Google

Code

Archive

redsvd - English.wiki

Export to GitHub

---

redsvd is a C++ library for solving several matrix decompositions including singular value decomposition (SVD), principal component analysis (PCA), and eigen value decomposition. redsvd can handle very large matrix efficiently, and optimized for a truncated SVD of sparse matrices. For example, redsvd can compute a truncated SVD with top 20 singular values for a 100K x 100K matrix with 1M nonzero entries in less than one second.

The algorithm is based on the randomized algorithm for computing large-scale SVD. Although it uses randomized matrices, the results is very accurate with very high probability.

- Nicolas Tessore made a header only version of redsvd, which is useful for manys https://github.com/ntessore/redsvd-h

How to Use

Currently, redsvd is supported in Linux Ubuntu

First, install eigen3 eigen3.0-beta1

Next, download the latest tarball of redsvd from Downloads.

Finally type the following commands. ```

tar xvjf redsvd-x.x.x.tar.bz2 cd redsvd-x.x.x ./waf configure ./waf sudo ./waf install ```

Now the program redsvd is installed.

If you want to install redsvd in local environment, type the following commands. ```

tar xvjf redsvd-x.x.x.tar.bz2 cd redsvd-x.x.x ./waf configure --prefix="/your/local/path" ./waf ./waf install ```

You can see the usage of redsvd by calling redsvd without options. ```

redsvd usage: redsvd --input=string --output=string [options] ...

redsvd supports the following format types (one line for each row)

[format=dense] (+\n)+ [format=sparse] ((colum_id:value)+\n)+ Example:

redsvd -i imat -o omat -r 10 -f dense compuate SVD for a dense matrix in imat and output omat.U omat.V, and omat.S with the 10 largest eigen values/vectors redsvd -i imat -o omat -r 3 -f sparse -m PCA compuate PCA for a sparse matrix in imat and output omat.PC omat.SCORE with the 3 largest principal components

options: -i, --input input file (string) -o, --output output file's prefix (string) -r, --rank rank (int [=10]) -f, --format format type (dense|sparse) See example. (string [=dense]) -m, --method method (SVD|PCA|SymEigen) (string [=SVD]) ```

```

*cat file1 1.0 2.0 3.0 4.0 5.0 -2.0 -1.0 0.0 1.0 2.0 1.0 -2.0 3.0 -5.0 7.0 redsvd -i file1 -o file1 -r 2 -f 2 read dense matrix from file1 ... 0.000102997 sec. SVD for a dense matrix rows: 3 cols: 5 rank: 2 compute SVD... -4.69685e-05 sec. write matrix to file1(.U|.S|.V) ... 0.018553 sec. finished. cat file1.U -0.372291 -0.926217 -0.005434 -0.061765 -0.928100 +0.371897 cat file1.V -0.411950 -0.186912 -0.031819 -0.450366 -0.441672 -0.257618 +0.432198 -0.806197 -0.668891 -0.214273 cat file1.S +9.176333 +6.647099 ```*

You can also use a sparse matrix representation. ```

*cat news20.binary 1:0.016563 3:0.016563 6:0.016563 ... ...*

*redsvd -i news20.binary -o news20 -f 1 -r 10 read sparse matrix from news20.binary ... 4.84901 sec. rows: 19954 cols: 1355192 nonzero: 9097916 rank: 10 compute SVD... 2.52615 sec. write matrix to news20(.U|.S|.V) ... 5.6056 sec. finished. cat news20.S +17.973207 +2.556800 +2.460566 +2.135978 +2.022737 +1.931362 +1.927033 +1.853175 +1.770231 +1.764138 ```*

# Experimental Result

See the detailed result redsvd_result.pdf

You can reproduce these result by ```

*performanceTest.sh accuracyTest.sh ```*

## Environment
```

- Intel(R) Core(TM)2 Quad CPU Q9550 @ 2.83GHz 8G

For dense matrices

| n | m | rank | time (msec) | |:----|:-------|:-----|:-----------| | 500 | 100 | 10 | 0.76 | | 500 | 1000 | 10 | 3.24 | | 500 | 10000 | 10 | 32.3 | | 500 | 100000 | 10 | 306.3 |

| n | m | rank | time (msec) | |:----|:-------|:-----|:-----------| | 500 | 100 | 100 | 12.3 | | 500 | 1000 | 1000 | 987.5 | | 500 | 10000 | 1000 | 3850.0 | | 500 | 100000 | 1000 | 32824.3 |

| n | m | rank | time (msec) | |:------|:------|:-----|:-----------| | 100 | 100 | 10 | 0.20 | | 1000 | 1000 | 10 | 6.34 | | 10000 | 10000 | 10 | 578 |

| n | m | rank | time (msec) | |:------|:------|:-----|:-----------| | 100 | 100 | 500 | 8.67 | | 1000 | 1000 | 500 | 8654 | | 10000 | 10000 | 500 | 45001 |

For sparse matrices

| n | m | rank | nonzero ratio (%) | time (msec) | |:-------|:-------|:-----|:-----------------|:------------| | 100 | 100 | 10 | 0.1 | 0.31 | | 1000 | 1000 | 10 | 0.1 | 1.17 | | 10000 | 10000 | 10 | 0.1 | 22.5 | | 100000 | 100000 | 10 | 0.1 | 1679.9 |

| n | m | rank | nonzero ratio (%) | time (msec) | |:-------|:-------|:-----|:-----------------|:------------| | 100 | 100 | 10 | 1.0 | 0.16 | | 1000 | 1000 | 10 | 1.0 | 2.0 | | 10000 | 10000 | 10 | 1.0 | 124.1 | | 100000 | 100000 | 10 | 1.0 | 12603.4 |

Latent Semantic Analyasis (SVD for doc-term matrix) of English Wikipedia

The target rank is 10

| # doc | # word | # total words | time (msec) | |:------|:-------|:-------------|:-----------| | 3560 | 27106 | 172823 | 27 | | 46857 | 147144 | 2418406 | 390 | | 118110 | 261495 | 6142438 | 1073 | | 233717 | 402239 | 12026852 | 1993 |

Inside of redsvd

The code redsvd.hpp is the core part of redsvd and self explanatory.

Let A be a matrix to be analyzed with n rows and m columns, and r be the ranks of a truncated SVD (if you choose r = min(n, m), then this is the original SVD).

First a random Gaussian matrix O with m rows and r columns is sampled and computes $Y = A^t O$. Then apply the Gram-Schmidt process to Y so that each column of Y is ortho-normalized. Then we

compute B = AY with n rows and r columns. Although the size of Y is much smaller than that of A, Y holds the informatin of A; that is $AYY^t = A$. Intuitively, the row informatin is compresed by Y and can be decompressed by Y^t

Similarly, we take another random Gaussian matrix P with r rows and r columns, and compute Z = BP. As in the previous case, the columns of Z are ortho-normalized by the Gram-Schmidt process. ZZ^tt B = B. Then compute C = Z^t B.

Finally we compute SVD of C using the traditional SVD solver, and obtain $C = USV^t$ where U and V are orthonormal matrices, and S is the diagonal matrix whose entriesa are singular values. Since a matrix C is very small, this time is negligible.

Now A is decomposed as $A = AYY^t = BY^t = ZZ^tBY^t = ZCY^t = ZUSV^tY^t$. Both ZU and YV are othornormal, and ZU is the left singular vectors and YV is the right singular vector. S is the diagonal matrix with singular values.

Thanks

- redsvd uses Eigen 3.0-beta1
- redsvd uses the algorithm based on the randomized algorithm described in the following paper. However, although the algorithm in redsvd samples both rows and columns, the original algorithm samples in one way (this would be because the performance analysis becomes complex).
    1. "Finding structure with randomness: Stochastic algorithms for constructing approximate matrix decompositions", N. Halko, P.G. Martinsson, J. Tropp, arXiv 0909.4061
- redsvd is developed in 20% project of Preferred Infrastructure(Japanese)