Randomized Singular Value Decomposition and Image Compression

Lucas de La Brosse, Anurag Miglani, Thomas Zamojski

ENSAI

14 Apr. 2016

1 Introduction

Introduction

- Big Data can be also about speed of processing data!
- Images from airplanes and satellites are about 1-3GB (up to thousands of channels), and go in at a rate of about 10 seconds.
- To deal with this speed, use compressive sensing: compress the image in realtime.
- We will look at a compressive algorithm used in such imaging, applied to a usual rgb image.

Singular Value Decomposition

SVD

Given a $m \times n$ matrix A, a singular value decomposition finds a $m \times m$ orthogonal matrix U, a $n \times n$ orthogonal matrix V and a nonnegative diagonal $m \times n$ matrix D such that

$$A = UDV^T.$$

- SVD is one of two classical methods for PCA.
- Truncating all but the first k singular values gives the best rank k approximation to A:

$$A \approx UD_kV^T.$$

- SVD is slow. Approximate via probabilistic methods.

Randomized SVD

- Randomized sample of the image of A: W a $n \times k$ random matrix for a small k, take $Y = AW$.
- $Q =$ orthogonalize Y. (e.g. econ.-QR-decomp.).
- $A \approx QQ^T A$ is a low-rank approximation.
- Do SVD of $Q^T A$. Actual for speed, can do econ.-QR-decomp. of $A^T Q = \hat{Q}$ and do SVD on the $k \times k$ R-part of it only.

Implementations: Big memory

- BigMemory is an R package allowing for using semi-big matrices.
- Matrices are stored out-of-core, and passed by reference into R, so fast.
- Shortcoming: smallest type is short. We need unsigned char.
- Shortcoming: atomic types.

Implementations: RcppArmadillo

- R language is not designed for numerical linear algebra.
- Relies instead on C/C++ libraries like BLAS, LAPACK.
- Armadillo's C++ package is a template package for linear algebra libraries.
- RcppArmadillo allows R to play nicely with Armadillo.
- We further optimize by installing parallelised libraries like openBLAS.

Code: C++

```
// [[Rcpp::export]]
List myrsvd(S4 bigmat, int k, int p){
  ...
  List desc = bigmat.slot("description");
  ...
  mat A = readBigMatrix(fname, nr, nc);
  mat W = randn(nc,k+p);
  qr_econ(Q,tmp, A * W);
  qr_econ(Qb, R, A.t() * Q);
  svd(U, s, V, R);
  mat Uf = Q * V; Uf = Uf.cols(1,k);
  mat Vf = Qb * U; Vf = Vf.cols(1,k);
  s = s.subvec(1,k);
  return List::create(
      Named("u") = Uf,
      Named("d") = s,
      Named("v") = Vf);
```

Code: R

```
require(bigmemory)
require(Rcpp)
require(RcppArmadillo)

A <- attach.big.matrix("data/pizza.desc")
Adesc <- describe(A)

sourceCpp("src/myrsvd.cpp")
ll <- myrsvd(Adesc,500,10)
```

Comparisons different Algorithms: speed

```
R> res <- svd(as.matrix(A))
utilisateur     système      écoulé
   5083.974     378.077     698.627

R> rer <- rsvd(as.matrix(A),k=1000,p=10,q=0)
utilisateur     système      écoulé
     36.952     278.806      76.876

R> rem <- myrsvd(describe(A),1000,10)
utilisateur     système      écoulé
    228.685      26.574      38.915
```

Comparisons different Algorithms: Accuracy

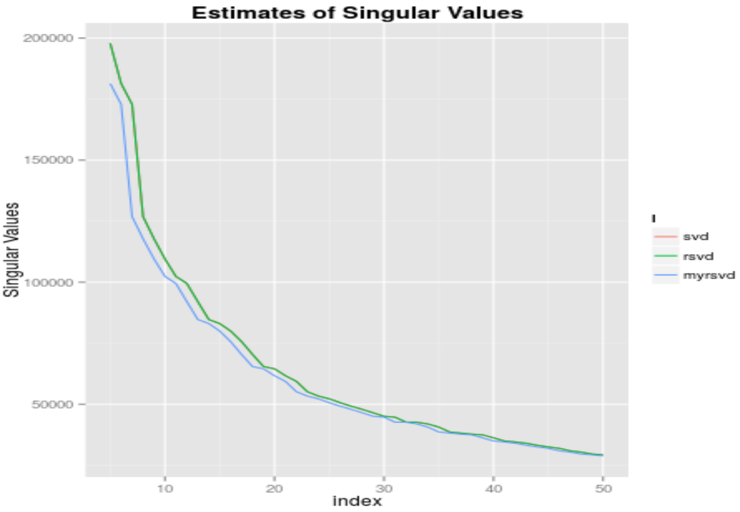

Figure : First singular values according to the 3 algorithms.

Compressed Images

Figure : Original image: 274MB

Figure : Compressed image at k=1000: 41.4MB

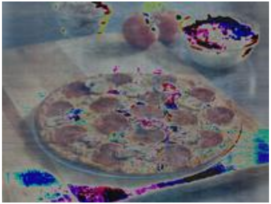

Figure : Compressed image at k=100%: 4.14MB

Figure : Compressed image at k=10: 414KB