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

Bahena, Zavaleta

CINVESTAV - ORACLE

August, 2015

Matrix Decompositions

Decomposition	**Expression**	**Components**
LU Decomposition	$A = LU$	L is a lower triangular matrix; U is a upper triangular matrix
Rank Factorization	$A = CF$	C is an $m \times r$ full column rank matrix; F is a $r \times n$ full row rank matrix
Cholesky Decomposition	$A = U^T U$	U is upper triangular with positive diagonal entries
QR Decomposition	$A = QR$	Q is an orthogonal matrix mxm; R is an upper triangular matrix $m \times n$
Eigendecomposition	$A = Q\Lambda Q^{-1}$	D is a diagonal matrix formed from eigenvalues of A; V are the corresponding eigenvectors

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:

- Synonymity
- 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 $A_k = U_k \Sigma_k V_k^T$
- This step is justified by Eckart-Young-Mirsky theorem

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

Latent Semantic Indexing and the Truncated SVD

1) In this space, to compare two terms:

$$A_k A_k^T = U_k \Sigma_k^2 U_k^T \tag{2}$$

2) To compare two documents:

$$A_k^T A_k = V_k \Sigma_k^2 V_k^T \tag{3}$$

3) The association between a term i and a document j is given by the value in position ij in matrix A_k

LSI: A quick example

$$U = \begin{pmatrix} -0.45 & -0.30 & 0.57 & 0.58 & 0.25 \\ -0.13 & -0.33 & -0.59 & 0 & 0.73 \\ -0.48 & -0.51 & -0.37 & 0 & -0.61 \\ -0.70 & 0.35 & 0.15 & -0.58 & 0.16 \\ -0.26 & 0.65 & -0.41 & 0.58 & -0.09 \end{pmatrix} \tag{4}$$

$$\Sigma = \begin{pmatrix} 2.16 & 0 & 0 & 0 & 0 \\ 0 & 1.59 & 0 & 0 & 0 \\ 0 & 0 & 1.28 & 0 & 0 \\ 0 & 0 & 0 & 1.00 & 0 \\ 0 & 0 & 0 & 0 & 0.39 \end{pmatrix} \tag{5}$$

$$V^T = \begin{pmatrix} -0.75 & -0.28 & -0.20 & -0.45 & -0.33 & -0.12 \\ -0.29 & -0.53 & -0.19 & 0.63 & 0.22 & 0.41 \\ 0.28 & -0.75 & 0.45 & -0.20 & 0.12 & -0.33 \\ 0 & 0 & 0.58 & 0 & -0.58 & 0.58 \\ -0.53 & 0.29 & 0.63 & 0.19 & 0.41 & -0.22 \end{pmatrix} \tag{6}$$

LSI: A quick example

$$\Sigma_2 = \begin{pmatrix} 2.16 & 0.00 \\ 0.00 & 1.59 \end{pmatrix} \tag{7}$$

$$A_2 = \begin{pmatrix} 1.0199 & 0.0060 & 1.0112 & 0.0095 & 0.0069 & -0.0047 \\ 0.0004 & 1.0057 & -0.0046 & 0.0009 & 0.0033 & 0.0052 \\ 1.0062 & 1.0063 & -0.0016 & 0.0052 & 0.0094 & 0.0006 \\ 0.9933 & 0.0025 & -0.0140 & 1.0045 & 1.0064 & -0.0039 \\ -0.0069 & -0.0071 & -0.0059 & 1.0021 & -0.0011 & 1.0084 \end{pmatrix} \tag{8}$$

$$\|A - A_k\|_2 = 1.28 \tag{9}$$

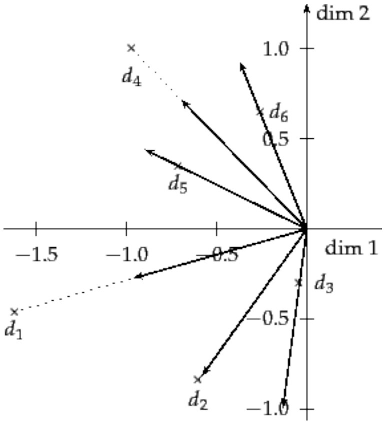
$$\Sigma_2 V_2^T = \begin{pmatrix} -1.62 & -0.60 & -0.44 & -0.97 & -0.70 & -0.26 \\ -0.46 & -0.84 & -0.30 & 1.00 & 0.35 & 0.65 \end{pmatrix} \tag{10}$$

LSI: A quick example

$$q = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{11}$$

$$q_k = \begin{pmatrix} -0.06 \\ -0.21 \end{pmatrix} \tag{12}$$

LSI: A quick example

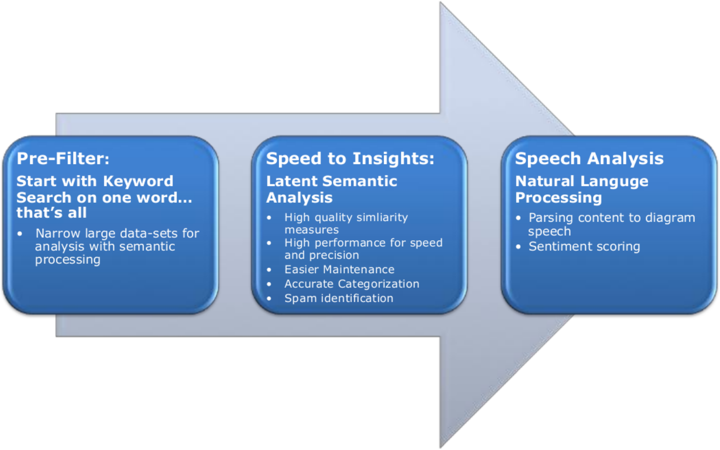


How did we come up with this idea?

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
  - Users of this product can know if user's posts in Social Networks are associated with certain topics
  - 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?

Algorithm 1: Distributed-SVD: Distributed SVD for LSI (global)

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

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 $[A_1, A_2, \dots]$

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

$(U, \Sigma) \leftarrow$ Reduce(Merge-SVD, $[P_1, P_2, \dots]$)

return (U, Σ)

Algorithm 2: 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**

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

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

return (U, Σ)

Algorithm 3: 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 (3.2M documents, 100K words) in 2.5 hours in a 4 nodes cluster
- Using BLAS/LAPACK in the merge algorithm it took 1 hour and 41 minutes
- Another custom implementation of another SVD algorithm took 109 hours

Conclusions

We found that the following is still required:

- Enhance memory complexity study of Rehurek's algorithm
- Need a modern sensible experiment that compares best parallel/distributed algorithms taking into account the architecture of modern computers
  - 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 alternative
- 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)