Outline

Goal: ‘Find similar / interesting things’

- Intro to DB
- Indexing - similarity search
- Data Mining

Indexing - Detailed outline

- primary key indexing
- secondary key / multi-key indexing
- spatial access methods
- fractals
- text
- Singular Value Decomposition (SVD)
- multimedia
- ...

SVD - Detailed outline

- Motivation
- Definition - properties
- Interpretation
- Complexity
- Case studies
- SVD properties
- More case studies
- Conclusions

SVD - detailed outline

- ...
- Case studies
- SVD properties
  - more case studies
    - google/Kleinberg algorithms
    - query feedbacks
  - Conclusions

SVD - Other properties - summary

- can produce orthogonal basis (obvious) (who cares?)
- can solve over- and under-determined linear problems (see C(1) property)
- can compute ‘fixed points’ (= ‘steady state prob. in Markov chains’) (see C(4) property)
SVD - outline of properties

- (A): obvious
- (B): less obvious
- (C): least obvious (and most powerful!)

Less obvious properties

\[ A(0): A_{[n \times m]} = U_{[n \times r]} A_{[r \times r]} V^T_{[r \times m]} \]

\[ B(1): A_{[n \times m]} (A^T_{[m \times n]}) = ?? \]

Properties - by defn.:

\[ A(0): A_{[n \times m]} = U_{[n \times r]} A_{[r \times r]} V^T_{[r \times m]} \]

\[ A(1): U^T_{[r \times n]} U_{[n \times r]} = I_{[r \times r]} \text{ (identity matrix)} \]

\[ A(2): V^T_{[r \times n]} V_{[n \times r]} = I_{[r \times r]} \]

\[ A(3): A^k = \text{diag}(\lambda_1^k, \lambda_2^k, ..., \lambda_r^k) \text{ (k: ANY real number)} \]

\[ A(4): A^T = V A U^T \]

Less obvious properties

\[ A(0): A_{[n \times m]} = U_{[n \times r]} A_{[r \times r]} V^T_{[r \times m]} \]

\[ B(1): A_{[n \times m]} (A^T_{[m \times n]}) = U A^2 U^T \]

symmetric; Intuition? ‘document-to-document’ similarity matrix

\[ B(2): \text{symmetrically, for ‘V’} \]

\[ (A^T_{[m \times n]} A_{[n \times m]})^k = V A^2 V^T \]

Intuition?

Less obvious properties

\[ A: \text{term-to-term similarity matrix} \]

\[ B(3): (A^T_{[m \times n]} A_{[n \times m]})^k = V A^2 V^T \]

and

\[ B(4): (A^T A)^k \sim v_1 \lambda_1^k v_1^T \text{ for k>>1} \]

where

\[ v_1: [m \times 1] \text{ first column (singular-vector) of } V \]

\[ \lambda_1: \text{strongest singular value} \]
Less obvious properties

B(4): \((A^T A)^k v_i \lambda_i^{2k} v_i^T\) for \(k \gg 1\)

B(5): \((A^T A)^k v' \sim (\text{constant}) v_i\)

ie., for (almost) any \(v'\), it converges to a vector parallel to \(v_i\).

Thus, useful to compute first singular vector/value (as well as the next ones, too...)

Least obvious properties

A(0): \(A_{[n \times m]} = U_{[n \times r]} A_{[r \times r]} V^T_{[r \times m]}\)

C(1): \(A_{[n \times m]} x_{[m \times 1]} = b_{[n \times 1]}\)

let \(x_0 = \lambda_{(1)}^{-1} U^T b\)

if under-specified, \(x_0\) gives ‘shortest’ solution

if over-specified, it gives the ‘solution’ with the smallest least squares error

(see Num. Recipes, p. 62)

Least obvious properties - cont’d

A(0): \(A_{[n \times m]} = U_{[n \times r]} A_{[r \times r]} V^T_{[r \times m]}\)

C(2): \(A_{[n \times m]} v_{i[1 \times 1]} \sim \lambda_i u_i_{[n \times 1]}\)

where \(v_i, u_i\) the first (column) vectors of \(V, U\) (\(v_i\) == right-singular-vector)

\(u_i\) == left-singular-vector

Therefore:

Illustration: over-specified, eg

\([3 \ 2]^T [w] = [1 \ 2]^T\) (ie, \(3 \ w = 1; \ 2 \ w = 2\))

desirable point \(b\)

reachable points \((3w, 2w)\)
Least obvious properties - cont’d

\[ A(0): A_{n \times m} = U_{n \times r} A_{r \times r} V^T_{r \times m} \]

\[ C(4): A^T A v_1 = \lambda_1^2 v_1 \]

(fixed point - the def of eigenvector for a symmetric matrix)

---

Properties - conclusions

\[ A(0): A_{n \times m} = U_{n \times r} A_{r \times r} V^T_{r \times m} \]

\[ B(5): (A^T A)^k v^* \sim (constant) v_1 \]

\[ C(1): A_{n \times m} x_{n \times 1} = b_{n \times 1} \]
then, \( x_0 = V \Lambda^{-1} U^T b \): shortest, actual or least-squares solution

\[ C(4): A^T A v_1 = \lambda_1^2 v_1 \]

---

SVD - detailed outline

• ...
• Case studies
• SVD properties
• more case studies
  – Kleinberg/google algorithms
  – query feedbacks
• Conclusions

---

Kleinberg’s algorithm

• Problem def: given the web and a query
• find the most ‘authoritative’ web pages for this query

Step 0: find all pages containing the query terms
Step 1: expand by one move forward and backward
Kleinberg’s algorithm

• on the resulting graph, give high score (= ‘authorities’) to nodes that many important
  nodes point to
• give high importance score (‘hubs’) to
  nodes that point to good ‘authorities’

Kleinberg’s algorithm

observations
• recursive definition!
• each node (say, ‘i’-th node) has both an
  authoritativness score $a_i$ and a hubness
  score $h_i$

Kleinberg’s algorithm

Then:

In conclusion, we want vectors $h$ and $a$ such that:

$h = A \ a$
$a = A^T \ h$

Recall properties:
C(2): $A \ [n \times m] \ y \ [m \times 1] = \lambda \ y \ [n \times 1]$
C(3): $u_i^T \ A = \lambda_i \ y_i^T$
Kleinberg’s algorithm

In short, the solutions to
\[ h = A \ a \]
\[ a = A^T \ h \]
are the left- and right- singular-vectors of the adjacency matrix \( A \). Starting from random \( a' \) and iterating, we’ll eventually converge (Q: to which of all the singular-vectors? why?)

Kleinberg’s algorithm - results

Eg., for the query ‘java’:
0.328 www.gamelan.com
0.251 java.sun.com
0.190 www.digitalfocus.com (“the java developer”)

Kleinberg’s algorithm

(Q: to which of all the singular-vectors? why?)
\[ A: \] to the ones of the strongest singular-value, because of property \( B(5) \):
\[ B(5): (A^T A)^k \ v' \sim \ \text{(constant)} \ v_1 \]

Kleinberg’s algorithm - discussion

• ‘authority’ score can be used to find ‘similar pages’ (how?)
• closely related to ‘citation analysis’, social networks / ‘small world’ phenomena

Google/Page-Rank algorithm

• closely related: imagine a particle randomly moving along the edges (*)
• compute its steady-state probabilities

(*) with occasional random jumps

• ~identical problem: given a Markov Chain, compute the steady state probabilities \( p_1 \ldots p_5 \)
(Simplified) PageRank algorithm

- Let $A$ be the transition matrix (= adjacency matrix); let $A^T$ become column-normalized - then

\[ A^T \cdot p = p \]

$A^T$ is symmetric and $x^T \cdot A^T \cdot x = \lambda x^T \cdot x$.

(Simplified) PageRank algorithm

- Thus, $p$ is the eigenvector that corresponds to the highest eigenvalue (=1, since the matrix is column-normalized)
- Formal definition of eigenvector/value: soon

Formal definition

If $A$ is a $(n \times n)$ square matrix, $(\lambda, x)$ is an eigenvalue/eigenvector pair of $A$ if

\[ A \cdot x = \lambda \cdot x \]

Closely related to singular values:

Eigen- vs singular-values

If

\[ B = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix}, \quad v = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix}, \quad \lambda \]

then $A = (B^T B)$ is symmetric and

\[ C(4): B^T B \cdot v = \lambda \cdot v \]

ie, $v_1, v_2, \ldots$: eigenvectors of $A = (B^T B)$.
• By definition, eigenvectors remain parallel to themselves ("fixed points")

Convergence

• Usually, fast:

Convergence

• Usually, fast:

Kleinberg/google - conclusions

SVD helps in graph analysis:
hub/authority scores: strongest left- and right-singular-vectors of the adjacency matrix
random walk on a graph: steady state probabilities are given by the strongest eigenvector of the transition matrix
SVD - detailed outline

- ...  
- Case studies  
- SVD properties  
- more case studies  
  - google/Kleinberg algorithms  
  - query feedbacks  
- Conclusions

Query feedbacks

[Chen & Roussopoulos, sigmod 94]

sample problem:
estimate selectivities (e.g., ‘how many movies
were made between 1940 and 1945?’

for query optimization,
LEARNING from the query results so far!!
Eventually, the problem becomes:
- estimate the parameters $a_1, \ldots, a_7$ of the model
- to minimize the least squares errors from the real answers so far.
Formally:

Formally, with $n$ queries and 6-th degree polynomials:

$$
\begin{bmatrix}
X_{11} & X_{12} & \cdots & X_{17} \\
X_{21} & X_{22} & \cdots & X_{27} \\
\vdots & \vdots & \ddots & \vdots \\
X_{n1} & X_{n2} & \cdots & X_{n7}
\end{bmatrix}
= \begin{bmatrix}
a_1 \\
a_2 \\
\vdots \\
a_7
\end{bmatrix}
$$

where $x_{ij}$ such that $\text{Sum}(x_{ij} \cdot a_i) = \text{our estimate for the # of movies}$ and $b_j$: the actual

In matrix form:

$$
X \ a = b
$$

and the least-squares estimate for $a$ is

$$
a = V \Lambda^{-1} U^T b
$$

according to property $C(1)$

(let $X = U \Lambda V^T$)
Query feedbacks - enhancements

the solution
\[ a = V \Lambda^{-1} U^T b \]
works, but needs expensive SVD each time a new query arrives
GREAT Idea #3: Use ‘Recursive Least Squares’, to adapt \( a \) incrementally.

Details: in paper - intuition:

Intuition:

\[ a_1 x + a_2 \]
new query

Query feedbacks - enhancements

GREAT idea #4: ‘forgetting’ factor - we can even down-play the weight of older queries, since the data distribution might have changed.
(comes for ‘free’ with RLS...)
Query feedbacks - conclusions

SVD helps find the Least Squares solution, to adapt to query feedbacks
(RLS = Recursive Least Squares is a great method to incrementally update least-squares fits)

SVD - detailed outline

• ...
• Case studies
• SVD properties
• more case studies
  – google/Kleinberg algorithms
  – query feedbacks

Conclusions

• SVD: a valuable tool
• given a document-term matrix, it finds ‘concepts’ (LSI)
• ... and can reduce dimensionality (KL)
• ... and can find rules (PCA; RatioRules)

Conclusions cont’d

• ... and can find fixed-points or steady-state probabilities (google/ Kleinberg/ Markov Chains)
• ... and can solve optimally over- and under-constraint linear systems (least squares / query feedbacks)

References


References cont’d