Parallel and Distributed Computing: Singular Value Decomposition in the context of massive online Latent Semantic Indexing

Bahena, Zavaleta

CINVESTAV - ORACLE

August, 2015

SVD Decomposition

$A = U \Sigma V^T$

- U and V are orthogonal matrices
- Σ is a diagonal matrix that contain the *singular values* σ_i of A; $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_n \geq 0$

In the 20th century, focus was on obtaining efficient algorithms for SVD computation

- Several implementations of SVD solvers
  - SVDPACK(C)
  - SVDLIBC
  - LAPACK
  - ...
- Gensim contains the SVD solver that best matches our problem (as far as we could investigate)

Latent Semantic Indexing

Documents are encoded using the Vector Space Model

- A matrix A of size $m \times n$
- m indexed terms
- n documents

The VSM can not handle ambiguity:

- Synonymy
- Polysemy

Latent Semantic Indexing

	d1	d2	d3	d4	d5	d6
ship	1	0	1	0	0	0
boat	0	1	0	0	0	0
ocean	1	1	0	0	0	0
voyage	1	0	0	1	1	0
trip	0	0	0	1	0	1

Table: Terms-Documents matrix

Latent Semantic Indexing and the Truncated SVD

According to Deerwester et al. there is a latent space where documents and indexed terms are expresed with r features.

- $A = U\Sigma V^T$
- U is the matrix of terms
- V is the matrix of documents
- Σ contains r singular values

The smallest singular values are actually noise and we can get rid of them.

- By doing so we get $A_k = U_k \Sigma_k V_k^T$
- That allows to compare documents against user queries (pseudo-documents), addressing the polysemy/synonymy issues.
- This step is justified by Eckart-Young-Mirsky theorem

$$\min_{\{Z|rank(Z)=k\}} \|A - Z\|_2 = \|A - A_k\|_2 = \sigma_{k+1} \tag{1}$$

LSI: A quick example

$$U = \begin{pmatrix} -0.45 & -0.30 & 0.57 & 0.58 & 0.25 \\ -0.13 & -0.33 & -0.59 & 0 & 0.73 \\ -0.48 & -0.51 & -0.37 & 0 & -0.61 \\ -0.70 & 0.35 & 0.15 & -0.58 & 0.16 \\ -0.26 & 0.65 & -0.41 & 0.58 & -0.09 \end{pmatrix} \tag{2}$$

$$\Sigma = \begin{pmatrix} 2.16 & 0 & 0 & 0 & 0 \\ 0 & 1.59 & 0 & 0 & 0 \\ 0 & 0 & 1.28 & 0 & 0 \\ 0 & 0 & 0 & 1.00 & 0 \\ 0 & 0 & 0 & 0 & 0.39 \end{pmatrix} \tag{3}$$

$$V^T = \begin{pmatrix} -0.75 & -0.28 & -0.20 & -0.45 & -0.33 & -0.12 \\ -0.29 & -0.53 & -0.19 & 0.63 & 0.22 & 0.41 \\ 0.28 & -0.75 & 0.45 & -0.20 & 0.12 & -0.33 \\ 0 & 0 & 0.58 & 0 & -0.58 & 0.58 \\ -0.53 & 0.29 & 0.63 & 0.19 & 0.41 & -0.22 \end{pmatrix} \tag{4}$$

LSI: A quick example

$$\Sigma_2 = \begin{pmatrix} 2.16 & 0.00 \\ 0.00 & 1.59 \end{pmatrix} \tag{5}$$

$$A_2 = \begin{pmatrix} 1.0199 & 0.0060 & 1.0112 & 0.0095 & 0.0069 & -0.0047 \\ 0.0004 & 1.0057 & -0.0046 & 0.0009 & 0.0033 & 0.0052 \\ 1.0062 & 1.0063 & -0.0016 & 0.0052 & 0.0094 & 0.0006 \\ 0.9933 & 0.0025 & -0.0140 & 1.0045 & 1.0064 & -0.0039 \\ -0.0069 & -0.0071 & -0.0059 & 1.0021 & -0.0011 & 1.0084 \end{pmatrix} \tag{6}$$

$$\|A - A_k\|_2 = 1.28 \tag{7}$$

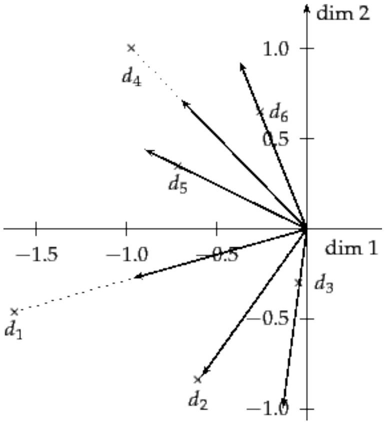
$$\Sigma_2 V_2^T = \begin{pmatrix} -1.62 & -0.60 & -0.44 & -0.97 & -0.70 & -0.26 \\ -0.46 & -0.84 & -0.30 & 1.00 & 0.35 & 0.65 \end{pmatrix} \tag{8}$$

LSI: A quick example

$$\mathbf{q} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{9}$$

$$\mathbf{q_k} = {\Sigma_k}^{-1} \, {U_k}^T \, \mathbf{q} = \begin{pmatrix} -0.06 \\ -0.21 \end{pmatrix} \tag{10}$$

LSI: A quick example

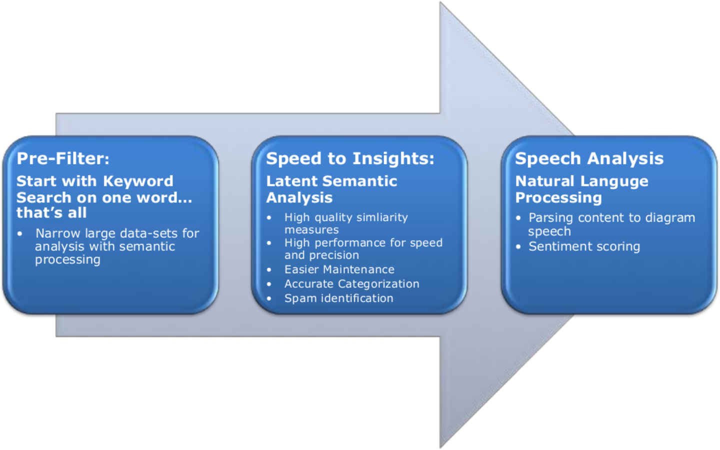


How did we come up with this idea?

Looking for ideas for a Msc. Thesis

- Possibly apporting something to Oracle :)
- Oracle bought *Collaborative Intellect Inc.* in 2012
    - Analytics engine that processes conversations from Social Networks
    - Companies using this product can know if user's posts in Social Networks are associated with certain topics they define (which can be their products, services, etc).
    - Tens of Millions of conversations in a day!
    - Social Networks produce Hundreds of Millions!
    - Requires LSI!

How did we come up with this idea?

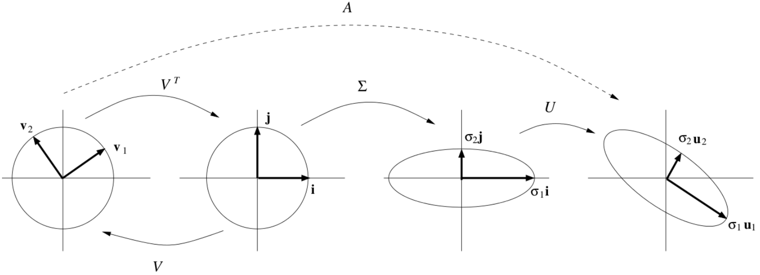


Statement of the Problem

Given the fact that modern LSI applications require end to end processing of hundreds of millions of documents every day, in a streamed updatable fashion, how can the particular SVD step be optimized through parallel or distributed computing?

What does $A = U\Sigma V^T$ mean?

Geometric interpretation



SVD as an Eigenproblem

Algebraic and Geometric Proofs

A whole chapter of the report, check it out!

The eigenproblem, solved for symmetric matrices

$B^{n \times n} = Q \Lambda Q^T$

$A^T A$ is symmetric!

$$A = U \Sigma V^T \quad \Longleftrightarrow \quad A^T A = V \Sigma^2 V^T$$

$A A^T$ is symmetric too! (used in GenSim)

$$A = U \Sigma V^T \quad \Longleftrightarrow \quad A A^T = U \Sigma^2 U^T$$

Is SVD problem well conditioned?

A problem is well conditioned if ...

$x \simeq (x + \delta) \implies f(x) \simeq f(x + \delta)$

Weyl's Theorem

$|\tilde{\sigma}_i - \sigma_i| \leq \|E\|_2 \quad \ni \quad \tilde{A} = A + E$

Wedin's Theorem

The singular vectors subproblem is not well conditioned in theory, but it has bounds.

In practice we are fine

We do not need singular vectors by themselves ($\mathbf{v} = \Sigma^{-1} U^T \mathbf{x}$).

Algorithm 1: The Single-Vector Lanczos Algorithm

Input : A matrix $A^{m \times n}$ and a truncation factor k

Output: The k singular values and its associated right singular vectors of A. Both are numeric approximations.

Use Lanczos Tridiagonalization Step to generate a family of c symmetric tridiagonal matrices $\{T_j\}$ for $A^T A$ $(c > k)$

Compute the eigenvalues and eigenvectors of T_k using the (implicit) QL Method.

For each computed λ_i of T_k, calculate the associated unit eigenvector $\mathbf{z_i}$ such that $T_k \mathbf{z_i} = \lambda_i \mathbf{z_i}$.

For each calculated eigenvector $\mathbf{z_i}$ of T_k, compute the Ritz vectors $v_i = Q_c \mathbf{z_i}$ as an approximation to the i-th eigenvector of $A^T A$. Note that the matrix Q_c is a side product of the first step.

return $(\{\sqrt{\lambda_1}, \sqrt{\lambda_2}, \cdots, \sqrt{\lambda_k}\}, \{\mathbf{v_1}, \mathbf{v_2}, \cdots, \mathbf{v_k}\})$

Algorithm 2: Lanczos Tridiagonalization Step (sparse,2,Q)

Input : A unit vector $\mathbf{q_1} \in \mathbb{R}^n$ and a symmetric matrix $B^{n \times n}$

Output: The sequences $\{\alpha_i\}$, $\{\beta_i\}$ and matrix $Q = [\mathbf{q_1}|\mathbf{q_2}|\cdots]$

$k \leftarrow 0, \beta_0 \leftarrow 1, \mathbf{q_0} \leftarrow 0, r_0 \leftarrow \mathbf{q_1}$

while $k = 0 \vee \beta_k \neq 0$ **do**

$$\mathbf{q_{k+1}} \leftarrow \frac{\mathbf{r_k}}{\beta_k}$$

$$k \leftarrow k + 1$$

$$\alpha_k \leftarrow \mathbf{q_k}^T B \mathbf{q_k}$$

$$\mathbf{r_k} \leftarrow B\mathbf{q_k} - \alpha_k q_k - \beta_{k-1}\mathbf{q_{k-1}}$$

$$\beta_k \leftarrow \|\mathbf{r_k}\|_2$$

return $(\{\alpha_i\}, \{\beta_i\}, Q = [\mathbf{q_1}|\mathbf{q_2}|\cdots])$

The T_k from Lanczos Tridiagonalization Step

$$T_k = \begin{bmatrix} \alpha_1 & \beta_1 & \cdots & 0 \\ \beta_1 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \beta_{k-1} \\ 0 & \cdots & \beta_{k-1} & \alpha_k \end{bmatrix}$$

Lanczos Algorithm Reality Check

Is it numerically stable?

No, the tridiagonalization step needs to be rewritten (thanks Paige!).

Is orthogonality really guaranteed?

No, it is lost as β_k becomes small (thanks again Paige!). Selective re-orthogonalization is required (thanks to Parlett and Simon).

What to do then?

Berry's implementation, LASVD/LAS2, consider all aspects above; including performance (parallelization).

LASVD/LAS2 Profiling by Berry

Routine	Library	Description
SPMXV	BLAS level 2	Sparse matrix-vector mult.
IMTQL2 / TRED2	EISPACK	Implicit QL Algorithm.
DAXPY	BLAS level 1	$\mathbf{x} \leftarrow \gamma \mathbf{x} + \mathbf{y}$
DAXPY	BLAS level 1	$\mathbf{x} \leftarrow \mathbf{y}$
DDOT	BLAS level 1	$\mathbf{x} \cdot \mathbf{y}$

	Alliant FX/80		Cray-2S/4-128	
Routine	Speedup	%CPU Time	Speedup	%CPU Time
SPMXV	3	27%	-	72%
IMTQL2	4.3	14%	-	12%
DAXPY	5	17%	-	-
DCOPY	3.6	20%	-	-
DDOT	7.7	2%	-	-

SVDPACK $\Longrightarrow$ SVDLIBC: lost parallelism

- Berry's SVDPACK LAS2: SPMXV (BLAS?), IMTQL2 (EISPACK)
- Berry ports to SVDPACKC: opa/opb (user), IMTQL2 (serial!)
- Rohde new skin of SVDLIBC: opa/opb (serial!), IMTQL2 (serial!).
- Radim's python wrapper SPARSESVD : same all
- Radim's GenSim uses a serial LAS2 routine!

Algorithm 3: Distributed-SVD: Distributed SVD for LSI (global)

Input : Truncation factor k, queue of jobs $A = [A_1, A_2, \ldots]$

Output: Matrices $U^{m \times k}$ and $\Sigma^{k \times k}$, from the SVD decomp. of A

for all *(node i in cluster)* **do**

  $B_i \leftarrow$ subset of the queue of jobs $[A_1, A_2, \ldots]$

  $P_i = (U_i, \Sigma_i) \leftarrow \text{SVD-Node}(k, B_i)$

$(U, \Sigma) \leftarrow \text{Reduce}(\text{Merge-SVD}, [P_1, P_2, \ldots])$

return (U, Σ)

Algorithm 4: SVD-Node: Distributed SVD for LSI (node)

Input : Truncation factor k, queue of jobs $A_1, A_2, \ldots$

Output: Matrices $U^{m \times k}$ and $\Sigma^{k \times k}$, from the SVD of $[A_1, A_2, \ldots]$

$P = (U, \Sigma) \leftarrow 0^{m \times k} 0^{k \times k}$

for *each job* A_i **do**

  $P^{'} = (U^{'}, \Sigma^{'}) \leftarrow$ Basecase-SVD(k, A_i)

  $P = (U^{m \times k}, \Sigma^{k \times k}) \leftarrow$ Merge-SVD$(k, P, P^{'})$

return (U, Σ)

Algorithm 5: Merge-SVD: Merge of two SVD factorizations

Input : Truncation factor k, decay factor γ, $P_1 = (U_1^{m\times k_1}, \Sigma_1^{k_1\times k_1})$,
$P_2 = (U_2^{m\times k_2}, \Sigma_1^{k_2\times k_2})$

Output: $(U^{m\times k}, \Sigma^{k\times k})$

$Z^{k_1\times k_2} \leftarrow\ {U_1}^T\ U_2$

$U^{'}R \xleftarrow{QR} U_2 - U_1 Z$

$U_R \Sigma\, {V_R}^T \ \xleftarrow{SVD_k} \begin{bmatrix} \gamma\Sigma_1 & Z\Sigma_2 \\ 0 & R\Sigma_2 \end{bmatrix}^{(k_1+k_2)\times(k_1+k_2)}$

$\begin{bmatrix} R_1^{k_1\times k} \\ R_2^{k2\times k} \end{bmatrix} = U_R$

$U \leftarrow U_1 R_1 + U^{'} R_2$

return (U, Σ)

Distributed Algorithm

Complexity is according to him $O(mk^2)$

- Processed Wikipedia (100K terms x 3.2M documents) in 8.5 hrs on a single machine, with an reduction to 2.5 hours in a 4 nodes cluster
- Using Intel's optimized BLAS/LAPACK in the merge algorithm, it took 1 hour and 41 minutes
- Another custom implementation of another SVD algorithm took 109 hours (vs his serial version of 8.5 hours). hours

Conclusions

We found that the following is still required:

- Need a modern sensible experiment that compares GenSim against other modern SVD parallel/distributed algorithms (which may not focus exclusively LSI problem).
  - Compare against Mahout, SLEPc, LingPipe, GraphLab, etc.
  - Use experiments similar to those that Rehurek did.
  - Need to use much more documents and more nodes in the cluster
  - Could get benchmark matrices used by Oracle's Product.
- For the case of Rehurek's, profiling experiments similar to those made by Berry, could help identify bottlenecks
  - Base case SVD
  - Merge Algorithm

Conclusions

- Need to research whether Rehurek's merge algorithm is optimal or not
- Need to try other merge alternatives
- Need an experiment with different implementations of the base case SVD solver
  - Could try parallel versions of SVDLIBC (BLAS/LAPACK)