PyTen Package Function Document

Qingquan Song, Hancheng Ge, Xing Zhao, Xiao Huang
James Caverlee, Xia Hu

Department of Computer Science & Engineering
Texas A&M University
{*song_3134,hge,zhaoxing623, xhuang, caverlee, xiahu*}*@tamu.edu*

November 27, 2017

Abstract

This is the first edition of class and function list in our PyTen package. Most of the operations are based on our Helios project requirement. Matlab Tensor toolbox Documentation and Python Tensor Library (pytensor) are used as reference.

1 Introduction

We divide this whole documentation into three parts:
Section 2. Class Description and Basic Operations. (Tenclass File)
Section 3. Tensor Decomposition & Completion Algorithm. (Method File)
Section 4. Other Functions. (Tools File)

2 Class Description and Basic Operations

2.1 Tensor Class

Tensor is a class that represents the n-dimensional array `numpy.ndarray`.
Example: t = pyten.tenclass.Tensor(data) where data is a numpy.ndarray.

1. Attributes (use ‘t’ as an example of a tensor)

   - `data` (`numpy.ndarray`)
     : n-dimensional array that the tensor object represents.
   - `shape` (`tuple`)
     : shape of the tensor.
   - `ndims` (`int`)
     : dimensions of the tensor.

2. Functions

Name \ Detail	type	call	Short Intro.
data	numpy.ndarray	t.data	N-dimensional array
shape	tuple	t.shape	Shape of the tensor ‘t’
ndims	int	t.ndims	Dimensions of the tensor ‘t’

- `__init__(self, data = None, shape = None)`
  : Constructor for tensor object. `data` is a n-dimensional array (numpy.ndarray) and `shape` is a shape (`tuple`, `list`, or `numpy.ndarray`) of the tensor.
- `size(self)`
  : Returns the number of elements in the tensor.
- `copy(self)`
  : Returns a deepcpoy of tensor object.
- `dimsize(self, idx=None)`
  : Returns the size of the specified dimension. `idx`(int) $\in [0, t.ndims-1]$
- `tosptensor(self)`
  : Returns the sptensor object that contains the same value with the tensor object.
- `permute(self, order=None)`
  : Returns a tensor permuted by the order specified. `order`(list): an arrangement of $[0, 1, 2, ..., t.ndims - 1]$
- `ipermute(self, order=None)`
  : Returns a tensor permuted by the inverse of the order specified. `order`(list): an arrangement of $[0, 1, 2, ..., t.ndims - 1]$
- `tondarray(self)`
  : Returns the numpy.ndarray object that contains the same value with the tensor.
- `ttm(self, mat = None, mode = None, option = None)`
  : Returns the product of the tensor and the given matrix that is a single 2-D array with `list` or `numpy.array`.
  - option=None $Result = mat * t_{dims}$
  - option='t' $Result = t_{dims} * mat$

  where t_{dims} is the mode-dims unfold of the tensor t. The result will be reverted to a tensor which has the same dimsize with the original tensor except the specified dimension of input `mode`.
- `ttv(self, vec = None, mode = None, opt = None)`
  : Returns the product of the tensor and the given vector that is a single 1-D array with `list` or `numpy.array`.
- `norm(self)`
  : Return the Frobenius norm of the tensor.
- `unfold(self, n = None)`
  : Return the mode-n unfold of the Tensor. `n` is a int.
- `nvecs(self, n = None, r = None)`
  : Return first r eigenvectors of the mode-n unfolding matrix.

2.2 Sptensor Class

Sptensor is a class that represents the sparse tensor (with many zero elements). It does not store the whole values of the tensor object but stores the non-zero values and the corresponding coordinates of them.
Example: spt = pyten.tenclass.Sptensor(subs, vals, siz)

1. Attributes (use 'spt' as an example of a sparse tensor)

    - `vals` (`numpy.ndarray`)
      : 1-dimensional array representing the non-zero values of the sparse tensor object.
    - `subs` (`tuple`)
      : 2-dimensional array of the coordinates that the values in original tensor object.
    - `shape` (`tuple`)
      : shape of the original tensor.
    - `ndims` (`int`)
      : dimensions of the original tensor.

Name \ Detail	type	call	Short Intro.
vals	numpy.ndarray	spt.vals	1-dimensional array of non-zero values of the sptensor
subs	numpy.ndarray	spt.subs	2-dimensional array of coordinates of the values in vals
shape	tuple	spt.shape	Shape of the original tensor
ndims	int	spt.ndims	Dimensions of the original tensor

2. Functions

    - `__init__(self, subs, vals, shape)`
      : (Re)constructor for a sptensor object where `subs` and `vals` are coordinates and values of the sparse tensor.
    - `nnz(self)`
      : Returns the number of non-zero elements in a sparse tensor object.
    - `copy(self)`
      : Returns a deepcopy of sparse tensor object.
    - `dimsize(self, idx=None)`
      : Returns the size of the specified dimension. `idx` $\in [0, spt.ndims-1]$
    - `totensor(self)`
      : Returns a new Tensor object that contains the same values.
    - `permute(self, order=None)`
      : Returns a new Sptensor permuted by the given order. `order` (`list`): an arrangement of $[0, 1, 2, ..., spt.ndims - 1]$
    - `ttm(self, mat = None, mode = None, option = None)`
      : Returns the product of the sparse tensor and the given matrix that is a single 2-D array with type `list` or `numpy.array`. If the result tensor's sparsity is larger than 0.5, it will return a sparse tensor. Otherwise it will return a full dense tensor.

2.3 Tenmat Class

Tenmat is a class that represents the matricized (2-dimensionally unfolded) version of a tensor object. Representation of tenmat depends on the row and column indices used for matricize the tensor.
Example: tmat=pyten.tenclass.Tenmat(X, rowIndices, colIndices, tensor_size)

1. Attributes (use 'tmat' as an example of a tenmat object)

   - `data` (`numpy.ndarray`)
     : 2-dimensional array representing the matricized tensor.
   - `shape` (`tuple`)
     : shape(size) of the original tensor.
   - `rowIndices` (`numpy.ndarray`)
     : 1-dimensional array of row indices used to matricize the tensor.
   - `colIndices` (`numpy.ndarray`)
     : 1-dimensional array of column indices used to matricize the tensor.

Detail / Name	type	call	Short Intro.
data	numpy.ndarray	tmat.data	2-dimensional array representing the matricized tensor
shape	tuple	tmat.shape	Shape of the original tensor
rowIndices	numpy.ndarray	tmat.rowIndices	1-dimensional array of row indices used to matricize the tensor
colIndices	numpy.ndarray	tmat.colIndices	1-dimensional array of column indices used to matricize the tensor

2. Functions

   - `__init__(self, X = None, rdim = None, cdim = None, tsiz = None`
     : (Re)constructor for a sptensor object where `subs` and `vals` are coordinates and values of the sparse tensor. `rdim` and `cdim` are the row and column indices to matricize the tensor. `tsize` is the size of the original tensor object.
   - `copy(self)`
     : Returns a copied tenmat object of the given tenmat.
   - `totensor(self)`
     : Returns a orignal tensor object of the tenmat.
   - `tondarray(self)`
     : Returns the (2-dimensional) `numpy.ndarray` object that has the same values with the tenmat.
   - `__str__(self)`
     : Returns the basic information of the tenmat object including three of its attributes: `tsize`, `rowIndices` and `colIndices`.

2.4 Sptenmat Class

Sptenmat is a class that represents the matricized (2-dimensionally unfolded) version of a sptensor object. Representation of sptenmat depends on the row and column indices used for matricize the sptensor.
Example: sptmat = pyten.tenclass.Sptenmat(sptmat, rowIndices, colIndices, tsize)

1. Attributes (use 'sptmat' as an example of a sparse matricized tensor)

   - `vals` (`numpy.ndarray`)
     : 1-dimensional array representing the non-zero values of the sparse tensor object.
   - `subs` (`tuple`)
     : 2-dimensional array of the coordinates that the values in original tensor object.
   - `shape` (`tuple`)
     : shape of the original tensor.
   - `rdim` (`numpy.ndarray`)
     : 1-dimensional array of row indices used to matricize the tensor.
   - `cdim` (`numpy.ndarray`)
     : 1-dimensional array of column indices used to matricize the tensor.

Detail / Name	type	call	Short Intro.
vals	numpy.ndarray	sptmat.vals	1-dimensional array of non-zero values of the sptensor
subs	numpy.ndarray	sptmat.subs	2-dimensional array of coordinates of the values in vals
shape	tuple	sptmat.shape	Shape of the original tensor
rdim	numpy.ndarray	sptmat.rdim	1-dimensional array of row indices used to matricize the tensor
cdim	numpy.ndarray	sptmat.cdim	1-dimensional array of column indices used to matricize the tensor

2. Functions

   - `__init__(self, X = None, rdim = None, cdim = None, tsiz = None`
     : (Re)constructor for a sptensor object where subs and vals are coordinates and values of the sparse tensor. `X` is a sparse tensor. `rdim` and `cdim` are the row and column indices to matricize the tensor. `tsize` is the size of the original tensor object.
   - `tosptensor(self)`
     : Returns the corresponding sparse tensor object of the sptenmat.
   - `__str__(self)`
     : Returns the basic information of the sptenmat object including its attribute `shape` and the number of its nonzero values.

2.5 Tucker Tensor Class

'ttensor' is a class that represents the Tucker decomposition of a tensor object. Tucker decomposition decomposes a tensor into a core tensor and a list of 2-dimensional factoring orthogonal matrices, whose number is same with the number of dimension of the original tensor object. If the core tensor gets multiplied by the factoring matrices, the original tensor object is returned.
Example: tucker= pyten.tenclass.Ttensor(core, Us)

1. Attributes (use 'tucker' as an example of a tucker tensor object)

   - `core` (`numpy.ndarray`)
     : core tensor of the ttensor object.
   - `us` (`tuple`)
     : list of 2-dimensional `numpy.ndarray` that are the factoring orthogonal matrices of the ttensor object.
   - `shape` (`tuple`)
     : shape of the original tensor.
   - `ndims` (`int`)
     : Dimensions of the original tensor.

Detail / Name	type	call	Short Intro.
core	tensor	tucker.core	core tensor of the ttensor object.
us	list of numpy.ndarray	tucker.us	list of 2-dimensional `numpy.ndarray` that are the factoring orthogonal matrices of the ttensor object.
shape	tuple	tucker.shape	shape of the original tensor.
ndims	int	tucker.ndims	Dimensions of the original tensor.

2. Function

   - `__init__(self, core, Us)`
     : create a tucker Tensor object with the `core` tensor and factorized `matrices`.
   - `size(self)`
     : Returns the number of elements of the original tensor object.
   - `copy(self)`
     : Returns a deepcpoy of tensor object.
   - `dimsize(self, idx=None)`
     : Returns the size of the dimension specified by `idx`. `idx`(int) $\in [0, t.ndims - 1]$
   - `totensor(self)`
     : Returns the reconstructed tensor object by multiple the core tensor with the decomposed orthogonal matrices.

2.6 Kruskal Tensor Class

'ktensor' is a class that represents the decomposition of a tensor as the sum of the outer products of the columns of matrices.(Kruskal/CP Decompostion)
Example: kruskal= pyten.tenclass.Ktensor(lmbda=None, Us=None)

1. Attributes (use 'kruskal' as an example of a kruskal tensor object)

   - `lmbda` (`numpy.ndarray`)
     : weight vector of each rank-1 tensor object in the CP decomposition.
   - `Us` (`tuple`)
     : list of 2-dimensional `numpy.ndarray` that are the factoring matrices of the ktensor object.
   - `shape` (`tuple`)
     : shape of the original tensor.
   - `ndim` (`int`)
     : Dimensions of the original tensor.
   - `rank` (`int`)
     : Rank of the decomposed tensor.

Detail / Name	type	call	Short Intro.
lmbda	numpy.ndarray	kruskal.lmbda	weight vector of each rank-1 tensor in the CP decompostion.
Us	list of numpy.ndarray	kruskal.Us	list of 2-dimensional `numpy.ndarray` that are the factoring matrices of the ktensor object
shape	tuple	kruskal.shape	shape of the original tensor.
ndim	int	kruskal.ndim	Dimensions of the original tensor.
rank	int	kruskal.rank	Rank of the decomposed tensor.

2. Function

   - `__init__(self, lmbda=None, Us=None)`
     : Create a kruskal tensor object with the weight vector and decomposition matrices. `lambda`: a list of weights of each rank-1 tensor object in a kruskal decompostion. `Us`: a list of arrays/matrices that are the factoring matrices of the ktensor object.
   - `size(self)`
     : Returns the number of elements of the original tensor object.
   - `totensor(self)`
     : Converts a Ktensor into a dense Tensor.
   - `tondarray(self)`
     : Converts a Ktensor into a dense multidimensional ndarray.
   - `norm(self)`
     : Efficient computation of the Frobenius norm for ktensors.

3 Tensor Decomposition & Completion Algorithm

Our current package contains ten methods for tensor decomposition or completion under three different scenarios:

- Basic Tensor Decomposition/Completion.,
- Tensor Decomposition/Completion with Auxiliary Information.,
- Dynamic Tensor Decomposition/Completion.
- Scalable Tensor Decomposition/Completion.

Methods are written in function format or class format.

3.1 Scenario 1. Basic Tensor Decomposition/Completion

- `cp_als(Y,R=20,Omega=None,tol=1e-4,maxiter=100,init='random',printitn=100)`

Intro: CP-ALS Method. Compute a Canonical Polyadic decomposition of a tensor using alternating least square optimization (or/and recover the tensor).

Input: `Y` - Tensor with Missing data
`R` - Rank of the tensor
`Omega` - Missing data Index Tensor
`tol` - Tolerance on difference in fit
`maxiters` - Maximum number of iterations
`init` - Initial guess ['random'‖'nvecs'‖'eigs']
`printitn` - Print fit every n iterations; 0 for no printing

Output: `P` - Decompose result.(kensor)
`X` - Recovered Tensor.

- `tucker_als(Y,R=20,Omega=None,tol=1e-4,maxiter=100,init='random',printitn=100)`

Intro: Tucker-ALS Method, also caled Higher-order orthogonal iteration (HOOI) method. Compute a tucker decomposition of a tensor using alternating least square optimization (or/and recover the tensor).

Input: `Y` - Tensor with Missing data
`R` - Rank of the tensor
`Omega` - Missing data Index Tensor
`tol` - Tolerance on difference in fit
`maxiters` - Maximum number of iterations
`init` - Initial guess ['random'‖'nvecs'‖'eigs']
`printitn` - Print fit every n iterations; 0 for no printing

Output: `T1` - Decompose result.(kensor)
`X` - Recovered Tensor.

- `silrtc(X,Omega=None,alpha=None,gamma=None,maxIter=100,epsilon=1e-5,printitn=100)`

Intro: Simple Low Rank Tensor Completion (SiLRTC). Reference: "Tensor Completion for Estimating Missing Values in Visual Data", PAMI, 2012.

Input: `X` - Tensor with Missing data
`Omega` - Missing data Index Tensor
`alpha` - The coefficient vector in the definition of tensor trace norm
`gamma` - The relaxation vector
`epsilon` - Tolerance on difference in fit
`maxIter` - Maximum number of iterations
`printitn` - Print fit every n iterations

Output: `X` - Recovered Tensor.

- `falrtc(X, Omega=None, alpha=None, mu=None, L=1e-5, C=0.6, maxIter=100, epsilon=1e-5,printitn=100)`

Intro: Fast Low Rank Tensor Completion (FaLRTC). Reference: "Tensor Completion for Estimating Missing Values in Visual Data", PAMI, 2012.

Input: `X` - Tensor with Missing data
`Omega` - Missing data Index Tensor
`alpha` - The coefficient vector in the definition of tensor trace norm
`mu` - The relaxation vector
`L` - The initial step size parameter, a positive number small enough, i.e., stepsize = $\frac{1}{L}$
`C` - The decreasing rate in the range $(0.5, 1)$
`epsilon` - Tolerance on difference in fit
`maxIter` - Maximum number of iterations
`printitn` - Print fit every n iterations

Output: `Y` - Recovered Tensor.

- `halrtc(X, Omega=None, alpha=None, beta=None, maxIter=100, epsilon=1e-5,printitn=100)`

Intro: High Accuracy Low Rank Tensor Completion (HaLRTC). Reference: "Tensor Completion for Estimating Missing Values in Visual Data", PAMI, 2012.

Input: `X` - Tensor with Missing data
`Omega` - Missing data Index Tensor
`alpha` - The coefficient vector in the definition of tensor trace norm
`beta` - The relaxation vector
`epsilon` - Tolerance on difference in fit
`maxIter` - Maximum number of iterations
`printitn` - Print fit every n iterations

Output: `X` - Recovered Tensor.

- `TNCP`.

Intro: Nuclear-norm regularized CP Tensor completion problem via Alternation Direction Method of Multipliers (ADMM).
Reference: 1. Yuanyuan Liu, Fanhua Shang, Hong Cheng, James Cheng, Hanghang Tong: Factor Matrix Trace Norm Minimization for Low-Rank Tensor Completion, SDM, 2014.
2. Yuanyuan Liu, Fanhua Shang, L. C. Jiao, James Cheng, Hong Cheng: Trace Norm Regularized CANDECOMP/PARAFAC Decomposition with Missing Data, IEEE Transactions on Cybernetics, 2015.

Attributes: `T` - Tensor with missing data
`X` - Recovered tensor
`omega` - Missing data index Tensor
`shape` - Shape of the tensor
`ndims` - Dimensions of the tensor
`rank` - Tensor Rank
`U` - CP decomposition matrices
`alpha` - Weight of trace norm terms for different modes
`lmbda` - Weights for regularization terms to prevent overfitting
`tol` - Tolerance on difference in fit
`maxIter` - Maximum number of iteraztions
`eta` - Weights for regularization terms in ADMM algorithm
`rho` - Constant to accelerate convergence
`errList` - Errlist to save the difference between fittings of two adjacent iteration
`Z` - Auxiliary matrices in ADMM algorithm
`Y` - Lagrange matrix multipliers in ADMM algorithm
`II` - Intermediate Diagonal tensor
`normT` - Norm of initial tensor T
`printitn` - Printing control

Function: `__init__(self, obser, omega=None, rank=20, tol=1e-5, maxIter=500, alpha=None, lmbda=None, eta=1e-4, rho=1.05, printitn=500)`
: Initialization stage of AirCP method. `obser` is the observed tensor.
`initializeLatentMatrices(self)`
: Initialization of all latent matrices.
`run(self)`
: Running (Optimization) stage for tensor completion.

- `dedicom`

Intro: ASALSAN Algorithm for DEDICOM proposed in : Bader, Brett W., Richard A. Harshman, and Tamara G. Kolda. ”Temporal analysis of semantic graphs using ASALSAN.” IEEE ICDM 2007.

Attributes: `T` - A Third-order Symmetric Tensor whose first and second modes are the same
`X` - Data of Tensor T (numpy.array format)
`shape` - Shape of the tensor
`ndims` - Dimensions of the tensor
`rank` - Tensor Rank

`maxiter` - Maximum number of iteraztions
`tol` - Tolerance on difference in fit
`printitn` - Printing control
`A` - The loading or embedding matrix for the first and second mode of T
`R` - Square matrix of size rank×rank captures the asymmetric relations
`D` - Tensor combined of K diagonal matrices, K is the shape of the last mode of T
`fit` - Fitted Tensor.
`errList` - Errlist to save the difference between fittings of two adjacent iteration.

Function: `__init__(self, x, rank=20, gamma=1e-1, lamb=1e-3, tol=1e-5, maxiter=500, printitn=100)`
: Initialization stage of Dedicom method. `initialize(self)`
: Random initialization of all decomposition matrices and tensors.
`run(self)`
: Running (Optimization) stage for Dedicom decomposition.

- `parafac2`

Intro: ALS Algorithm for PARAFAC2 proposed in : PARAFAC2-part i. A direct fitting algorithm for the PARAFAC2 model. Journal of Chemometrics, 13(3-4):275–294, 1999. Henk AL Kiers, Jos MF Ten Berge, and Rasmus Bro.

Attributes: `X` - A list of multiset matrices whose column size are the same
`K` - Number of matrices in the multiset
`L` - The Column size of all multiset matrices
`rank` - Decomposition Rank
`maxiter` - Maximum number of iteraztions
`tol` - Tolerance on difference in fit
`printitn` - Printing control
`U` - A list of row factor matrices
`H` - A square factor matrix of size rank×rank
`S` -Tensor combined of L diagonal matrices
`V` - The Column factor matrix
`fit` - Fitted Multiset Matrices.
`errList` - Errlist to save the difference between fittings of two adjacent iteration.

Function: `__init__(self, x, rank=20, gamma=1e-1, lamb=1e-3, tol=1e-5, maxiter=500, printitn=100)`
: Initialization stage of Parafac2 method. `initialize(self)`
: Random initialization of all decomposition matrices and tensors.
`run(self)`
: Running (Optimization) stage for Parafac2 decomposition.

3.2 Scenario 2. Tensor Decomposition/Completion with Auxiliary Information

- `AirCP`.

Intro: This routine solves the auxiliary information regularized CP Tensor completion via Alternation Direction Method of Multipliers (ADMM), which has been presented in our paper: Hancheng Ge, James Caverlee, Nan Zhang, Anna Squicciarini: Uncovering the Spatio-Temporal Dynamics of Memes in the Presence of Incomplete Information, CIKM, 2016.

Attributes: `T` - Tensor with missing data
`X` - Recovered tensor
`omega` - Missing data index Tensor
`shape` - Shape of the tensor
`ndims` - Dimensions of the tensor
`rank` - Tensor Rank
`U` - CP decomposition matrices
`simMats` - Similarity matrices
`L` - Laplacian matrices
`alpha` - Weight of auxiliary terms for different modes
`lmbda` - Weights for regularization terms to prevent overfitting
`tol` - Tolerance on difference in fit
`maxIter` - Maximum number of iteraztions
`eta` - Weights for regularization terms in ADMM algorithm
`rho` - Constant to accelerate convergence
`errList` - Errlist to save the difference between fittings of two adjacent iteration
`Z` - Auxiliary matrices in ADMM algorithm
`Y` - Lagrange matrix multipliers in ADMM algorithm
`II` - Intermediate Diagonal tensor
`normT` - Norm of initial tensor T
`printitn` - Printing control

Function: `__init__(self, obser, omega=None, rank=20, tol=1e-5, maxIter=500, simMats=None, alpha=None, lmbda=None, eta=1e-4, rho=1.05, printitn=500)`
: Initialization stage of AirCP method. `obser` is the observed tensor.
`initializeLatentMatrices(self)`
: Initialization of all latent matrices.
`run(self)`
: Running (Optimization) stage for tensor completion.

- `cmtf(X, Y=None, CM=None, R=2, Omega=None, tol=1e-4, maxiter=100, init='random', printitn=100)`

Intro: Compute Coupled Matrices and Tensor Factorization (and recover the Tensor).

Input: `X` - Tensor with Missing data
`Y` - Coupled Matries
`CM` - Shared Modes
`R` - Rank of the tensor
`Omega` - Missing data Index Tensor
`tol` - Tolerance on difference in fit
`maxiters` - Maximum number of iterations
`init` - Initial guess ['random'‖'nvecs'‖'eigs']
`printitn` - Print fit every n iterations; 0 for no printing

Output: `P` - Decompose result.(kensor)
`X` - Recovered Tensor.
`V` - Projection Matrices.

3.3 Scenario 3. Dynamic Tensor Decomposition/Completion

- `onlineCP`.

Intro: This routine solves the online Tensor decomposition problem using CP decomposition. Reference: Shuo Zhou, Nguyen Xuan Vinh, James Bailey, Yunzhe Jia, and Ian Davidson. 2016. Accelerating online CP decompositions for higher order tensors. KDD (2016).

Attributes: `T` - Tensor object for decomposition.
`shape` - Shape of the tensor.
`ndims` - Dimensions of the tensor.
`As` - CP decomposition matrices. (last mode is temporal mode)
`alpha` - The decomposed matrix of temporal mode at current timestamp.
`Ps` - Intermediate auxiliary matrices
`Qs` - Intermediate auxiliary matrices
`Ks` - Intermediate khatrirao product matrices
`H` - Intermediate Hadamard product matrices
`rank` - Tensor Rank
`X` - Reconstructed fitted whole tensor

Function: `__init__(self, initX, rank=20, tol=1e-8, printitn=100)` : Initialization stage of OnlineCP object. `initX` is the tensor used for initialization. `As` is a list of array contains the loading matrices of initX. `rank` is the tensor rank, `tol` is the tolerance on difference in fit. `printitn` is the printing control variable. `update(self, newX)` :Update stage of OnlineCP. `newX` is the newly arrived tensor at current time step.

- `OLSGD`.

Intro: This routine solves the three-order online tensor completion problem with CP decomposition and online Stochastic Gradient Descendant (SGD) optimization scheme. Reference: Morteza Mardani, Gonzalo Mateos, and Georgios B Giannakis.*Subspace learning and imputation for streaming big data matrices and tensors.* IEEE TSP (2015).

Attributes: `mu` - Step size of stochastic gradient descendant method
`rank` - Tensor Rank
`lmbda` - Time-varying factor λ for regularization term
`A` - Factorized Matrix (one of the non-temporal mode)
`B` - Factorized Matrix (one of the non-temporal mode)
`X` - Fitted whole tensor
`rec` - Recovered whole tensor
`timestamp` - The number of total timestamps
`fitx` - Fitted tensor at current timestamp
`recx` - Recovered whole tensor at current timestamp
`omega` - Current step missing data index tensor
`shape` - Shape(Size) of the whole tensor

Function: `__init__(self, rank=20, mu=0.01, lmbda=0.1)`
: Create a OLSGD object. `mu` is the step size of stochastic gradient descendant method, `rank` is the tensor rank, `lmbda` is the time-varying factor λ for regularization term.
`update(self, newX, omega=None, mut=0.01, lmbdat=0.1)`
: The update process of OLSGD method. `newX` is the newly arrived tensor. `omega` is the current step missing data index tensor.

- `MAST`.

Intro: This routine solves the multi-aspect streaming tensor completion problem with CP decomposition and ADMM optimization scheme. (Only at current timestep T, T is not 1) Reference: Song, Qingquan, et al. "Multi-Aspect Streaming Tensor Completion." Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2017.

Attributes: `C` - A dict to save seperate tensors. (dim=N, size=2^N)
`OmegaC` - A dict to save observation index subtensors.
`As` - Old decomposition matrices.
`olddims` - Dimension of the tensor in former time step.
`newdims` - Dimension of the tensor in current time step.
`idx` - Block (subtensor) index.
`rank` - Tensor Rank.
`lmbda` - Regularization penalty for factorization (this penalty is incorported into 'alpha' in paper)
`alpha`-Regularization penalty for trace norm term.
`maxiter` - Max iteration.
`tol`- Convergence tolerance.
`mu` - Forgetting factor.
`eta` - step size.
`rho` - increasing factor.
`printitn` - is the printing control variable.
`N` - Tensor dimension.
`Inc_size` - increasing size. (newdims-olddims)
`CAs1` - A list of former decomposition matrices.
`CAs2` - A list of incremental decomposition matrices.

`Zs` - Auxiliary matrices in ADMM.
`Ys` - Lagrange parameter matrices in ADMM.
`finalAs` -Final decomposition matrices in current timestep. (not current iteration)

Function: `__init__(self, init=None, omega=None, decomp=None, olddims=np.zeros(3), newdims=np.zeros(3), idx=np.ones([8, 3]), rank=2, lmbda=1, alpha=np.ones(3) / 3, maxiter=500, tol=1e-5, mu=1, dimchange=np.zeros(3), timestep=1, eta=1e-4, rho=1.05, printitn=500)`
: Initialization stage of MAST.

`init` - A dict to save seperate tensors. (dim=N, size=2^N) `omega` - A dict to save observation index subtensors. `decomp` - Old decomposition matrices. `olddims` - Dimension of the tensor in former time step `newdims` - Dimension of the tensor in current time step `idx` - Block (subtensor) index. `rank` - Tensor Rank. `lmbda` - Regularization penalty for factorization (this penalty is incorported into 'alpha' in paper) `alpha`-Regularization penalty for trace norm term `maxiter` - Max iteration. `tol`- Convergence tolerance. `mu` - Forgetting factor `eta` - step size `rho` - increasing factor `printitn` - is the printing control variable

`update(self)`
: Update stage of MAST.

3.4 Scenario 4. Scalable Tensor Decomposition/Completion

- `DistTensor.`

Intro: This routine solves the 3rd-order Tensor decomposition problem using Distributed CP ALS method.

Attributes: `dir_data` - dictionary of data.
`I,J,K` - Size of each dimension of the tensor.
`size` - Size of the tensor.
`nnz` - number of non-zero elements.
`numIBlocks, numJBlocks,numKBlocks` - The number of blocks used to ditribute the factorized matrices.
`rank` - Tensor rank
`maxIteration` - maximum iteration.
`tol` - error tolerance
`intermediateRDDStorageLevel` - Intermediate RDD storage level.
`finalRDDStorageLevel` - Final storage level
`iFactors,jFactors,kFactors` - Distributed factorized matrices.
`Us` - Combined final factorized matrices.
`lambdaVals` - Weights of each rank-1 tensor.
`ktensor` - Reconstructed kruskal tensor object.

Core Functions: `__init__(self, dir_data = None, appName = ''TensorDecomposition_ALS'', I = 100, J = 100, K = 100, rank = 10, numIBlocks = 5, numJBlocks = 5, numKBlocks = 5, maxIteration = 10, errorTolerance = 0.000001)`

: Initialization stage of DistTensor object. `initFactors(self, outBlocks)`
:Initialize factor matrices for a dimension.

`makeBlock(self, Blocks, mode, partitioner)`
: Generate the information of which blocks an index in a dimension is located in.

`computeAtA(self, factors)`
: Calculate AtA for a factor matrix.
`computeAtAInverse(self, AtA1, AtA2)`
: Calculate the inverse of the multiplication of two factor matrices.
`computeFactors(self, mode, tensorBlocks, factors1, outBlock1, factors2, outBlock2, AtAInverse)`
: Update a factor matrix.

`computeNormOfEstimatedTensor(self, iAtA, jAtA, kAtA, lambdaVals)`
: Compute the norm of the estimated tensor.

`calculateError(self, mkktrp, factors, normData, normEst, lambdaVals)`
: Calculate the error between the original and estimated tensors.

`columnNormalization(self, factors, iteration)`
: Column-wise normalize the factor matrix.

`run(self)`
: Run this algorithm.

- `DistTensorADMM.`

Intro: This routine solves the 3rd-order Tensor decomposition problem using Distributed CP ADMM method.

Attributes: `dir_data` - dictionary of data.
`I,J,K` - Size of each dimension of the tensor.
`size` - Size of the tensor.
`nnz` - number of non-zero elements.
`numIBlocks, numJBlocks,numKBlocks` - The number of blocks used to ditribute the factorized matrices.
`rank` - Tensor rank
`maxIteration` - maximum iteration.
`tol` - error tolerance
`intermediateRDDStorageLevel` - Intermediate RDD storage level.
`finalRDDStorageLevel` - Final storage level

`regAlpha` - Weight of auxiliary terms for different modes
`regLambda` - Weights for regularization terms to prevent overfitting
`regEta` - Weights for regularization terms in ADMM algorithm
`maxEta` - maximum eta.
`rho` - Constant to accelerate convergence.

`iFactors,jFactors,kFactors` - Distributed factorized matrices.
`Us` - Combined final factorized matrices.
`lambdaVals` - Weights of each rank-1 tensor.
`ktensor` - Reconstructed kruskal tensor object.

Core Functions: `__init__(self)`
: Initialization stage of DistTensor object. `initFactors(self, dim, sc, partitioner, label='factor')`
:Initialize factor matrices for a dimension.

`makeBlock(self, Blocks, mode, partitioner)`
: Generate the information of which blocks an index in a dimension is located in.

`computeAtA(self, factors)`
: Calculate AtA for a factor matrix.
`computeAtAInverse(self, AtA1, AtA2)`
: Calculate the inverse of the multiplication of two factor matrices.
`computeFactors(self, mode, tensorBlocks, factors1, outBlock1, factors2, outBlock2, Z, Y, AtAInverse)`
: Update a factor matrix.

`computeDualVariable(self, factors, Y)`
: Update the dual factor matrices Zs in ADMM.

`computeLargMultiplier(self, Y, factors, Z)`
: Update the Lagrange multiplier matrices Ys in ADMM.

`computeNormOfEstimatedTensor(self, iAtA, jAtA, kAtA, lambdaVals)`
: Compute the norm of the estimated tensor.

`calculateError(self, mkktrp, factors, normData, normEst, lambdaVals)`
: Calculate the error between the original and estimated tensors.

`columnNormalization(self, factors, iteration)`
: Column-wise normalize the factor matrix.

`run(self)`
: Run this algorithm.

- `DistTensorCompletionADMM.`

Intro: This routine solves the 3$^{\text{rd}}$-order Tensor completion problem using the HELIOS DistTenC method.

Attributes: `dir_data` - dictionary of data.
`I,J,K` - Size of each dimension of the tensor.
`size` - Size of the tensor.
`nnz` - number of non-zero elements.

`numIBlocks, numJBlocks,numKBlocks` - The number of blocks used to ditribute the factorized matrices.
`rank` - Tensor rank
`maxIteration` - maximum iteration.
`tol` - error tolerance
`intermediateRDDStorageLevel` - Intermediate RDD storage level.
`finalRDDStorageLevel` - Final storage level

`regAlpha` - Weight of auxiliary terms for different modes
`regLambda` - Weights for regularization terms to prevent overfitting
`regEta` - Weights for regularization terms in ADMM algorithm
`maxEta` - maximum eta.
`rho` - Constant to accelerate convergence.
`iFactors,jFactors,kFactors` - Distributed factorized matrices.
`Us` - Combined final factorized matrices.
`lambdaVals` - Weights of each rank-1 tensor.
`ktensor` - Reconstructed kruskal tensor object.

Core Functions: `__init__(self)`
: Initialization stage of DistTensor object. `initFactors(self, dim, sc, partitioner, label='factor')`
:Initialize factor matrices for a dimension.

`makeBlock(self, Blocks, mode, partitioner)`
: Generate the information of which blocks an index in a dimension is located in.

`computeAtA(self, factors)`
: Calculate AtA for a factor matrix.

`computeUtU(self, AtA1, AtA2)`
: Calculate UtU by element-wise multiplying all AtAs excluding the current dimension.

`computeFactors(self, mode, factors1, tensorBlocks, factors2, outBlock2, factors3, outBlock3, Z, Y, UtU, Pt, sc, iter_idx)`
: Update a factor matrix.

`computeDualVariable(self, factors, Y)`
: Update the dual factor matrices Zs in ADMM.

`computeLargMultiplier(self, Y, factors, Z)`
: Update the Lagrange multiplier matrices Ys in ADMM.

`computeNormOfEstimatedTensor(self, iAtA, jAtA, kAtA, lambdaVals)`
: Compute the norm of the estimated tensor.

`calculateError(self, Pt, normData, lambdaVals)`
: Calculate the error by directly computing the square root of the

residual tensor.

`columnNormalization(self, factors, iteration)`
: Column-wise normalize the factor matrix.

`initResidualTensor(self, tensorBlocks)`
: Initialize the residual tensors.

`computeResidualTensor(self, tensorBlocks, factors1, factors2, factors3, outBlock1, outBlock2, outBlock3, sc)`
: Calculate the residual tensor.

`run(self)`
: Run this algorithm.

4 Other Functions (Tools File)

- `create(problem='basic', siz=None, r=2, miss=0, tp='CP', aux=None, timestep=5)`

Intro: A function to create a Tensor decomposition or completion problem corresponding to three scenarios: basic tensor completion, tensor completion with auxiliary information and dynamic tensor completion.

Input: `problem` - Tensor completion/decomposition problem (basic,auxiliary,dynamic)
`siz` - Size of tensor
`r` - Rank of the tensor
`miss` - Missing percentage of the entries in the synthetic tensor
`tp` - Type of expect solution (basic:Tucker, CP; auxiliary: sim, couple)
`aux` - Predefined auxiliary similarity matrices or coupled matrices
`timestep` - Total number of timesteps for dynamic tensor completion

Output: `T` - Generated tensor
`Omega` - Missing data indicating tensor (0: Miss; 1:Exist);
`sol` - solution of the tensor completion problem. (e.g. a Ktensor object)

- `tenerror(fitx, realx, omega)`Calculate three Kinds of Error of between a completed tensor (`fitx`) and the ground truth tensor (`realx`). `omega` is the missing entry index tensor. The three kind of errors include one absolute error and two relative errors.
- `khatrirao(U)`
  : Calculate The Khatrirao Product of a series of matrices.
- `tendiag(v, sz)`
  : Create a Diagonal Tensor of Size `sz` with Diagnal Values `v`

- `mttkrp(x, u, n)`
  : Calculate Unew $= x_(n) \times$ khatrirao(all u except n in a reverse order).

⋆ `[tools]`
  : A module contains other small tool functions.