Irlbpy – A fast partial SVD for Python

J. Baglama [1], M. Kane [2], and B. Lewis [3]

[1]Department of Mathematics, The University of Rhode Island
[2]Department of Biostatistics, Yale University
[3]Paradigm4

Overview

The singular value decomposition (SVD) is central to many important data-analytic methods and applications including principal component analysis, canonical correlation analysis, correspondence analysis, latent semantic indexing and non-linear iterative partial least squares to name a few. However, numerical implementations of the SVD are intensive, generally incurring a computational complexity of $O(m^2n + n^3)$ for an $m \times n$ matrix with m greater than n. As a result, data scientist's have fewer analytical tools to understand the structure of data as those data become large and the resulting computational cost becomes too expensive to carry out.

However, many of these methods and applications only require a few singular values and corresponding singular vectors. With this in mind, some numerical analysts have focused on the developement of *truncated* SVD algorithms that calculate the largest or smallest singular values and vectors for a matrix. The *implicitly restarted Lanczos bidiagonalization* (IRLB) algorithm [BR06a] is a fast and efficient approach for calculating truncated singular values and vectors, generally scaling linearly in the size of the matrix. This innovative algorithm calculates a key numerical decomposition for statistical and machine learning procedures and it effectively scales to massive data. Standard analyses can now be applied to much larger data sets than previously possible. Furthermore, the approach used by the IRLB algorithm suggests a general class of new approximation algorithms that can be used for big-data challenges where current approaches fall short.

While IRLB implementations have existed for some time now in both Matlab [BR06b] and R [BR12] the scientific Python community has not enjoyed the computational savings offered by the algorithm until now. This poster introduces the irlbpy package for Python, a pip-installable open-source implementation of the IRLB algorithm that is available from github at https://github.com/bwlewis/irlbpy. The package is compatible with numpy for dense matrices challenges scipy for sparse matrices. The rest of this poster gives an overview of the algorithm and benchmarks performance. The benchmarks were performed on a Mac Book Pro with a quad-core 2.7 GHz Intel Core i7 with 16 GB of 1600 MHz DDR3 RAM running Python version 2.7.3, Numpy version 1.7.0, and SciPy version 0.12.0.

Partial SVD Definition

The SVD of a positive, semidefinite matrix $X \in \mathcal{R}^{m \times n}$ with $m \geq n$ is a factorization of the form

$$X = U\Sigma V^T \tag{1}$$

where U is an $m \times m$ unitary matrix, Σ is an $m \times n$ rectangular diagonal matrix, and V is an $n \times n$ unitary matrix. The diagonal values of Σ, $\sigma_1, ...\sigma_n \geq 0$ and by convention they are sorted in decreasing order with the corresponding left-singular vectors $(u_1, ..., u_m)$ and right-singular vectors, $(v_1, ..., v_n)$ ordered similarly.

Alternatively, the factorization in Equation 1 can be written as

$$X = \sum_{j=1}^{n} \sigma_j u_j v_j^T. \tag{2}$$

where again u_j and v_j represent the jth column of U and V respectively. Based on this formulation, we define the *partial* SVD as

$$X_k = \sum_{j=1}^{k} \sigma_j u_j v_j^T. \tag{3}$$

X_k can be thought of as an approximation of the original matrix X. As k increases to n the approximation becomes more accurate. Alternatively, a singular value along with its corresponding left and right singular vectors capture salient information about the matrix. The amount of "signal" is proportional to the size of the singular value.

Properties of the IRLB algorithm

1. In practice the IRLB algorithm scales linearly in the size of the data. This gives it a tremendous advantage over the traditional SVD when only the largest singular value information is needed.
2. The algorithms computational cost as the number of singular values required increases.
3. The IRLB algorithm and irlbpy implementation are compatible with both dense numpy matrices and sparse scipy matrices.
4. Implementations of the IRLB algorithm have been successfully used on an incidence matrix based on the Netflix data set. The matrix was composed of approximately 500,000 rows (movies) by 18,000 columns (viewers) with 100 million nonzero elements. The singular values were calculated on a laptop computer and required only 20 minutes of computing time.
5. A distributed version of the IRLB algorithm has been implemented for SciDB [Par13]. This version calculates the truncated SVD on matrices with approximately 50 million rows, 40 million columns and 4 billion non-zero elements on a cluster of 4 Xeon machines (Sandy Bridge, 16 cores, 128 GB RAM) in approximately five minutes per singular value and vector.

References

[BR06a] J. Baglama and L. Reichel. Restarted block lanczos bidiagonalization methods. *Numerical Algorithms*, 43:251–272, 2006.

[BR06b] Jim Baglama and Lothar Reichel. *irlba*, 2006. Matlab package version 1.0.

[BR12] Jim Baglama and Lothar Reichel. *irlba: Fast partial SVD by implicitly-restarted Lanczos bidiagonalization*, 2012. R package version 1.0.2, Implemented by Bryan Lewis.

[Par13] Paradigm4, Waltham, MA. *SciDB: Open Source Data Management and Analytics Software for Scientific Research*, 2013.

Dense Matrix Comparison

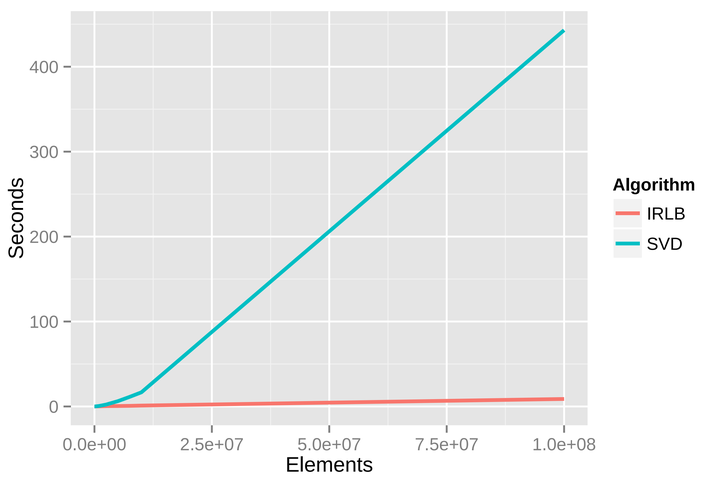

Figure 1: Performance comparison of the IRLB and the numpy implementation of the SVD calculating the 10 larges singular values and vectors.

Dense Matrix Scaling

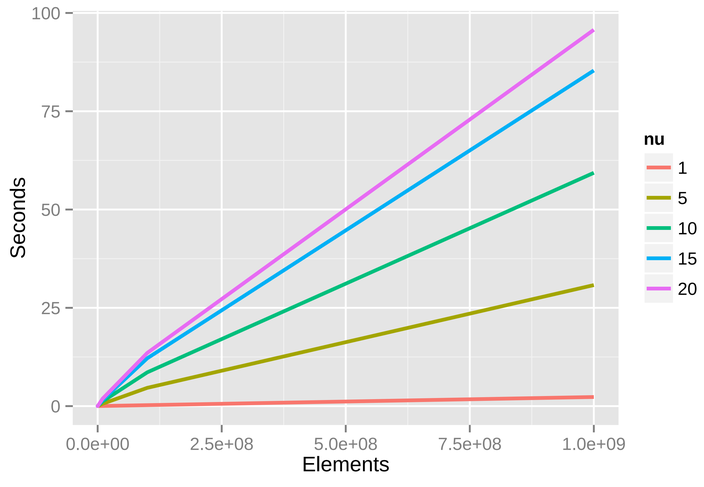

Figure 2: The time required to calculate the IRLB on dense matrices for specified values of nu (the number of singular vectors).

Sparse Matrix Scaling

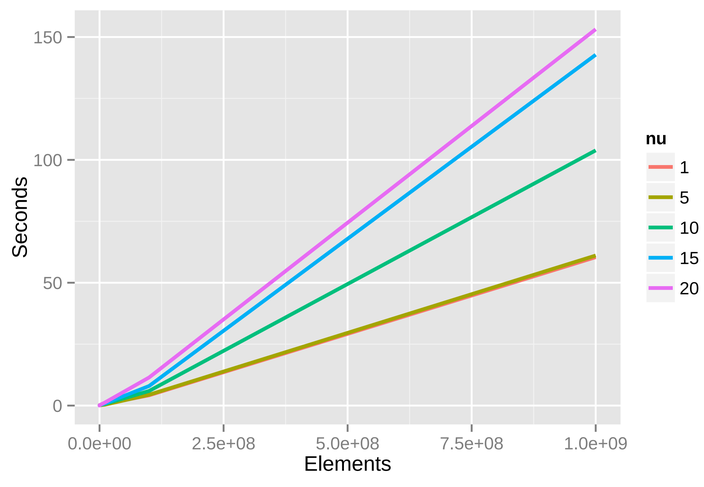

Figure 3: The time required to calculate the IRLB on sparse matrices for specified values of nu (the number of singular vectors).