Package 'RDFTensor'

May 31, 2020

Type Package

Title Different Tensor Factorization (Decomposition) Techniques for RDF Tensors (Three-Mode-Tensors)

Version 1.2

Date 2020-5-31

Author Abdelmoneim Amer Desouki

Maintainer Abdelmoneim Amer Desouki <desouki@mail.upb.de>

Depends R (>= 3.2.0), Matrix, methods, pracma, doParallel, parallel, foreach

Suggests pryr, rARPACK, knitr, rmarkdown

VignetteBuilder knitr

Description Different Tensor Factorization techniques suitable for RDF Tensors.
RDF Tensors are three-mode-tensors, binary tensors and usually very sparse.
Currently implemented methods are
'RESCAL' Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel (2012) <doi:10.1145/2187836.2187874>,
'NMU' Daniel D. Lee and H. Sebastian Seung (1999) <doi:10.1038/44565>,
'ALS', Alternating Least Squares
'parCube' Papalexakis, Evangelos, C. Faloutsos, and N. Sidiropoulos (2012) <doi:10.1007/978-3-642-33460-3_39>,
'CP_APR' C. Chi and T. G. Kolda (2012) <doi.org/10.1137/110859063>.
The code is mostly converted from MATLAB and Python implementations of these methods.
The package also contains functions to get Boolean (Binary) transformation of the real-number-decompositions.
These methods also are for general tensors, so with few modifications they can be applied for other types of tensor.

License GPL-3

NeedsCompilation no

Repository CRAN

Date/Publication 2020-05-31 11:30:02 UTC

R topics documented:

`RDFTensor-package` *Different Tensor Factorization (Decomposition) Techniques for RDF Tensors (Three-Mode-Tensors)*

Description

Different Tensor Factorization techniques suitable for RDF Tensors. RDF Tensors are three-mode-tensors, binary tensors and usually very sparse. Currently implemented methods are 'RESCAL' Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel (2012) <doi:10.1145/2187836.2187874>, 'NMU' Daniel D. Lee and H. Sebastian Seung (1999) <doi:10.1038/44565>, 'ALS', Alternating Least Squares 'parCube' Papalexakis, Evangelos, C. Faloutsos, and N. Sidiropoulos (2012) <doi:10.1007/978-3-642-33460-3_39>, 'CP_APR' C. Chi and T. G. Kolda (2012) <doi.org/10.1137/110859063>. The code is mostly converted from MATLAB and Python implementations of these methods. The package also contains functions to get Boolean (Binary) transformation of the real-number-decompositions. These methods also are for general tensors, so with few modifications they can be applied for other types of tensor.

Details

The DESCRIPTION file:

Package:	RDFTensor
Type:	Package
Title:	Different Tensor Factorization (Decomposition) Techniques for RDF Tensors (Three-Mode-Tensors)
Version:	1.2
Date:	2020-5-31
Author:	Abdelmoneim Amer Desouki
Maintainer:	Abdelmoneim Amer Desouki <desouki@mail.upb.de>
Depends:	R (>= 3.2.0), Matrix, methods, pracma, doParallel, parallel, foreach
Suggests:	pryr, rARPACK, knitr, rmarkdown
VignetteBuilder:	knitr
Description:	Different Tensor Factorization techniques suitable for RDF Tensors. RDF Tensors are three-mode-tensors,
License:	GPL-3

Index of help topics:

	resident tensors
spt_mttkrp	Matricized tensor times Khatri-Rao product for ktensor
tensor2mat	Reduce tensor to matrix
tnsr2trp	sparse tensor to triples
umls_tnsr	RDF tensor of UMLS small graph

Further information is available in the following vignettes:

RDFTensor-Demo	Title of your vignette (source, pdf)

Read the tensor using getTensor or use NTriples file and parseNT. Use one of the methods to factorize it cp_nmu, cp_apr, parCube or rescal.

Author(s)

Abdelmoneim Amer Desouki

References

-Brett W. Bader, Tamara G. Kolda and others. MATLAB Tensor Toolbox, Version [v3.0]. Available online at https://www.tensortoolbox.org, 2015.

-Papalexakis, Evangelos E., Christos Faloutsos, and Nicholas D. Sidiropoulos. "Parcube: Sparse parallelizable tensor decompositions." Machine Learning and Knowledge Discovery in Databases. Springer Berlin Heidelberg, 2012. 521-536.

NMU -Lee, Daniel D., and H. Sebastian Seung. "Algorithms for non-negative matrix factorization." In Advances in neural information processing systems, pp. 556-562. 2001.

-Anh Huy Phan, Petr Tichavský, Andrzej Cichocki, On Fast Computation of Gradients for CANDECOMP/PARAFAC Algorithms, arXiv:1204.1586, 2012

CP_APR -E. C. Chi and T. G. Kolda. On Tensors, Sparsity, and Nonnegative Factorizations, SIAM J. Matrix Analysis, 33(4):1272-1299, Dec. 2012, http://dx.doi.org/10.1137/110859063

-S. Hansen, T. Plantenga and T. G. Kolda, Newton-Based Optimization for Kullback-Leibler Nonnegative Tensor Factorizations, Optimization Methods and Software, 2015, http://dx.doi.org/10.1080/10556788.2015.100997

RESCAL -Nickel, Maximilian, and Volker Tresp. "Tensor factorization for multi-relational learning." In Joint European Conference on Machine Learning and Knowledge Discovery in Databases, pp. 617-621. Springer, Berlin, Heidelberg, 2013.

See Also

cp_nmu cp_apr serial_parCube rescal cp_als

Examples

```
## Not run:
 data(umls_tnsr)
 ntnsr=umls_tnsr
    r=10

    sr=NULL
    t0=proc.time()
    X_=list()
    library(Matrix)
  tt=rescal(umls_tnsr$X,rnk=10,ainit='nvecs',verbose=1,lambdaA=0,epsilon=1e-2,lambdaR=0)
    R=tt$R
    A=tt$A
     for(s in 1:length(R)){
      print(sprintf('-------------=======s=%d==========---------',s))
      t1=proc.time()
      t1n=A%*%R[[s]]%*%Matrix::t(A)
      mx=max(t1n@x)
      Xb=Matrix::spMatrix(i=ntnsr$X[[s]]@i+1,j=ntnsr$X[[s]]@j+1,x=ntnsr$X[[s]]@x==1,
      nrow=nrow(ntnsr$X[[s]]),ncol=ncol(ntnsr$X[[s]]))

      all_scale_fact=c(0.7,0.8,0.9,0.95,1,1.05,1.1,1.2,1.3,1.4,1.5,1.8)
      for(scale_fact in all_scale_fact){
          qntl=scale_fact*sum(Xb)/(nrow(ntnsr$X[[s]])*ncol(ntnsr$X[[s]]))
            thr=quantile(t1n@x,1-qntl)
            print(sprintf('-------------%f---------',thr))
                 aa=which(t1n>thr,arr.ind=TRUE)
                 if(length(aa)>0){
                    X_[[s]]=Matrix::spMatrix(i=aa[,1],j=aa[,2],x=rep(1,nrow(aa)),
                    nrow=nrow(A),ncol=nrow(A))#tt > threshold[i],'sparseMatrix')
                }else{
                   X_[[s]]=Matrix::spMatrix(i=1,j=1,x=0,nrow=nrow(A),ncol=nrow(A))
                }
        #---
            li=Xb@i[Xb@x]+1
            lj=Xb@j[Xb@x]+1
            tp=sum(X_[[s]][cbind(li,lj)])
            fn=sum(Xb@x)-tp#sum(!X_[cbind(li,lj)])
            # incase of scale_fact=1 fp=fn as number of 1's in X_ and X is the same
            fp=sum(X_[[s]]@x)-tp
            sr=rbind(sr,cbind(s=s,r=r,scale_fact=scale_fact,mx=mx,thr=thr,nnz=sum(Xb),
            tp=tp,fn=fn,fp=fp,R=tp/(tp+fn),P=tp/(tp+fp)))
        #    if(tp==0) break;
        }
        t2=proc.time()
        print(t2-t1)
    }
 tf=proc.time()
 print(tf-t0)

 Res=NULL
```

```
    for(sf in all_scale_fact){
        sr.sf=sr[sr[,'scale_fact']==sf ,]
         R=sum(sr.sf[,'tp'])/(sum(sr.sf[,'tp'])+sum(sr.sf[,'fn']))
         P=sum(sr.sf[,'tp'])/(sum(sr.sf[,'tp'])+sum(sr.sf[,'fp']))
        cnt=nrow(sr.sf)
        Res=rbind(Res,cbind(sf=sf,P,R,cnt))
    }
   print(Res)

   stats=Res

     plot(stats[,'sf'],stats[,'R']*100,type='b',col='red',lwd=2,
    main=sprintf('RESCAL, choosing scale factor (sf):(ntrp*sf), dataset: %s,
        #Slices:%d\n #Known facts:%d','UMLS',length(ntnsr$X),
        sum(sr.sf[,'tp']+sr.sf[,'fn'])),ylab="",xlab='Scale Factor',
    xlim=c(0,max(sf)),ylim=c(0,100))
    HM=apply(stats,1,function(x){2/(1/x['P']+1/x['R'])})
    points(stats[,'sf'],stats[,'P']*100,col='blue',lwd=2,type='b')
    points(stats[,'sf'],100*HM,col='green',lwd=2,type='b')
    grid(nx=10, lty = "dotted", lwd = 2)
    legend(legend=c('Recall','Precision','Harmonic mean'),col=c('red','blue','green'),
    x=0.6,y=20,pch=1,cex=0.75,lwd=2)

    max(HM)


    hist(sr[sr[,'scale_fact']==1,'thr'],col='grey',
    main='UMLS Reconstring the same number of triples, Actual threshold',
    xlab='threshold',cex.main=0.75)

## End(Not run)


trp=rbind(
    cbind('Alex',  'loves', 'Don'),
    cbind('Alex',  'loves', 'Elly'),
    cbind('Alex',  'hates', 'Bob'),
    cbind('Don',   'loves', 'Alex'),
    cbind('Don',   'hates', 'Chris'),
    cbind('Chris', 'hates', 'Bob'),
    cbind('Bob',   'hates', 'Chris'),
    cbind('Elly',  'hates', 'Chris'),
    cbind('Elly',  'hates', 'Bob'),
    cbind('Elly',  'loves', 'Alex')
    )
######
# form tensor as a set of frontal slices (Predicate mode)
    tnsr=getTensor(trp)
    subs=getTnsrijk(tnsr$X)
    X=list(subs=subs,vals=rep(1,nrow(subs)),size=c(5,2,5))
    normX=sqrt(sum(X$vals))
    set.seed(123)
    # NMU decomposition with rank 2
```

```
P1=cp_nmu(X,2)

###
# find best CP boolean Factorization based on NMU
res=CP_01(X,P1[[1]])
Fac=res$sol$u # The factorization
# TP,FP,FN
print(sprintf("TP=%d, FP=%d, FN=%d, Threshold=%f",res$sol$TP,res$sol$FP,res$sol$FN,res$sol$thr))

#################   CP_APR   ##################
res=cp_apr(X,R=2,opts=list(alg='pdnr',printinneritn=1))

set.seed(12345)
res1=CP_01(X,res[[1]])
res2=CP_R01(X,res[[1]])
#res3=CP_01ext(X,res[[1]])
##################   RESCAL  ############################

tt=rescal(tnsr$X,2,ainit='random',verbose=3)
R=tt[['R']]
A=tt[['A']]

Tensor_error(tnsr$X,A,R)
t1n= A
t2n= A
```

CP_01 *transformation of the real-number-CP-decompositions to Binary*

Description

transforms CP tensor decomposition generated by real-number methods (like nmu) to Binary by trying different threshold values.

Usage

```
CP_01(X, P, pthr = c(0.001, 0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.8),
          testAll = FALSE)
```

Arguments

X	the original tensor as sptensor list(subs,vals,size,nnz)
P	a LIST containing the CP transformation
pthr	list of threshold values to be tried
testAll	a flag to try all threshold values or stop after having a zero in fp or tp

Value

If it is a LIST, use

Res	the tp, fn, fp result of each threshold value.
bestSol	best solution in terms of minimum error (fn+fp)

Author(s)

Abdelmoneim Amer Desouki

See Also

cp_apr serial_parCube rescal rescal_01 cp_nmu

Examples

```
trp=rbind(
    cbind('Alex',  'loves', 'Don'),
    cbind('Alex',  'loves', 'Elly'),
    cbind('Alex',  'hates', 'Bob'),
    cbind('Don',   'loves', 'Alex'),
    cbind('Don',   'hates', 'Chris'),
    cbind('Chris', 'hates', 'Bob'),
    cbind('Bob',   'hates', 'Chris'),
    cbind('Elly',  'hates', 'Chris'),
    cbind('Elly',  'hates', 'Bob'),
    cbind('Elly',  'loves', 'Alex')
    )
######
# form tensor as a set of frontal slices (Predicate mode)
    tnsr=getTensor(trp)
    subs=getTnsrijk(tnsr$X)
    X=list(subs=subs,vals=rep(1,nrow(subs)),size=c(5,2,5))
    normX=sqrt(sum(X$vals))
    set.seed(123)
    # NMU decomposition with rank 2
    P1=cp_nmu(X,2)
    res=CP_01(X,P1[[1]])
    Fac=res$sol$u # The factorization
    # TP,FP,FN
  print(sprintf("TP=%d, FP=%d, FN=%d, Threshold=%f",res$sol$TP,res$sol$FP,res$sol$FN,res$sol$thr))
```

---

cp_als *Compute a CP decomposition using an alternating least-squares algorithm(als)*

Description

computes an estimate of the best rank-R PARAFAC model of a tensor X using an alternating least-squares algorithm Translated from cp_als.m : MATLAB Tensor Toolbox

Usage

```
cp_als(X, R, opts = list())
```

Arguments

X	is a sparse tensor (a LIST containing subs, vals and size)
R	The rank of the factorization
opts	a list containing the options for the algorithm like maxiters:maximum iterations, tol:tolerance .. etc.

Value

P	the factorization of X as a LIST representing Kruskal Tensor (lambda and u)
Uinit	the initial solution
iters	number of iterations.
fit	fraction explained by the model.

Author(s)

Abdelmoneim Amer Desouki

References

-Brett W. Bader, Tamara G. Kolda and others. MATLAB Tensor Toolbox, Version [v3.0]. Available online at https://www.tensortoolbox.org, 2015.

See Also

cp_apr serial_parCube rescal cp_nmu

Examples

```
subs=matrix(c(5,1,1,
              3,1,2,
              1,1,3,
              2,1,3,
              4,1,3,
              6,1,3,
              1,1,4,
              2,1,4,
              4,1,4,
              6,1,4,
              1,2,1,
              3,2,1,
```

```
              5,2,1),byrow=TRUE,ncol=3)

X=list(subs=subs,vals=rep(1,nrow(subs)),size=c(6,2,4))
set.seed(123)
P1=cp_als(X,2)
```

`cp_apr` *Compute nonnegative CP with alternating Poisson regression(CP_APR)*

Description

computes an estimate of the best rank-R CP model of a nonnegative tensor X using an alternating Poisson regression. This is most appropriate for sparse count data (i.e., nonnegative integer values) because it uses Kullback-Liebler divergence.

Usage

```
cp_apr(X, R, opts = list())
```

Arguments

`X`	is a sparse tensor (a LIST containing subs, vals and size)
`R`	The rank of the factorization
`opts`	a list containing the options for the algorithm like maxiters:maximum iterations, tol:tolerance .. etc.

Details

Different algorithm variants are available (selected by the 'alg' parameter): 'pqnr' - row subproblems by projected quasi-Newton (default) 'pdnr' - row subproblems by projected damped Hessian 'mu' - multiplicative update Additional input parameters for algorithm 'mu': 'kappa' - Offset to fix complementary slackness 100 'kappatol' - Tolerance on complementary slackness 1.0e-10

Additional input parameters for algorithm 'pdnr': 'epsActive' - Bertsekas tolerance for active set 1.0e-8 'mu0' - Initial damping parameter 1.0e-5 'precompinds' - Precompute sparse tensor indices TRUE 'inexact' - Compute inexact Newton steps TRUE

Additional input parameters for algorithm 'pqnr': 'epsActive' - Bertsekas tolerance for active set 1.0e-8 'lbfgsMem' - Number vector pairs to store for L-BFGS 3 'precompinds' - Precompute sparse tensor indices TRUE

Value

`M`	the factorization of X as a LIST representing Kruskal Tensor (lambda and u)
`Minit`	the initial solution
`output`	statistics about the solution like the running time of each step and the error.

Author(s)

Abdelmoneim Amer Desouki

References

-Brett W. Bader, Tamara G. Kolda and others. MATLAB Tensor Toolbox, Version [v3.0]. Available online at https://www.tensortoolbox.org, 2015.

-E. C. Chi and T. G. Kolda. On Tensors, Sparsity, and Nonnegative Factorizations, SIAM J. Matrix Analysis, 33(4):1272-1299, Dec. 2012, http://dx.doi.org/10.1137/110859063 -S. Hansen, T. Plantenga and T. G. Kolda, Newton-Based Optimization for Kullback-Leibler Nonnegative Tensor Factorizations, Optimization Methods and Software, 2015, http://dx.doi.org/10.1080/10556788.2015.1009977

See Also

cp_nmu serial_parCube rescal cp_als

Examples

```
subs=matrix(c(5,1,1,
              3,1,2,
              1,1,3,
              2,1,3,
              4,1,3,
              6,1,3,
              1,1,4,
              2,1,4,
              4,1,4,
              6,1,4,
              1,2,1,
              3,2,1,
              5,2,1),byrow=TRUE,ncol=3)

   X=list(subs=subs,vals=rep(1,nrow(subs)),size=c(6,2,4))
   set.seed(12345)#for reproducability
   P1=cp_apr(X,2,opts=list(alg='mu'))
   print(P1$M)
   set.seed(12345)#for reproducability
   P2=cp_apr(X,2,opts=list(alg='pdnr'))
   print(P2$M)
```

---

cp_nmu *Compute nonnegative CP with multiplicative updates(NMU)*

Description

computes an estimate of the best rank-R PARAFAC model of a tensor X with nonnegative constraints on the factors. This version uses the Lee & Seung multiplicative updates from their NMF algorithm. Translated from cp_nmu.m : MATLAB Tensor Toolbox

Usage

```
cp_nmu(X, R, opts = list())
```

Arguments

X	is a sparse tensor (a LIST containing subs, vals and size)
R	The rank of the factorization
opts	a list containing the options for the algorithm like maxiters:maximum iterations, tol:tolerance .. etc.

Value

P	the factorization of X as a LIST representing Kruskal Tensor (lambda and u)
Uinit	the initial solution
stats	statistics about the solution like the running time of each step and the error.
fit	fraction explained by the model.

Author(s)

Abdelmoneim Amer Desouki

References

-Brett W. Bader, Tamara G. Kolda and others. MATLAB Tensor Toolbox, Version [v3.0]. Available online at https://www.tensortoolbox.org, 2015.

-Lee, Daniel D., and H. Sebastian Seung. "Algorithms for non-negative matrix factorization." In Advances in neural information processing systems, pp. 556-562. 2001.

See Also

cp_apr serial_parCube rescal cp_als

Examples

```
subs=matrix(c(5,1,1,
              3,1,2,
              1,1,3,
              2,1,3,
              4,1,3,
              6,1,3,
              1,1,4,
              2,1,4,
              4,1,4,
              6,1,4,
              1,2,1,
              3,2,1,
              5,2,1),byrow=TRUE,ncol=3)

X=list(subs=subs,vals=rep(1,nrow(subs)),size=c(6,2,4))
```

```
set.seed(123)
P1=cp_nmu(X,2)
```

CP_R01 *inverse of real-number-CP-decompositions to Binary*

Description

gets the best reconstruction of a CP-factorization by trying different thresholds on the result. If the size of the tensor is too big sampling is done to get estimates of TP, FP.

Usage

```
CP_R01(X, P,
   pthr = c(1e-06,1e-04,0.001,0.01,0.05,0.1,0.2,0.3,0.4,0.5,0.55,0.6, 0.65, 0.7, 0.8),
    cntNnz = 200, startSize = 1e+07)
```

Arguments

X	the original tensor as sptensor list(subs,vals,size,nnz)
P	a LIST containing the CP transformation
pthr	list of threshold values to be tried
cntNnz	If |X| is the number of non-zeros in data X, the sampled locations will be cntNnz*|X| of 0s.
startSize	if size of tensor < startSize all values are calculated, default 1e7.

Value

A LIST containing the TP, FP, FN and threshold value also the result of best threshold

Author(s)

Abdelmoneim Amer Desouki

See Also

cp_apr serial_parCube rescal rescal_01 cp_nmu

Examples

```
trp=rbind(
    cbind('Alex',  'loves', 'Don'),
    cbind('Alex',  'loves', 'Elly'),
    cbind('Alex',  'hates', 'Bob'),
    cbind('Don',   'loves', 'Alex'),
    cbind('Don',   'hates', 'Chris'),
    cbind('Chris', 'hates', 'Bob'),
```

```
    cbind('Bob',   'hates', 'Chris'),
    cbind('Elly',  'hates', 'Chris'),
    cbind('Elly',  'hates', 'Bob'),
    cbind('Elly',  'loves', 'Alex')
    )
######
# form tensor as a set of frontal slices (Predicate mode)
    tnsr=getTensor(trp)
    subs=getTnsrijk(tnsr$X)
    X=list(subs=subs,vals=rep(1,nrow(subs)),size=c(5,2,5))
    normX=sqrt(sum(X$vals))
    set.seed(123)
    # NMU decomposition with rank 2
    P1=cp_nmu(X,2)
    res=CP_R01(X,P1[[1]])
```

---

default_parcube_options *parCube initialization parameters*

Description

a LIST of the default options for serial_parCube method

Usage

```
default_parcube_options()
```

Value

a LIST of the default options for serial_parCube method

Author(s)

Abdelmoneim Amer Desouki

---

getTensor *getting tensor from triples*

Description

gets a tensor data structure as a set of frontal slices represented as sparse matrices. The entities have the same id as subject and/or object.

Usage

```
getTensor(trp, SO = NULL, P = NULL)
```

Arguments

trp	three columns representing the set of triples (subject, predicate, object). Each row representes one triple.
SO	the predifined list of entities
P	The set of predicates

Value

A LIST containing:

X	set of sparse matrices with ones in the position of triples, each sparse matix represents one predicate (relationship)
SO	the set of entities in the subject and object domains
P	the set of predicates

Author(s)

Abdelmoneim Amer Desouki

---

getTensor3m *getting tensor from triples*

Description

gets a tensor data structure as a set of frontal slices represented as sparse matrices. The entities as subject has a different id than object and the size of subject and object modes can be different.

Usage

```
getTensor3m(trp, S = NULL, P = NULL, O = NULL)
```

Arguments

trp	three columns representing the set of triples (subject, predicate, object). Each row representes one triple.
S	the predifined list of subjects
P	The set of predicates
O	the predifined list of objects

Value

A LIST containing:

X	set of sparse matrices with ones in the position of triples, each sparse matix represents one predicate (relationship)
S	the set of entities in the subject domain
O	the set of entities in the object domain
P	the set of predicates

Author(s)

Abdelmoneim Amer Desouki

`getTnsrijk` *tensor frontal slices to indices*

Description

convert tensor frontal slices to indices with three columns (i.e i ,j , k)

Usage

```
getTnsrijk(X)
```

Arguments

`X`	LIST of sparce matrices, with each entry representing one predicate.

Value

a matrix of three columns representing indices of 1 values in the tensor

Author(s)

Abdelmoneim Amer Desouki

See Also

`tnsr2trp` `getTensor`

Examples

```
trp=rbind(
    cbind('Alex',  'loves', 'Don'),
    cbind('Alex',  'loves', 'Elly'),
    cbind('Alex',  'hates', 'Bob'),
    cbind('Don',   'loves', 'Alex'),
    cbind('Don',   'hates', 'Chris'),
    cbind('Chris', 'hates', 'Bob'),
    cbind('Bob',   'hates', 'Chris'),
    cbind('Elly',  'hates', 'Chris'),
    cbind('Elly',  'hates', 'Bob'),
    cbind('Elly',  'loves', 'Alex')
    )
######
# form tensor as a set of frontal slices (Predicate mode)
    tnsr=getTensor(trp)
    subs=getTnsrijk(tnsr$X)
    print(subs)
```

`inv_rescal_sf_prd_chnkgrp` *Reconstruct predicate(s) via chunks and grouping*

Description

reconstruct a predicate or set of predicates from RESCAL factorization result by returning top scores in each predicate.

Usage

```
inv_rescal_sf_prd_chnkgrp(R, A, cnt_prd, scale_fact = 1, verbose = 1, ncores = 3,
ChnkLen = 1000, saveRes = FALSE, OS_WIN = FALSE, pve = 1e-10, grpLen = NULL,
dsname = "", rmx = NULL, readjustChnkLen = TRUE, TotalChnkSize = 5e+08,
chTrpCntTol = 1000, generateLog = FALSE)
```

Arguments

`R`	core tensor resulting from RESCAL factorization (r by r by m).
`A`	Embedding matrix part resulting from RESCAL factorization.
`cnt_prd`	vector with length of number of predicate giving number of triples in each predicate. When 0 the predicate will not be processed.
`scale_fact`	scale factor, the generated number of triples in a predicate s is sf*prd_cnt[s]
`verbose`	the level of messages to be displayed, 0 is minimal.
`ncores`	number of cores used to run in parallel, 0 means no paralellism
`OS_WIN`	True when the operating system is windows, used to allow using Fork when running in parallel
`pve`	positive value: representing the smallest value of allowed score of reconstructed triple.
`grpLen`	length of one group of iterations, when running iterations in parallel results are collected for all iterations to be summarized after last iteration. Thus more memory is required. To avoid that iterations are divided to groups with summaries calculated for each group. Default 15.
`ChnkLen`	number of rows in one chunk. Default 1000.
`generateLog`	save output when running in parallel to a log file in current directory.
`saveRes`	opionally save each predicate
`dsname`	optional:name of dataset
`rmx`	optionally give the max absolute value in A to save its calculation in multiple calls
`readjustChnkLen`	automatically increase chunk length when possible

TotalChnkSize	instead of defining ChnkLen define the number of pairs of entities to processed every chunk. Equal to ChnkLen*N, where N is the number of entities (i.e. nrows(A))
chTrpCntTol	tolerance in number of triples returned in one chunk, will be eliminated at end of group

Details

Multiplication of A*R[[p]]*A^T can be impossible in typical 16GB RAM machine when the number of entities is more than 50K. (needs ~25GB RAM) To deal with that we use chunks to obtain the top scores from such multiplication. The main idea is to get the required number of triples from each chunk then choose from them again the top scores.

Value

LIST of three components:

ijk	A data frame containing the reconstructed triples using indexes of entities
val	A vector containing the scores of the reconstructed triples
act_thr	the minimum score in each predicate (minimum score of a triple, threshold)

Author(s)

Abdelmoneim Amer Desouki

References

-Maximilian Nickel, Volker Tresp, Hans-Peter-Kriegel, "Factorizing YAGO: Scalable Machine Learning for Linked Data" WWW 2012, Lyon, France

-SynthG: mimicking RDF Graphs Using Tensor Factorization, Desouki et al

See Also

rescal scRescal

Examples

```
## Not run:
  library(RDFTensor)
  data('umls_tnsr')
   ntnsr=umls_tnsr

   #Calculate Factorization
    tt0=proc.time()
    tt=rescal(ntnsr$X,rnk=10,ainit='nvecs',verbose=2,lambdaA=0,epsilon=1e-4,lambdaR=0)
    ttq1=proc.time()
    A=tt$A
    R=tt$R

    # reconstruct second predicate (slice) in tensor
```

```
 p=2
 prd_cnt=rep(0,length(ntnsr$X))#Zero counts will not be reconstructed
 prd_cnt[p]=sum(ntnsr$X[[p]])
tmpRes=inv_rescal_sf_prd_chnkgrp(R,A,cnt_prd=prd_cnt,ChnkLen=50,grpLen=6,OS_WIN=TRUE,ncores=1,
          chTrpCntTol=1000,  TotalChnkSize=1e4)

  ijk=tmpRes[[1]]
  ix=which(ntnsr$X[[p]]==1,arr.ind=TRUE)
  oijk=cbind(ix[,1],p,ix[,2])#Original
  flag= paste(oijk[,1],oijk[,2],oijk[,3]) %in% paste(ijk[,1],ijk[,2],ijk[,3])
  print(table( flag))#True positives

  pTrp=cbind(ntnsr$SO[ijk[,1]],ntnsr$P[ijk[,2]],ntnsr$SO[ijk[,3]])


## End(Not run)
```

---

recon_1prd_topn_par *Reconstruct predicate top scores on very large graphs*

Description

Reconstruct predicate top scores on very large graphs from RESCAL Factorization A & R. NB: A and R should be already loaded in the environment calling the function. Calculates top scores of A%*%R[[p]]%*%A^T. Uses chunks for rows and columns and constraints on maximum possible value of scores to avoid calculations of too small values

Usage

```
recon_1prd_topn_par(A,R,p, pcnt, rchLen = 1000, cchLen = 200, pve = 1e-10,
mxrIter = 5, mxcIter = 25, grpLen = 40, OS_WIN = FALSE, dsname = "", ncores = 8)
```

Arguments

A	Embedding matrix part resulting from RESCAL factorization.
R	core tensor resulting from RESCAL factorization (r by r by m).
p	predicate number.
pcnt	Number of triples in the predicate (to be reconstructed)
rchLen	row chunk length.
cchLen	column chunk length
pve	positive value: representing the smallest value of allowed score of reconstructed triple.
mxrIter	maximum number of iterations (chunks) in rows
mxcIter	maximum number of iterations (chunks) in columns

grpLen	length of one group of iterations, when running iterations in parallel results are collected for all iterations to be summarized after last iteration. Thus more memory is required. To avoid that iterations are divided to groups with summaries calculated for each group. Default 40.
ncores	number of cores used to run in parallel, 0 means no paralellism
OS_WIN	True when the operating system is windows, used to allow using Fork when running in parallel
dsname	optional:name of dataset

Value

The result is a LIST of three items:

ikv	A data frame containing the reconstructed triples (subject, Object, value, rchnk,chnk) using indexes of entities, rchnk and colun chunk are the chunk in which the triple is generated note predicate is an argument
minThr	the minimum score in each predicate (minimum score of a triple, threshold)
Iter	Number of iterations done

Author(s)

Abdelmoneim Amer Desouki

References

-Maximilian Nickel, Volker Tresp, Hans-Peter-Kriegel, "Factorizing YAGO: Scalable Machine Learning for Linked Data" WWW 2012, Lyon, France

-SynthG: mimicking RDF Graphs Using Tensor Factorization, Desouki et al

See Also

rescal scRescal RescalReconstructBack inv_rescal_sf_prd_chnkgrp

Examples

```
## Not run:
#Reconstructing one predicate from DBpedia factorization
#print(load('dbpi42_r100.RData'))
lst=recon_1prd_topn_par(A=A,R=R,p,pcnt=prd_cnt1[p],rchLen=250000,cchLen=750,
                    mxrIter=30,mxcIter=70,ncores=12,grpLen=40,pve=pve,dsname=name)
    Res=cbind(S=lst$ikv[,1],P=p,lst$ikv[,2:5])#triples , value


## End(Not run)
```

rescal *RESCAL: Tensor Factorization.*

Description

RESCAL-ALS algorithm to compute the RESCAL tensor factorization. The solution is a matrix and a core tensor. RESCAL factors a (usually sparse) three-way tensor X such that each frontal slice X_k is factored into X_k = A * R_k * A^T

Usage

```
rescal(X, rnk,ainit = 'nvecs',verbose=2,Ainit=NULL,Rinit=NULL,lambdaA=0,lambdaR=0,
    lambdaV=0,epsilon=1e-3,maxIter=100, minIter=1, P = list(),orthogonalize=FALSE,
      func_compute_fval='compute_fit')
```

Arguments

X	is a sparse tensor as set of sparse matrices, one for every relation (predicate). (a LIST of SparseMatrix)
rnk	The rank of the factorization
ainit	the method used to initialize matrix A
verbose	the level of messages to be displayed, 0 is minimal.
Ainit	the initial value of matrix A.
Rinit	the initial value of R (the core tensor, as LIST of frontal slices)
lambdaA	Regularization parameter for A factor matrix. 0 by default
lambdaR	Regularization parameter for R_k factor matrices. 0 by default
lambdaV	Regularization parameter for R_k factor matrices. 0 by default
epsilon	error threshold
maxIter	Maximum number of iterations
minIter	Minimum number of iterations
P	Not implemented
orthogonalize	Not implemented
func_compute_fval	function used to compute fit.

Value

list(A=A, R=R, all_err, nitr=itr + 1, times=as.vector(exectimes) Returns a LIST of the following:

A	The matrix A of the factorization (n by r)
R	The core tensor R the factorization as r (rank) matrices of (r by r)
nitr	number of iterations
times	list of running times of each step.

Author(s)

Abdelmoneim Amer Desouki

References

-Maximilian Nickel, Volker Tresp, Hans-Peter-Kriegel, "A Three-Way Model for Collective Learning on Multi-Relational Data", ICML 2011, Bellevue, WA, USA

-Maximilian Nickel, Volker Tresp, Hans-Peter-Kriegel, "Factorizing YAGO: Scalable Machine Learning for Linked Data" WWW 2012, Lyon, France

See Also

cp_apr serial_parCube cp_nmu cp_als

Examples

```
   X1=matrix(c(1,0,0,0,0, 0,1,0,0,0, 0,0,1,1,0, 0,0,0,0,1, 1,0,0,0,0),byrow=TRUE,nrow=5,ncol=5)
   X2=matrix(c(0,1,0,1,1, 1,0,0,1,0, 0,1,0,1,1, 0,0,0,0,1, 0,0,1,0,0),byrow=TRUE,nrow=5,ncol=5)
   X2_=matrix(c(0,1,0,1,1, 1,0,0,1,0, 0,0,0,0,0, 0,0,0,0,1, 0,0,1,0,0),byrow=TRUE,nrow=5,ncol=5)
    X=list(t(X1),t(X2),t(X2_))

    N=nrow(X1)
    Xs=list()
    for(s in 1:length(X)){
      aa=which(X[[s]]==1,arr.ind=TRUE)
      Xs[[s]]=Matrix::sparseMatrix(x=rep(1,nrow(aa)),i=aa[,1],j=aa[,2],dims=c(N,N))
    }

    print(Xs)

    rf=rescal(Xs,2)
    A=rf$A
    R=rf$R
    Tensor_error(Xs,A,R)
    tmp=rescal_01(Xs,A,R,scale_fact=1.5)#generate 1.5*original number of triples
   print(sprintf('Precision:%.4f, Recall:%.4f',tmp$tp/(tmp$tp+tmp$fp),tmp$tp/(tmp$tp+tmp$fn)))

#using RESCAL for prediction missing relations.

aa=read.table(file = paste0(path.package("RDFTensor"), '/toy_vicePresident.nt'),
                                sep=' ',header=FALSE,stringsAsFactors=FALSE)
trp=aa[,1:3]

tnsr=getTensor(trp)
    r=4

    sr=NULL
    t0=proc.time()
    X_=list()
    library(Matrix)
    tt=rescal(tnsr$X,rnk=r,ainit='nvecs',verbose=1,lambdaA=0,epsilon=1e-4,lambdaR=0)
```

```
    R=tt$R
    A=tt$A
s1=A%*%R[[1]]%*%Matrix::t(A)
s2=A%*%R[[2]]%*%Matrix::t(A)
#predict the party of AlGore (no explicit info is given in the nt file)
print(s1[tnsr$SO=='<http://example.com/AlGore>',tnsr$SO=='<http://example.com/RepublicanParty>'])
print(s1[tnsr$SO=='<http://example.com/AlGore>',tnsr$SO=='<http://example.com/DemocraticParty>'])
partyOf=data.frame(tnsr$SO,Repub=s1[,tnsr$SO=='<http://example.com/RepublicanParty>'],
           Democ=s1[,tnsr$SO=='<http://example.com/DemocraticParty>'],
    GivenRepub=tnsr$X[[1]][,tnsr$SO=='<http://example.com/RepublicanParty>'],
          GivenDemoc=tnsr$X[[1]][,tnsr$SO=='<http://example.com/DemocraticParty>'],
    stringsAsFactors=FALSE)

 print(partyOf)
```

RescalReconstructBack *reconstruct a tensor from RESCAL factorization result*

Description

reconstruct a tensor from RESCAL factorization result

Usage

```
RescalReconstructBack(A, R, trpo = NULL, otnsr = NULL, chkZR = FALSE, prd_cnt =
NULL, sf = 1, verbose = 1, ncores = 3, OS_WIN = FALSE, pve = 1e-10,
grpLen = NULL, ChnkLen = 1000, generateLog = FALSE)
```

Arguments

A	Embedding matrix part resulting from RESCAL factorization.
R	core tensor resulting from RESCAL factorization (r by r by m).
trpo	optional original tensor in for of triples(data frame of three columns subject, predicate, object)
otnsr	sparse tensor containing the triples to be reconstructed (i.e. to calculate scores for).
chkZR	flag to check if there is any slices in core tensor which are entirely zeros
prd_cnt	optional. Giving number of triples in each predicate.
sf	scale factor, the generated number of triples in a predicate s is sf*prd_cnt[s]
verbose	the level of messages to be displayed, 0 is minimal.
ncores	number of cores used to run in parallel, 0 means no paralellism
OS_WIN	True when the operating system is windows, used to allow using Fork when running in parallel
pve	positive value: representing the smallest value of allowed score of reconstructed triple.

grpLen	length of one group of iterations, when running iterations in parallel results are collected for all iterations to be summarized after last iteration. Thus more memory is required. To avoid that iterations are divided to groups with summaries calculated for each group. Default 15.
ChnkLen	number of rows in one chunk. Default 1000.
generateLog	save output when running in parallel to a log file in current directory.

Details

The function finds triples with top scores from RESCAL factorization. When the scale factor is 1 each predicate the resulting number of triples will be equal to that of the original tensor. Each predicate will get the same number of triples as that of the original graph. Since direct multiplication of A by A^T may not fit into RAM even for mid-size graphs, chunks are used to get the results.

Value

The result is a LIST of three items:

ijk	A data frame containing the reconstructed triples using indexes of entities
val	A vector containing the scores of the reconstructed triples
act_thr	the minimum score in each predicate (minimum score of a triple, threshold)

Author(s)

Abdelmoneim Amer Desouki

References

-Maximilian Nickel, Volker Tresp, Hans-Peter-Kriegel, "Factorizing YAGO: Scalable Machine Learning for Linked Data" WWW 2012, Lyon, France

-SynthG: mimicking RDF Graphs Using Tensor Factorization, Desouki et al

See Also

rescal scRescal

Examples

```
## Not run:
  library(RDFTensor)
  data('umls_tnsr')
   ntnsr=umls_tnsr

   #Calculate Factorization
    tt0=proc.time()
    tt=rescal(ntnsr$X,rnk=10,ainit='nvecs',verbose=2,lambdaA=0,epsilon=1e-4,lambdaR=0)
    ttq1=proc.time()
    A=tt$A
    R=tt$R
```

```
    # reconstruct tensor and calulate
    RecRes=RescalReconstructBack(R=R,A=A,otnsr=ntnsr,ncore=3,OS_WIN=TRUE)
    print(sprintf('True positive rate:%.2f %%',100*sum(RecRes$TP)/length(RecRes$TP)))


## End(Not run)
```

`rescal_01` *transformation of the real-number-RESCAL-decompositions to Binary*

Description

applies different thresholds to get a number of triples from A and R (result of RESCAL) decomposition as scale_fact*the_number_of_triples_in_original_tensor.

Usage

```
rescal_01(X, A, R, scale_fact = 1)
```

Arguments

`X`	is a sparse tensor as set of sparse matrices, one for every relation (predicate). (a LIST of SparseMatrix)
`A`	the A matrix returned by RESCAL factorization
`R`	the R LIST returned by RESCAL factorization
`scale_fact`	scale of the number of triples to be considered in the result. When it is 1 then a threshold will be taken to get the same number of triples in each slice as the original tensor.

Value

a LIST

`X_`	The reconstructed tensor as a set of frontal slices.
`tp`	the number of true positives
`fp`	the number of false positives
`fn`	the number of false negatives
`sr`	the details of each slice i.e the number of tp, fn, fp, etc

Author(s)

Abdelmoneim Amer Desouki

References

-Maximilian Nickel, Volker Tresp, Hans-Peter-Kriegel, "A Three-Way Model for Collective Learning on Multi-Relational Data", ICML 2011, Bellevue, WA, USA

See Also

rescal CP_01

Examples

```
X1=matrix(c(1,0,0,0,0, 0,1,0,0,0, 0,0,1,1,0, 0,0,0,0,1, 1,0,0,0,0),byrow=TRUE,nrow=5,ncol=5)
X2=matrix(c(0,1,0,1,1, 1,0,0,1,0, 0,1,0,1,1, 0,0,0,0,1, 0,0,1,0,0),byrow=TRUE,nrow=5,ncol=5)
X2_=matrix(c(0,1,0,1,1, 1,0,0,1,0, 0,0,0,0,0, 0,0,0,0,1, 0,0,1,0,0),byrow=TRUE,nrow=5,ncol=5)
 X=list(t(X1),t(X2),t(X2_))

 N=nrow(X1)
 Xs=list()
 for(s in 1:length(X)){
   aa=which(X[[s]]==1,arr.ind=TRUE)
   Xs[[s]]=Matrix::sparseMatrix(x=rep(1,nrow(aa)),i=aa[,1],j=aa[,2],dims=c(N,N))
 }

 print(Xs)

 rf=rescal(Xs,2)
 A=rf$A
 R=rf$R
 tmp=rescal_01(Xs,A,R,scale_fact=1.5)#generate 1.5*original number of triples
print(sprintf('Precision:%.4f, Recall:%.4f',tmp$tp/(tmp$tp+tmp$fp),tmp$tp/(tmp$tp+tmp$fn)))
```

`rescal_SO_Val` *calaculate scores Subj,Predicate,Obj*

Description

calaculate scores from RESCAL factorization for a pairse of subjects and objects indices on one predicate

Usage

```
rescal_SO_Val(R, A, Subj, P, Obj)
```

Arguments

R	core tensor resulting from RESCAL factorization (r by r by m).
A	Embedding matrix part resulting from RESCAL factorization.
Subj	integer vector containing indices of Subjects
P	index of predicate (slice in tensor)
Obj	integer vector containing inedices of Objects

Details

calaculate scores from RESCAL factorization for a pairse of subjects and objects indices on one predicate. Subj and Obj parameters must have the same length. rowSums(A[Subj,] %*% R[[p]] * A[Obj,])

Value

Data frame of four columns: Subj, Predicate, Obj, and val. val column is the score

Author(s)

Abdelmoneim Amer Desouki

See Also

rescal scRescal rescal_Trp_Val

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (R, A, Subj, P, Obj)
{
    if (length(Subj) != length(Obj)) {
        stop("rescal_SO_Val: List of subjects and objects must be of the same length")
    }
    val = rowSums((A[Subj, , drop = FALSE] %*% R[[P]]) * (A[Obj,
        , drop = FALSE]))
    return(data.frame(Subj, P, Obj, val))
  }
```

---

rescal_Trp_Val *RESCAL scores of triples*

Description

Calculate score values corresponding to triples in tensor from RESCAL factorization.

Usage

```
rescal_Trp_Val(R, A, ntnsr, Sl_ix = NULL, verbose = 1)
```

Arguments

R	core tensor resulting from RESCAL factorization (r by r by m).
A	Embedding matrix part resulting from RESCAL factorization.
ntnsr	sparse tensor containing the triples to be reconstructed (i.e. to calculate scores for).
Sl_ix	Optional: indexes of slices to be processed.
verbose	the level of messages to be displayed, 0 is minimal.

Value

Returns a data frame containing four columns:

Subj	index(es) of subject(s)
P	index(es) of predicate
Obj	index(es) of object(s)
val	the score resuling from A[Subj,]%*% R_p * A[Obj,].

Author(s)

Abdelmoneim Amer Desouki

See Also

rescal scRescal RescalReconstructBack

Examples

```
## Not run:
  library(RDFTensor)
  data('umls_tnsr')
  ntnsr=umls_tnsr

   #Calculate RECAL Factorization
    tt=rescal(ntnsr$X,rnk=10,ainit='nvecs',verbose=2,lambdaA=0,epsilon=1e-4,lambdaR=0)
    A=tt$A
    R=tt$R

    #Calculate scores
    res=rescal_Trp_Val(tt$R,tt$A,ntnsr)


    print(summary(res['val']))

## End(Not run)
```

scRescal *Scalable RESCAL Factorization*

Description

implements improvements of RESCAL to be able to factorize graaphs with millions of entities. Use parallelization and compact representation of sparse matrices. Modified error calculation criteria that is more efficient and is based on the change of A & R instead of calulating tensor difference.

Usage

```
scRescal(X, rnk, ainit = "nvecs", verbose = 2, Ainit = NULL, Rinit = NULL, lambdaA = 0,
lambdaR = 0, lambdaV = 0, epsilon = 0.001, maxIter = 100, minIter = 1, P = list(),
orthogonalize = FALSE, func_compute_fval = "compute_fit", retAllFact = FALSE,
useQR = FALSE, ncores = 0, OS_WIN = FALSE, savePath = "", oneCluster = TRUE,
useXc = FALSE, saveARinit = FALSE, saveIter = 0, dsname = "", maxNrows = 50000,
generateLog = FALSE)
```

Arguments

X	is a sparse tensor as set of sparse matrices, one for every relation (predicate). (a LIST of SparseMatrix)
rnk	The rank of the factorization
ainit	the method used to initialize matrix A
verbose	the level of messages to be displayed, 0 is minimal.
Ainit	the initial value of matrix A, can be used to continue factorization from previous results.
Rinit	the initial value of R (the core tensor, as LIST of frontal slices)
lambdaA	Regularization parameter for A factor matrix. 0 by default
lambdaR	Regularization parameter for R_k factor matrices. 0 by default
lambdaV	Regularization parameter for R_k factor matrices. 0 by default
epsilon	error threshold
maxIter	Maximum number of iterations
minIter	Minimum number of iterations
P	Not implemented
orthogonalize	Not implemented
func_compute_fval	function used to compute fit.
retAllFact	flag to return intermediate values of A & R
useQR	was suggested by Nickel in Factorizing Yago and implemented by Michail Huffman; found to be converging more slowly.
ncores	number of cores used to run in parallel, 0 means no paralellism

useXc	compact the sparse tensor: require more space but much faster.
saveIter	iterations in which A&R to be saved, default 0 :none
saveARinit	option to save init A and R
maxNrows	used as limit to decide if the compact form of the sparse matrix is to be dense matrix (#rows <maxNrows) or to be sparse used in updateA to consider the predicate having rows (in compact form) more than the given number as Big and hence return dense matrix. Default 50000.
OS_WIN	True when the operating system is windows, used to allow using Fork when running in parallel
savePath	optional path to save intermediate results into it.
oneCluster	Boolean flag indicating that one cluster will be used in different steps when running in parallel
dsname	the dataset name
generateLog	save output when running in parallel to log file in current directory.

Value

list(A=A, R=R, all_err, nitr=itr + 1, times=as.vector(exectimes) Returns a LIST of the following:

A	The matrix A of the factorization (n by r)
R	The core tensor R the factorization as r (rank) matrices of (r by r)
nitr	number of iterations
times	list of running times of each step.

Author(s)

Abdelmoneim Amer Desouki

References

-Maximilian Nickel, Volker Tresp, Hans-Peter-Kriegel, "A Three-Way Model for Collective Learning on Multi-Relational Data", ICML 2011, Bellevue, WA, USA

-Maximilian Nickel, Volker Tresp, Hans-Peter-Kriegel, "Factorizing YAGO: Scalable Machine Learning for Linked Data" WWW 2012, Lyon, France

-SynthG: mimicking RDF Graphs Using Tensor Factorization

See Also

cp_apr serial_parCube cp_nmu cp_als rescal rescal_Trp_Val

Examples

```
## Not run:
   data('umls_tnsr')
   ntnsr=umls_tnsr
   tt=scRescal(ntnsr$X,rnk=10,ainit='nvecs',verbose=1,lambdaA=0,epsilon=1e-4,
                   lambdaR=0,ncores = 2,OS_WIN = TRUE)

## End(Not run)
```

`serial_parCube` *Serial 3-mode ParCube algorithm for memory resident tensors*

Description

ParCube uses sampling to reduce the problem then apply one of the tensor factorization methods: `cp_apr` , `cp_als`, `cp_nmu` on the small tensor. It is suitable for large sparse tensors.

Usage

```
serial_parCube(X, R, sample_factor, repetitions, opts = NULL)
```

Arguments

`X`	is a sparse tensor (a LIST containing subs, vals and size)
`R`	The rank of the factorization
`sample_factor`	[s1 s2 s3] such that each sampled tensor is of size [I/s1 J/s2 K/s3]
`repetitions`	number of sampling repetitions
`opts`	a list containing the options for the algorithm like maxiters:maximum iterations, tol:tolerance .. etc.

Details

opts: structure that stores options of the algorithm. For default, leave blank or use 'default_parcube_options()'. opts.p: percentage of common indices opts.nonneg: nonnegativity constraint enforced (binary) opts.loss: loss function (opts: 'fro' for Frobenius norm 'kl' for KL-divergence) opts.weights: function of calculating sampling weights (opts: 'sum_abs' for sum of absolute values or 'sum_squares' for sum of squares) opts.normalize: normalize the factor matrices to unit norm per column (binary); opts.tolerance: the numerical tolerance of the algorithm (everything smaller than that is considered zero) opts.internal_tolerance: the tolerance of the solvers used interally opts.num_restarts: number of repetitions of each decomposition of each sample

Value

`lambda`	lambdas of Kruskal tensor
`u`	LIST of A, B and C factor matrices for mode 1, 2 and 3 respectively.

Author(s)

Abdelmoneim Amer Desouki

References

-Brett W. Bader, Tamara G. Kolda and others. MATLAB Tensor Toolbox, Version [v3.0]. Available online at https://www.tensortoolbox.org, 2015.

-Papalexakis, Evangelos E., Christos Faloutsos, and Nicholas D. Sidiropoulos. "Parcube: Sparse parallelizable tensor decompositions." Machine Learning and Knowledge Discovery in Databases. Springer Berlin Heidelberg, 2012. 521-536.

See Also

cp_apr cp_als rescal cp_nmu

Examples

```
subs=matrix(c(5,1,1,
              3,1,2,
              1,1,3,
              2,1,3,
              4,1,3,
              6,1,3,
              1,1,4,
              2,1,4,
              4,1,4,
              6,1,4,
              1,2,1,
              3,2,1,
              5,2,1),byrow=TRUE,ncol=3)

X=list(subs=subs,vals=rep(1,nrow(subs)),size=c(6,2,4))
set.seed(123)
 opts = default_parcube_options();
 opts[['loss']]='fro'
 opts[['nonneg']]='1'#nmu
 P1=serial_parCube(X,2,1,2,opts=opts)
```

---

spt_mttkrp *Matricized tensor times Khatri-Rao product for ktensor*

Description

V = MTTKRP(X,U,n) efficiently calculates the matrix product of the n-mode matricization of X with the Khatri-Rao product of all entries in U, a cell array of matrices, except the nth. The tensor is considered to be sparse.

Usage

```
spt_mttkrp(X, U, n)
```

Arguments

X	sparse tensor
U	list of matrices to be multiplied by X
n	the mode used

Value

a dense matrix of size nrow=X$size[n],ncol=R

Author(s)

Abdelmoneim Amer Desouki

References

-Brett W. Bader, Tamara G. Kolda and others. MATLAB Tensor Toolbox, Version [v3.0]. Available online at https://www.tensortoolbox.org, 2015.

---

tensor2mat *Reduce tensor to matrix*

Description

Reduce the tensor to one matrix by summing all slices efficiently.

Usage

```
tensor2mat(X,binary=FALSE,symmetrize=FALSE)
```

Arguments

X	List of tensor slices (Latent folding).
binary	when true returns binary matrix (i.e. ignores multiple connections between entities), default FALSE.
symmetrize	when true all relations are considered symetric before adding the slices, default FALSE.

Value

Adjacency matrix according to flags binary and symmetrize.

Author(s)

Abdelmoneim Amer Desouki

Tensor_error *RESCAL Tensor error*

Description

Calculates error (Eculidean distance) between X:sparse tensor (frontal slices) and the approximations as R :core tensor and matrix :A.

Usage

```
Tensor_error(X, A, R)
```

Arguments

X	
A	matrix A from RESCAL factorization (n by r).
R	core tensor from RESCAL factorization as set of frontal slices (r by r by r)

Value

return numeric value as the Eculidean distance between the two tensors.

Author(s)

Abdelmoneim Amer Desouki

See Also

rescal

Examples

```
X1=matrix(c(1,0,0,0,0, 0,1,0,0,0, 0,0,1,1,0, 0,0,0,0,1, 1,0,0,0,0),byrow=TRUE,nrow=5,ncol=5)
  X2=matrix(c(0,1,0,1,1, 1,0,0,1,0, 0,1,0,1,1, 0,0,0,0,1, 0,0,1,0,0),byrow=TRUE,nrow=5,ncol=5)
  X2_=matrix(c(0,1,0,1,1, 1,0,0,1,0, 0,0,0,0,0, 0,0,0,0,1, 0,0,1,0,0),byrow=TRUE,nrow=5,ncol=5)
   X=list(t(X1),t(X2),t(X2_))

   N=nrow(X1)
   Xs=list()
   for(s in 1:length(X)){
     aa=which(X[[s]]==1,arr.ind=TRUE)
     Xs[[s]]=Matrix::sparseMatrix(x=rep(1,nrow(aa)),i=aa[,1],j=aa[,2],dims=c(N,N))
   }

   print(Xs)

   rf=rescal(Xs,2)
   A=rf$A
   R=rf$R
```

```
Tensor_error(Xs,A,R)
```

---

tnsr2trp *sparse tensor to triples*

Description

convert a sparse tensor (e.g. created using getTensor) to set of triples.

Usage

```
tnsr2trp(tnsr)
```

Arguments

tnsr	a sparse tensor (e.g. created using getTensor)

Value

a matrix of three columns containing the triples

Author(s)

Abdelmoneim Amer Desouki

See Also

getTnsrijk getTensor

Examples

```
trp=rbind(
    cbind('Alex',  'loves', 'Don'),
    cbind('Alex',  'loves', 'Elly'),
    cbind('Alex',  'hates', 'Bob'),
    cbind('Don',   'loves', 'Alex'),
    cbind('Don',   'hates', 'Chris'),
    cbind('Chris', 'hates', 'Bob'),
    cbind('Bob',   'hates', 'Chris'),
    cbind('Elly',  'hates', 'Chris'),
    cbind('Elly',  'hates', 'Bob'),
    cbind('Elly',  'loves', 'Alex')
    )
######
# form tensor as a set of frontal slices (Predicate mode)
    tnsr=getTensor(trp)
    trp_=tnsr2trp(tnsr)
    #print(any(! paste(trp_[,1],trp_[,2],trp_[,3]) %in% paste(trp[,1],trp[,2],trp[,3])))
    #print(any(! paste(trp[,1],trp[,2],trp[,3]) %in% paste(trp_[,1],trp_[,2],trp_[,3])))
```

umls_tnsr *RDF tensor of UMLS small graph*

Description

The dataset is a list that contains UMLS dataset basic graph used by Nickel et al 2011 to test rescal .

Usage

```
data(umls_tnsr)
```

Format

slot X contains the frontal slices and SO contains the names of the entities and P contains the names of the predicates

References

-Maximilian Nickel, Volker Tresp, Hans-Peter-Kriegel, "A Three-Way Model for Collective Learning on Multi-Relational Data", ICML 2011, Bellevue, WA, USA

Index