- form an *n* x *k* matrix Q whose columns form an orthonormal basis for the columns of Y
- form the small *k* x *n* matrix B
- decompose B on a single machine
- use this to compute U

$$Y = A\Omega$$

$$Q = \mathrm{qr}(Y)$$

$$B = Q^T A$$

$$B = \hat{U} \Sigma V^T$$

$$U = Q \hat{U}$$