15-826: Multimedia Databases and Data Mining

Lecture #20: SVD - part III (more case studies)
C. Faloutsos

Must-read Material

• [Textbook](#) Appendix D

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!)

---

**Properties - by defn.:**

\[ A(0): \mathbf{A}_{[n \times m]} = \mathbf{U}_{[n \times r]} \mathbf{\Lambda}_{[r \times r]} \mathbf{V}^T_{[r \times m]} \]

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

\[ A(2): \mathbf{V}^T_{[r \times n]} \mathbf{V}_{[n \times r]} = \mathbf{I}_{[r \times r]} \]

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

\[ A(4): \mathbf{A}^T = \mathbf{V} \mathbf{\Lambda} \mathbf{U}^T \]
Less obvious properties

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

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

symmetric; Intuition?
Less obvious properties

A: term-to-term similarity matrix

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

and

\[ B(4): (A^T A)^k \sim v_1 \lambda_1^{2k} v_1^T \text{ for } k \gg 1 \]

where

- \( v_1 \): [m x 1] first column (singular-vector) of \( V \)
- \( \lambda_1 \): strongest singular value

Less obvious properties

\[ B(4): (A^T A)^k \sim v_1 \lambda_1^{2k} v_1^T \text{ for } k \gg 1 \]

\[ B(5): (A^T A)^k v' \sim (\text{constant}) v_1 \]

ie., for (almost) any \( v' \), it converges to a vector parallel to \( v_1 \)

Thus, useful to compute first singular vector/value (as well as the next ones, too...)
Proof of (B5)?

Less obvious properties - repeated:

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

\[ B(1): \mathbf{A}_{[n \times m]} (\mathbf{A}^T)^{-1}_{[m \times n]} = U \mathbf{A}^2 U^T \]

\[ B(2): (\mathbf{A}^T)^{-1}_{[m \times n]} \mathbf{A}_{[n \times m]} = \mathbf{V} \mathbf{A}^2 \mathbf{V}^T \]

\[ B(3): ((\mathbf{A}^T)^{-1}_{[m \times n]} \mathbf{A}_{[n \times m]})^k = \mathbf{V} \mathbf{A}^{2k} \mathbf{V}^T \]

\[ B(4): (\mathbf{A}^T \mathbf{A})^k \mathbf{v} \sim (\text{constant}) \mathbf{v}_1 \]

\[ B(5): (\mathbf{A}^T \mathbf{A})^{k} \mathbf{v} \sim (\text{constant}) \mathbf{v}_1 \]

Least obvious properties

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

\[ C(1): \mathbf{A}_{[n \times m]} \mathbf{x}_{[m \times 1]} = \mathbf{b}_{[n \times 1]} \]

let \( \mathbf{x}_0 = \mathbf{V} \mathbf{A}^{(-1)} U^T \mathbf{b} \)

if under-specified, \( \mathbf{x}_0 \) gives ‘shortest’ solution
if over-specified, it gives the ‘solution’ with the smallest least squares error

(see Num. Recipes, p. 62)
Illustration: under-specified, eg

\[ [1 \ 2 \ w \ z]^T = 4 \] (ie, \( w + 2z = 4 \))

Least obvious properties

shortest-length solution

all possible solutions

Verify formula:

\[ A = [1 \ 2] \quad b = [4] \]
\[ A = U \Lambda V^T \]
\[ U = ?? \]
\[ \Lambda = ?? \]
\[ V = ?? \]
\[ x_0 = V \Lambda^{-1} U^T b \]
Verify formula:

\[
\begin{align*}
A &= \begin{bmatrix} 1 & 2 \end{bmatrix} \quad b = [4] \\
A &= U \Lambda V^T \\
U &= [1] \\
\Lambda &= [\sqrt{5}] \\
V &= \begin{bmatrix} 1/\sqrt{5} & 2/\sqrt{5} \end{bmatrix}^T \\
x_0 &= V \Lambda (-1) \quad U^T \ b = [1/5 \ 2/5]^T \quad [4] \\
&= [4/5 \ 8/5]^T: \ w = 4/5, \ z = 8/5
\end{align*}
\]

Verify formula:

Show that \( w = 4/5, \ z = 8/5 \) is
(a) A solution to \( 1*w + 2*z = 4 \) and
(b) Minimal (wrt Euclidean norm)

Verify formula:

Show that \( w = 4/5, \ z = 8/5 \) is
(a) A solution to \( 1*w + 2*z = 4 \) and
A: easy
(b) Minimal (wrt Euclidean norm)
A: \( [4/5 \ 8/5] \) is perpendicular to \( [2 \ -1] \)
Illustration: over-specified, eg
\[ [3, 2]^T \cdot [w] = [1, 2]^T \text{ (ie, } 3 \cdot w = 1; 2 \cdot w = 2) \]

Least obvious properties – cont’d

Verify formula:

\[ \Lambda = [3, 2]^T \quad b = [1, 2]^T \]
\[ \Lambda = U \Lambda V^T \]
\[ U = ?? \]
\[ \Lambda = ?? \]
\[ V = ?? \]
\[ x_0 = V \Lambda^{-1} U^T b \]
Verify formula:

\[
\begin{bmatrix} 3 & 2 \end{bmatrix}^T \begin{bmatrix} 7/13 \\ 21/13 & 14/13 \end{bmatrix}^T \rightarrow \text{red point}
\]

\[
\begin{bmatrix} 1 & 2 \end{bmatrix}^T
\]

reachable points (3w, 2w)

desirable point b

Verify formula:

\[
\begin{bmatrix} 3 & 2 \end{bmatrix}^T \begin{bmatrix} 7/13 \\ 21/13 & 14/13 \end{bmatrix}^T \rightarrow \text{red point} \rightarrow \text{perpendicular}
\]

Verify formula:

\[
A: [3 \ 2] \cdot ([1 \ 2] - [21/13 \ 14/13]) = [3 \ 2] \cdot [-8/13 \ 12/13] = [3 \ 2] \cdot [-2 \ 3] = 0
\]
Least obvious properties - cont’d

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

C(2): \( A_{[n \times m]} v_{1_{[n \times 1]}} = \lambda_{1_{1}} u_{1_{[n \times 1]}} \)
where \( v_{1} \), \( u_{1} \) the first (column) vectors of \( V, U \).
(\( v_{1} \) == right-singular-vector)
C(3): symmetrically: \( u_{1}^{T} A = \lambda_{1_{1}} v_{1}^{T} \)
where \( u_{1} \) == left-singular-vector

Therefore:

\[ \Lambda = \begin{bmatrix} \lambda_{1_{1}} & \cdots & \lambda_{n_{r}} \\ \vdots & \ddots & \vdots \\ \lambda_{n_{r}} & \cdots & \lambda_{n_{1}} \end{bmatrix} \]

Least obvious properties - cont’d

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

C(4): \( A^{T} A v_{1} = \lambda_{1_{1}}^{2} v_{1} \)

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

Least obvious properties - altogether

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

C(1): \( A_{[n \times m]} x_{[m \times 1]} = b_{[n \times 1]} \)
then, \( x_{0} = V^{T} A^{-1} U b \): shortest, actual or least-squares solution
C(2): \( A_{[n \times m]} v_{1_{[n \times 1]}} = \lambda_{1_{1}} u_{1_{[n \times 1]}} \)
C(3): \( u_{1}^{T} A = \lambda_{1_{1}} v_{1}^{T} \)
C(4): \( A^{T} A v_{1} = \lambda_{1_{1}}^{2} v_{1} \)
Properties - conclusions

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

B(5): \( (A^T A)^k v' \sim (constant) v_1 \)

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

Kleinberg’s algorithm

• Problem dfn: 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

• 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 authoritativeness score $a_i$ and a hubness score $h_j$

Then:

Let $E$ be the set of edges and $A$ be the adjacency matrix:
the $(i,j)$ is 1 if the edge from $i$ to $j$ exists

Let $h$ and $a$ be $[n \times 1]$ vectors with the ‘hubness’ and ‘authoritativness’ scores.

Then:

$$ a_i = h_k + h_j + h_m $$

that is

$$ a_i = \text{Sum} \ (h_j) \ \text{over all} \ j \ \text{that} \ \text{(j,i) edge exists} $$

or

$$ a = A^T h $$
Kleinberg’s algorithm

\[ h_i = a_n + a_p + a_q \]

that is

\[ h_j = \text{Sum} (a_q) \quad \text{over all } j \quad \text{that} \]

\((i,j) \) edge exists

or

\[ h = A a \]

In conclusion, we want vectors \( h \) and \( a \) such that:

\[ h = A a \]

\[ a = A^T h \]

Recall properties:

C(2): \( A \) [\( n \times m \)] \( v_1 \) [\( m \times 1 \)] \( = \lambda_1 \) \( u_1 \) [\( n \times 1 \)]

C(3): \( u_i^T A = \lambda_i v_i^T \)

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

(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 - 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 - discussion

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

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

PageRank (google)


Problem: PageRank

Given a directed graph, find its most interesting/central node

A node is important, if it is connected with important nodes (recursive, but OK!)
Problem: PageRank - solution

Given a directed graph, find its most interesting/central node

Proposed solution: Random walk; spot most ‘popular’ node (⇒ steady state prob. (ssp))

A node has high ssp, if it is connected with high ssp nodes

(Simplified) PageRank algorithm

• Let $A$ be the adjacency matrix;

• let $B$ be the transition matrix: transpose, column-normalized ⇒ then

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

$B^T = \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 1 & 1
\end{bmatrix}$

$B = \begin{bmatrix}
p_1 \\
p_2 \\
p_3
\end{bmatrix}$

$B = \begin{bmatrix}
p_1 \\
p_2 \\
p_3
\end{bmatrix}$

$B = \begin{bmatrix}
p_1 \\
p_2 \\
p_3
\end{bmatrix}$

(Simplified) PageRank algorithm

• $B \cdot p = p$

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

$p = \begin{bmatrix}
p_1 \\
p_2 \\
p_3
\end{bmatrix}$

$p = \begin{bmatrix}
p_1 \\
p_2 \\
p_3
\end{bmatrix}$

$p = \begin{bmatrix}
p_1 \\
p_2 \\
p_3
\end{bmatrix}$
(Simplified) PageRank algorithm

- \( B \mathbf{p} = \mathbf{1} \cdot \mathbf{p} \)
- thus, \( \mathbf{p} \) is the eigenvector that corresponds to the highest eigenvalue (=1, since the matrix is column-normalized)
- Why does such a \( \mathbf{p} \) exist?
  - \( \mathbf{p} \) exists if \( B \) is \( n \times n \), nonnegative, irreducible
  [Perron–Frobenius theorem]

In short: imagine a particle randomly moving along the edges

compute its steady-state probabilities (ssp)

Full version of algo: with occasional random jumps
Why? To make the matrix irreducible

Full Algorithm

- With probability \( 1-c \), fly-out to a random node
- Then, we have
  \[
  \mathbf{p} = c \mathbf{B} \mathbf{p} + (1-c)/n \mathbf{1} \implies \mathbf{p} = (1-c)/n \left( \mathbf{I} - c \mathbf{B} \right)^{-1} \mathbf{1}
  \]
Parenthesis: intuition behind eigenvectors

Formal definition

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

\[ Ax = \lambda x \]

CLOSELY related to singular values:

Property #1: Eigen- vs singular-values

If 

\[ B_{[n \times m]} = U_{[n \times r]} \Lambda_{[r \times r]} (V_{[m \times r]})^T \]

then \( A = (B^TB) \) is symmetric and 

C(4): \( B^TB v_i = \lambda_i^2 v_i \)

ie, \( v_1, v_2, \ldots \): eigenvectors of \( A = (B^TB) \)
Property #2
- If $A_{|nxn|}$ is a real, symmetric matrix
  - Then it has $n$ real eigenvalues

(if $A$ is not symmetric, some eigenvalues may be complex)

Property #3
- If $A_{|nxn|}$ is a real, symmetric matrix
  - Then it has $n$ real eigenvalues
  - And they agree with its $n$ singular values, except possibly for the sign

Intuition
- $A$ as vector transformation

\[
\begin{bmatrix}
2 \\
1
\end{bmatrix}
= \begin{bmatrix}
2 & 1 & 3 \\
1 & 0
\end{bmatrix}
\begin{bmatrix}
x \\
x'
\end{bmatrix}
\]
**Intuition**

- By defn., eigenvectors remain parallel to themselves ("fixed points")

\[
\lambda_1 \begin{bmatrix} 0.52 \\ 0.85 \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} 0.52 \\ 0.85 \end{bmatrix}
\]

**Convergence**

- Usually, fast:
Convergence

- Usually, fast:
- depends on ratio $\lambda_1 : \lambda_2$

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!!

Query feedbacks

Idea #1: consider a function for the CDF
cummulative distr. function), eg., 6-th
degree polynomial (or splines, or anything
else)
count, so far

year

For example
F(x) = # movies made until year 'x'
= a_1 + a_2 * x + a_3 * x^2 + ... a_7 * x^6
GREAT idea #2: adapt your model, as you see the actual counts of the actual queries.
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:
Query feedbacks

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

\[
\begin{bmatrix}
X_{11} & X_{12} & \cdots \\
X_{21} & X_{22} & \cdots \\
\vdots & \vdots & \ddots
\end{bmatrix}
\begin{bmatrix}
a_1 \\
a_2 \\
\vdots
\end{bmatrix}
= 
\begin{bmatrix}
b_1 \\
b_2 \\
\vdots
\end{bmatrix}
\]

Query feedbacks

For example, for query 'find the count of movies during (1920-1932) :

\[ a_1 + a_2 \cdot 1932 + a_3 \cdot 1932^2 + \ldots \]

\[ - (a_1 + a_2 \cdot 1920 + a_3 \cdot 1920^2 + \ldots) \]

\[
\begin{bmatrix}
X_{11} & X_{12} & \cdots \\
X_{21} & X_{22} & \cdots \\
\vdots & \vdots & \ddots
\end{bmatrix}
\begin{bmatrix}
a_1 \\
a_2 \\
\vdots
\end{bmatrix}
= 
\begin{bmatrix}
b_1 \\
b_2 \\
\vdots
\end{bmatrix}
\]

Query feedbacks

And thus \( X_{11} = 0; X_{12} = 1932-1920, \) etc

\[
\begin{bmatrix}
X_{11} & X_{12} & \cdots \\
X_{21} & X_{22} & \cdots \\
\vdots & \vdots & \ddots
\end{bmatrix}
\begin{bmatrix}
a_1 \\
a_2 \\
\vdots
\end{bmatrix}
= 
\begin{bmatrix}
b_1 \\
b_2 \\
\vdots
\end{bmatrix}
\]
Query feedbacks

In matrix form:

\[
\begin{bmatrix}
X_1 & a_1 \\
\vdots & \vdots \\
X_n & a_n
\end{bmatrix}
\begin{bmatrix}
a_1 \\
\vdots \\
a_n
\end{bmatrix}
= 
\begin{bmatrix}
b_1 \\
\vdots \\
b_n
\end{bmatrix}
\]

1st query

n-th query

Query feedbacks

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:
Query feedbacks - enhancements

Intuition: least squares fit

![Graph showing least squares fit with new query](image)

---

Query feedbacks - enhancements

Intuition: least squares fit

![Graph showing least squares fit with new query](image)

---

Query feedbacks - enhancements

Intuition: least squares fit

![Graph showing least squares fit with new query](image)
Query feedbacks - enhancements
the new coefficients can be quickly computed from the old ones, plus statistics in a (7x7) matrix
(no need to know the details, although the RLS is a brilliant method)

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