UCLA
Department of Statistics Papers

Title
The Multiway Package

Permalink
https://escholarship.org/uc/item/258260n9

Author
Jan de Leeuw

Publication Date
2011-10-25

eScholarship.org Powered by the California Digital Library
University of California

THE MULTIWAY PACKAGE

JAN DE LEEUW

ABSTRACT. This note documents R code for the CANDECOMP and TUCKER generalizations of the Singular Value Decomposition to multiway arrays. Alternating Least Squares algorithms are used to generate a convergent sequence of low-rank approximations.

1. CANDECOMP

```
candecomp<-function(a,x,ortho=rep(FALSE,length(x)),
    itmax=1000,eps=1e-6,verbose=FALSE)
```

The two leading arguments of the function `candecomp()` are an $n_1 \times n_2 \times \cdots \times n_m$ array `a` and a list of m matrices `x`, where `x[[i]]` has $n_i \times p$ elements. The function minimizes the least squares loss function over the `x[[i]]`, using the original values are starting values.

$$\sigma(x) = \sum_{i_1=1}^{n_1} \sum_{i_2=1}^{n_2} \cdots \sum_{i_m=1}^{n_m} (a_{i_1 i_2 \cdots i_m} - \sum_{s=1}^{p} x^1_{i_1 s} x^2_{i_2 s} \cdot \cdots x^m_{i_m s})^2.$$

There is an additional parameter `ortho`, a logical vector of length m. If `ortho[i]` is `TRUE` then `x[[i]]` is required to be columnwise orthonormal. The additional arguments `itmax`, `eps` and `verbose` set the iteration parameters.

2. TUCKER

```
tucker<-function(a,x,ident=rep(FALSE,length(x)),
    itmax=1000,eps=1e-6,verbose=FALSE)
```

Date: March 29, 2008 — 12h 24min — Typeset in TIMES ROMAN.

2000 *Mathematics Subject Classification.* 00A00.

Key words and phrases. Multidimensional Scaling, Array Approximation, Generalized SVD.

The leading arguments `a` and `x` are the same as before, except that now the `x[[i]]` are $n_i \times p_i$ columnwise orthonormal matrices. The loss function is

$$\sigma(x,b) = \sum_{i_1=1}^{n_1}\sum_{i_2=1}^{n_2}\cdots\sum_{i_m=1}^{n_m}(a_{i_1 i_2\cdots i_m} - \sum_{s_1=1}^{p_1}\sum_{s_2=1}^{p_2}\cdots\sum_{s_m=1}^{p_m} b_{s_1 s_2\cdots s_m} x^1_{i_1 s_1} x^2_{i_2 s_2}\cdots x^m_{i_m s_m})^2.$$

The $p_1 \times p_2 \times \cdots \times p_m$ array `b` is called the *core array*. There is an additional parameter `ident`, a logical vector of length m. If `ident[i]` is `TRUE` then matrix `x[[i]]` is restricted to be the $n_i \times n_i$ identity matrix.

Note that we can also write

$$\sigma(x,b) = \|a - (X_1 \otimes X_2 \otimes \cdots \otimes X_m)b\|^2,$$

where a and b are the $vec(\bullet)$ of the arrays, $\otimes$ is the Kronecker product, and $\|\bullet\|$ is the Frobenius norm.

3. Utilities

The package includes functions that compute generalized Hadamard products, generalized outer products and generalized Kronecker products on lists of matrices. A Procrustus approximation routine is also included.

4. Three-way

In the important three-way case `tucker()` with `ident[3]=TRUE` fits the approximation $A_k = XB_kY'$, while `candecomp()` fits the same approximation $A_k = XB_kY'$, with the B_k restricted to be diagonal.

5. Code

```
#
#   multiway package
#   Copyright (C) 2008  Jan de Leeuw <deleeuw@stat.ucla.edu>
#   UCLA Department of Statistics, Box 951554, Los Angeles, CA 90095-1554
#
#   This program is free software; you can redistribute it and/or modify
#   it under the terms of the GNU General Public License as published by
#   the Free Software Foundation; either version 2 of the License, or
#   (at your option) any later version.
#
#   This program is distributed in the hope that it will be useful,
#   but WITHOUT ANY WARRANTY; without even the implied warranty of
#   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#   GNU General Public License for more details.
```

```
#
#   You should have received a copy of the GNU General Public License
#   along with this program; if not, write to the Free Software
#   Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
#
#####################################################################
#
# version 0.1.0, 2008-03-27,            first release
# version 0.2.0, 2008-03-28,            added tucker and constraints
#

candecomp<-function(a,x,ortho=rep(FALSE,length(x)),itmax=1000,eps=1e-6,verbose=
     FALSE) {
        ndam<-dim(a)
        nard<-length(ndam)
        ndim<-ncol(x[[1]])
        for (k in 1:nard) if (ortho[k]) x[[k]]<-procrustus(x[[k]])
        c<-lapply(x,crossprod)
        oloss<-candeValue(a,x)$loss
        itel<-1
        repeat {
                for (k in 1:nard) {
                        cc<-arrHadamard(c[-k])
                        for (p in 1:ndim) {
                                y<-lapply(x[-k],function(z) z[,p])
                                b<-arrOuter(y)
                                x[[k]][,p]<-apply(a,k,function(z) sum(z*b))
                                }
                        if (ortho[k]) x[[k]]<-procrustus(x[[k]])
                                else x[[k]]<-t(solve(cc,t(x[[k]])))
                        c[[k]]<-crossprod(x[[k]])
                        }
                        cval<-candeValue(a,x)
                        nloss<-cval$loss; ahat<-cval$ahat
                if (verbose)
                        cat("Iteration: ",formatC(itel,digits=3,width=3),
                                "Old Loss: ",formatC(oloss,digits=6,width=10,
                                     format="f"),
                                "New Loss: ",formatC(nloss,digits=6,width=10,
                                     format="f"),
                                "\n")
                if ((itel == itmax) | ((oloss - nloss) < eps)) break()
                itel<-itel+1; oloss<-nloss
                }
        return(list(x=x,ahat=ahat,loss=nloss))
        }

candeValue<-function(a,x) {
        ndim<-ncol(x[[1]])
        ahat<-array(0,dim(a))
        for (p in 1:ndim) {
                y<-sapply(x,function(z) z[,p])
                b<-arrOuter(y)
```

```
                ahat<-ahat+b
                }
        loss<-sum((a-ahat)^2)
        return(list(ahat=ahat,loss=loss))
        }

tucker<-function(a,x,ident=rep(FALSE,length(x)),itmax=1000,eps=1e-6,verbose=FALSE)
      {
        ndam<-dim(a)
        for (k in 1:nard) if (ident[k]) x[[k]]<-diag(ndam[k])
        ndbm<-sapply(x,function(z) ncol(z))
        nard<-length(ndam)
        rard<-rev(1:nard)
        x<-lapply(x,procrustus)
        xx<-arrKronecker(x)
        aa<-as.vector(aperm(a,rard))
        bb<-colSums(aa*xx)
        b<-aperm(array(bb,rev(ndbm)),rard)
        ahat<-aperm(array(drop(xx%*%bb),rev(ndam)),rard)
        oloss<-sum((a-ahat)^2)
        itel<-1
        repeat {
                for (k in 1:nard) {
                        if (!ident[k]) {
                                z<-crossprod(flatten(a,k),arrKronecker(x[-k]))
                                     %*%flatten(b,k)
                                x[[k]]<-procrustus(z)
                                }
                        }
                xx<-arrKronecker(x)
                aa<-as.vector(aperm(a,rard))
                bb<-colSums(aa*xx)
                b<-aperm(array(bb,rev(ndbm)),rard)
                ahat<-aperm(array(drop(xx%*%bb),rev(ndam)),rev(1:nard))
                nloss<-sum((a-ahat)^2)
                if (verbose)
                        cat("Iteration: ",formatC(itel,digits=3,width=3),
                                "Old Loss: ",formatC(oloss,digits=6,width=10,
                                     format="f"),
                                "New Loss: ",formatC(nloss,digits=6,width=10,
                                     format="f"),
                                "\n")
                if ((itel == itmax) | ((oloss - nloss) < eps)) break()
                itel<-itel+1; oloss<-nloss
                }
        return(list(x=x,b=b,ahat=ahat,loss=nloss))
        }

arrHadamard<-function(c,fun=function(x,y) x*y) {
        nmat<-length(c)
        if (nmat == 0) stop("empty argument in arrHadamard")
        res<-c[[1]]
        if (nmat == 1) return(res)
```

```
        for (i in 2:nmat) res<-fun(res,c[[i]])
        return(res)
        }

arrOuter<-function(x,fun="*") {
        nmat<-length(x)
        if (nmat == 0) stop("empty argument in arrOuter")
        res<-x[[1]]
        if (length(x) == 1) return(res)
        for (i in 2:nmat) res<-outer(res,x[[i]],fun)
        return(res)
        }

arrKronecker<-function(x,fun="*") {
        nmat<-length(x)
        if (nmat == 0) stop("empty argument in arrKronecker")
        res<-x[[1]]
        if (length(x) == 1) return(res)
        for (i in 2:nmat) res<-kronecker(res,x[[i]],fun)
        return(res)
        }

flatten<-function(a,k) {
        nard<-rev(1:(length(dim(a))-1))
        apply(a,k,function(z) as.vector(aperm(z,nard)))
}

procrustus<-function(x) {
        res<-svd(x)
        return(tcrossprod(res$u,res$v))
}
```

DEPARTMENT OF STATISTICS, UNIVERSITY OF CALIFORNIA, LOS ANGELES, CA 90095-1554

E-mail address, Jan de Leeuw: deleeuw@stat.ucla.edu

URL, Jan de Leeuw: http://gifi.stat.ucla.edu