Notes on performance-related changes to the s4vd package

Bryan W. Lewis
blewis@illposed.net

June 10, 2015

1 Introduction

This note outlines a few performance improvements and bug fixes affecting the `ssvdBC` and `ssvd` functions.

Portions of the functions that rely on a low-rank singular value decomposition require many fewer arithmetic operations than their original counterparts. The algorithms used by the modified functions, outlined below, are mathematically equivalent to the originals. Numerical clustering results should agree closely to within the limits of machine epsilon and differences in accumulated roundoff errors.

The `ssvdBC` function in the original package contains a minor bug affecting cases when the rank of the approximation matrix is greater than one. In those cases, modified function will produce different clustering results than the original.

2 Modified methods

The following sections describe the changes we made to several portions of the package.

2.1 Partial SVD

The `s4vd` package uses the default R `svd` function to compute a rank-one singular value decomposition (SVD) of the data matrix or deflated data matrix at each step. The default R `svd` function uses a numerical method that computes a full decomposition–much more information than is required by the biclustering method. The modified function uses the `irlba` package[2] to efficiently produce the required low-rank decomposition.

The computational savings of `irlba` over the original `svd` varies with the input data, but can be significant. The modified functions also avoid recomputing the SVD unnecessarily, yielding further computational savings.

The change in partial SVD computation affects the `ssvd` function.

2.2 BIC Optimization Loop

Equations (12) and (13) in the biclustering paper of Lee, Shen, Huange, and Marron[1], and in the corresponding `s4vd` R code[4] are performance bottlenecks. These notes consider just equation (12)–similar observations apply to Eqn. (13). Equation (12) presents the minimization problem:

$$\min_{\lambda_v} \left(\frac{\|Y - \hat{Y}\|_F^2}{nd\hat{\sigma}^2} + \frac{\log(nd)}{nd} \hat{df}(\lambda_v) \right),$$

where,

$$\begin{aligned} \hat{Y} &= uv_{\lambda_v}^T, \\ v_{\lambda_v} &= \text{thresholded vector depending on} \lambda_v, \\ u^T u &= 1. \end{aligned}$$

The computationally expensive part of Eqn. (12) is evaluation of $\|Y - \hat{Y}\|_F^2$ for each new v_{λ_v}. We can exploit the fact that u is orthonormal to significantly reduce the computational cost (also noting that $\hat{Y} \perp \text{range}(I - uu^T)$ by construction):

$$\begin{aligned} \|Y - \hat{Y}\|_F^2 &= \|(I - uu^T)(Y - \hat{Y})\|_F^2 + \|uu^T(Y - \hat{Y})\|_F^2 \\ &= \|(I - uu^T)Y\|_F^2 + \|u(u^TY - u^T\hat{Y})\|_F^2 \\ &= \|(I - uu^T)Y\|_F^2 + \|u(u^TY - u^Tuv_{\lambda_v}^T)\|_F^2 \\ &= \|(I - uu^T)Y\|_F^2 + \|u(u^TY - v_{\lambda_v}^T)\|_F^2 \\ &= \|(I - uu^T)Y\|_F^2 + \|u^TY - v_{\lambda_v}^T\|_2^2. \end{aligned}$$

(The last identity is easy to show.)

Note that the first term of the sum,

$$\|(I - uu^T)Y\|_F^2,$$

does not depend on λ_v at all and is simply a constant term in the minimization problem (and thus, does not affect the optimization). Then

$$\min_{\lambda_v} \left(\frac{\|Y - \hat{Y}\|_F^2}{nd\hat{\sigma}^2} + \frac{\log(nd)}{nd} \hat{df}(\lambda_v) \right) = \min_{\lambda_v} \left(\frac{\|u^TY - v_{\lambda_v}^T\|_2^2}{nd\hat{\sigma}^2} + \frac{\log(nd)}{nd} \hat{df}(\lambda_v) \right).$$

Moreover, the term $u^T Y$ does not depend on λ_v and may be computed just once at the beginning of each BIC optimization loop.

In summary, we've replaced an expensive evaluation of a matrix Frobenius norm in Eqn. (12), $\|Y - \hat{Y}\|_F^2$, with a cheaper evaluation of a Euclidean norm of a vector, $\|u^T Y - v_{\lambda_v}^T\|_2^2$.

The optimizations outlined in this section apply to the `ssvd` function.

2.3 Parallel loop evaluation

The two BIC optimization for loops described in the reference paper[1] and implemented in the `s4vd` package contain completely independent loop iterations. Thus the iterations can be computed in parallel. R includes many mechanisms for parallel loop evaluation.

We modified the loops to use the `foreach`[3] package. The `foreach` package provides a simple way to dynamically choose from among many available parallel evaluation frameworks (called *parallel back ends*). The loops run sequentially if a parallel back end is not explicitly specified. Available parallel back ends include the `doSNOW, doMC, doMPI, doRedis` packages. See the `foreach` package documentation for more information.

The optimizations outlined in this section apply to the `ssvd` function.

2.4 Bug in ssvdBC function

Version 1.0 of the package contains an error in the biclustering function that calls the `ssvd` function. The `ssvdBC` function erroneously uses the kth singular vectors when calling `ssvd` in each "layer" loop. It should always use the 1st singular vectors of the deflated matrix (since the previous subspace has already been deflated away). We modified the `ssvdBC` function to use the 1st singular vectors of the deflated matrix in each layer.

This change only affects results that compute more than one solution layer.

3 Example

We walk through a simple, small example included in the `s4vd` package to illustrate the use of the new functions. The example first computes a result using the original functions present in version 1.0 of the `s4vd` package, and then computes a result using the modified functions from version 1.1 `s4vd` package.

This example should be run from a fresh R session, before loading the `s4vd` package.

Listing 1: Example.

```
install.packages("s4vd")
library("s4vd")

# example data set according to the simulation study in Lee et al. 2010
# generate artificial data set and a corresponding biclust object
u <- c(10,9,8,7,6,5,4,3,rep(2,17),rep(0,75))
v <- c(10,-10,8,-8,5,-5,rep(3,5),rep(-3,5),rep(0,34))
u <- u/sqrt(sum(u^2))
v <- v/sqrt(sum(v^2))
d <- 50
set.seed(1)
X <- (d*u%*%t(v)) + matrix(rnorm(100*50),100,50)
params <- info <- list()
RowxNumber <- matrix(rep(FALSE,100),ncol=1)
NumberxCol <- matrix(rep(FALSE,50),nrow=1)
RowxNumber[u!=0,1] <- TRUE
NumberxCol[1,v!=0] <- TRUE
Number <- 1
ressim <- BiclustResult(params,RowxNumber,NumberxCol,Number,info)

# perform ssvd biclustering
resssvd <- biclust(X,BCssvd,K=1)

# Now repeat computation using s4vd...

# XXX
# XXX


unloadNamespace("s4vd")
devtools::install_github("bwlewis/s4vd")
library(s4vd)
resssvd_new <- biclust(X,BCssvd,K=1)

# Compare the results:
cat("Original s4vd jaccardind output:\t")
print(jaccardind(ressim,resssvd))
[1] 0.8333333

cat("New s4vd jaccardind output:\t")
print(jaccardind(ressim,resssvd_new))
[1] 0.8333333
```

References

[1] Biclustering via Sparse Singular Value Decomposition, M. Lee, H. Shen, J. Huang, J. S. Marron, Biometrics 66, pp. 1087-1095, December 2010.

[2] Augmented Implicitly Restarted Lanczos Bidiagonalization Methods, J. Baglama and L. Reichel, SIAM J. Sci. Comput. 2005.

[3] http://cran.r-project.org/web/packages/foreach The foreach package

[4] http://cran.r-project.org/web/packages/s4vd The s4vd package