PyTen Package User Manual

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

1 Quick Start

1.1 Import Package

```
# Require pandas & numpy packages
import pyten
```

1.2 Call the Guide Function

```
[OriTensor, DeTensor, TenClass, RecTensor] = pyten.UI.helios()
    #OriTensor: Original Tensor
    #DeTensor: Decomposed tensor.
    #TenClass: ttensor or ktensor which contains the decomposition matrices
    #RecTensor: Recovered tensor.
```

1.3 Choose the Scenario:

```
Please choose the Scenario:
 1. Basic Tensor Decomposition/Completion  2.Tensor Completion with Auxiliary
                                           Information 3.Dynamic Tensor
                                           Decomposition 0.Exit
>>> 1
```

1.4 Loading an dataset

```
Please input the file_name of the data: (if it is multiset data, please seperate
                                          them to different files and input the
                                          first one)
>>> ./test/syntensor.csv
```

1.5 Choose the Method from Given Method Pool

```
Please choose the method you want to use to recover data(Input one number):
 1. Tucker(ALS)  2.CP(ALS) 3.TNCP(Trace Norm + ADMM, Only For Recovery) 4.SiLRTC(
                                           Only For Recovery) 5.FaLRTC(Only For
                                           Recovery) 6.HaLRTC(Only For Recovery) 7.
                                            PARAFAC2 8.DEDICOM 0.Exit
> 1
```

1.6 Choose to Recover or Just Decompose

```
If there are missing values in the file? (Input one number)
1. Yes, recover it  2.No, just decompose (Missing entries in the original tensor
                                          will be replaced by 0)
 0.Exit
> 1
```

1.7 Running Results

```
The original Tensor is:
[[[    nan      nan      nan      nan]
  [ 0.0193      nan  0.023   0.0138]]
 [[    nan  0.0209      nan  0.0192]
  [ 0.2244  0.4468  0.4514  0.2389]]
 [[    nan  0.4337  0.3854      nan]
  [ 0.0098  0.0198  0.0199  0.0105]]]

The Recovered Tensor is:
[[[  5.36313536e-03   1.15882448e-02   2.43375570e-02   5.87269923e-03]
  [  1.93000000e-02   3.13159793e-02   2.30000000e-02   1.38000000e-02]]
 [[ -2.28098016e-03   2.09000000e-02   1.67983832e+00   1.92000000e-02]
  [  2.24400000e-01   4.46800000e-01   4.51400000e-01   2.38900000e-01]]
 [[ -4.94710294e-04   4.33700000e-01   3.85400000e-01   4.16284014e-03]
  [  9.80000000e-03   1.98000000e-02   1.99000000e-02   1.05000000e-02]]]
```

1.8 Print the Results in Python

```
print DeTensor.data  # Full Tensor reconstructed by decomposed matrices
[[[  5.36313536e-03   1.15882448e-02   2.43375570e-02   5.87269923e-03]
  [  1.60599120e-02   3.13159793e-02   2.29682316e-02   1.69444977e-02]]
 [[ -2.28098016e-03   1.09703113e-01   1.67983832e+00   1.92196182e-02]
  [  2.16599608e-01   4.30870483e-01   4.35287581e-01   2.30142749e-01]]
 [[ -4.94710294e-04   2.37633650e-02   3.63895371e-01   4.16284014e-03]
  [  4.69195225e-02   9.33346671e-02   9.42924456e-02   4.98532330e-02]]]

print TenClass  # Final Decomposition Results e.g. Ttensor or Ktensor
Ttensor of size (3, 2, 4)
Core = Tensor of size (2, 2, 2) with 8 elements.

u[0] =
[[-0.02159847 -0.99976672]
 [-0.9771037   0.02108615]
 [-0.2116645   0.00467759]]
u[1] =
[[ 0.95730904 -0.28906643]
 [ 0.28906643  0.95730904]]
u[2] =
[[ 0.03461903 -0.42542392]
 [ 0.13126046 -0.77881066]
 [ 0.98955031  0.13976448]
 [ 0.04860447 -0.43924291]]
```

1.9 Result in File

The recovered data are saved in the same dictionary named “synthensor_Recover.csv”.

1.10 Recovering Missing Data

	A	B	C	D	E
1	x1	x2	x3	r	
2	1	1	1		
3	2	2	1	0.2244	
4	3	1	1		
5	1	2	1	0.0193	
6	2	1	1		
7	3	2	1	0.0098	
8	1	1	2		
9	2	2	2	0.4468	
10	3	1	2	0.4337	
11	1	2	2		
12	2	1	2	0.0209	
13	3	2	2	0.0198	
14	1	1	3		
15	2	2	3	0.4514	
16	3	1	3	0.3854	
17	1	2	3	0.023	
18	2	1	3		
19	3	2	3	0.0199	
20	1	1	4		
21	2	2	4	0.2389	
22	3	1	4		
23	1	2	4	0.0138	
24	2	1	4	0.0192	
25	3	2	4	0.0105	

	A	B	C	D	E
1	x1	x2	x3	r	
2	1	1	1	0.00634358	
3	2	2	1	0.2244	
4	3	1	1	-0.0002772	
5	1	2	1	0.0193	
6	2	1	1	-0.0010754	
7	3	2	1	0.0098	
8	1	1	2	0.01372454	
9	2	2	2	0.4468	
10	3	1	2	0.4337	
11	1	2	2	0.03115167	
12	2	1	2	0.0209	
13	3	2	2	0.0198	
14	1	1	3	0.02478928	
15	2	2	3	0.4514	
16	3	1	3	0.3854	
17	1	2	3	0.023	
18	2	1	3	1.37089693	
19	3	2	3	0.0199	
20	1	1	4	0.00691875	
21	2	2	4	0.2389	
22	3	1	4	0.00492622	
23	1	2	4	0.0138	
24	2	1	4	0.0192	
25	3	2	4	0.0105	

Figure 2: Comparison Between Original Data & Data After Recovering

2 API Description (Three UI Functions For Three Scenarios)

```
# Scenario 1: Basic Tensor completion or decomposition.
[OriTensor, DeTensor, TenClass, RecTensor] = pyten.UI.basic()

# Scenario 2: Tensor completion or decomposition with auxiliary information
[OriTensor, DeTensor, TenClass, RecTensor] = pyten.UI.auxiliary()

# Scenario 3: Dynamic/Online/Streaming Tensor completion or decomposition
[OriTensor, DeTensor, TenClass, RecTensor] = pyten.UI.dynamic()
```

3 Usage of Main Functions

3.1 Create A Decomposition/Completion Problem <Function: create>

```
# 1. Create Tensor Completion Problem
from pyten.tools import create  # Import the problem creation function

problem = 'basic'  # Problem Definition
siz = [20, 20, 20]  # Size of the Created Synthetic Tensor
r = [4, 4, 4]  # Rank of the Created Synthetic Tensor
miss = 0.8  # Missing Percentage
tp = 'CP'  # Solution Format (Creating Method) of the Created Synthetic Tensor
[X1, Omega1, sol1] = create(problem, siz, r, miss, tp)

#X1: The created tensor object.

#Omega1: An 0-1 'numpy.ndarray' which represent the missing data. (0 for missing)

#sol1: Solution of the created problem.
```

3.2 Scenario 1: Basic Tensor Completion/Decomposition

3.2.1 Solve Synthetic Completion Problem

```
from pyten.tools import create  # Import the problem creation function

problem = 'basic'  # Define Problem As Basic Tensor Completion Problem
siz = [20, 20, 20]  # Size of the Created Synthetic Tensor
r = [4, 4, 4]  # Rank of the Created Synthetic Tensor
miss = 0.8  # Missing Percentage
tp = 'CP'  # Define Solution Format of the Created Synthetic Tensor As 'CP
                                            decomposition'
[X1, Omega1, sol1] = create(problem, siz, r, miss, tp)

# Basic Tensor Completion with methods: CP-ALS,Tucker-ALS, FaLRTC, SiLRTC, HaLRTC,
                                            TNCP
from pyten.method import *

r = 4  # Rank for CP-based methods
R = [4, 4, 4]  # Rank for tucker-based methods
# CP-ALS
[T1, rX1] = cp_als(X1, r, Omega1)  # if no missing data just omit Omega1 by using [
                                            T1,rX1]=cp_als.cp_als(X1,r)
# print sol1.totensor().data
# print rX1.data

# Tucker-ALS
[T2, rX2] = tucker_als(X1, R, Omega1)  # if no missing data just omit Omega1
# FalRTC, SiLRTC, HaLRTC
rX3 = falrtc(X1, Omega1)
rX4 = silrtc(X1, Omega1)
```

```
rX5 = halrtc(X1, Omega1)
# TNCP
self1 = TNCP(X1, Omega1, rank=r)
self1.run()

# Error Testing
from pyten.tools import tenerror

realX = sol1.totensor()
[Err1, ReErr11, ReErr21] = tenerror(rX1, realX, Omega1)
[Err2, ReErr12, ReErr22] = tenerror(rX2, realX, Omega1)
[Err3, ReErr13, ReErr23] = tenerror(rX3, realX, Omega1)
[Err4, ReErr14, ReErr24] = tenerror(rX4, realX, Omega1)
[Err5, ReErr15, ReErr25] = tenerror(rX5, realX, Omega1)
[Err6, ReErr16, ReErr26] = tenerror(self1.X, realX, Omega1)
print '\n', 'The Relative Error of the Six Methods are:', ReErr21, ReErr22, ReErr23
                                                    , ReErr24, ReErr25, ReErr26
```

3.2.2 Real Problem - Image Recovery

```
import matplotlib.image as mpimg  # Use it to load image
import numpy as np

lena = mpimg.imread("./test/testImg.png")
im = np.double(np.uint8(lena * 255))
im = im[0:50, 0:50, 0:3]

from pyten.tenclass import Tensor  # Use it to construct Tensor object

X1 = Tensor(im)  # Construct Image Tensor to be Completed
X0 = X1.data.copy()
X0 = Tensor(X0)  # Save the Ground Truth
Omega1 = (im < 100) * 1.0  # Missing index Tensor
X1.data[Omega1 == 0] = 0

# Basic Tensor Completion with methods: CP-ALS, Tucker-ALS, FaLRTC, SiLRTC, HaLRTC,
                                                       TNCP
from pyten.method import *

r = 10
R = [10, 10, 3]  # Rank for tucker-based methods
[T1, rX1] = cp_als(X1, r, Omega1, maxiter=1000, printitn=100)
[T2, rX2] = tucker_als(X1, R, Omega1, max_iter=100, printitn=100)
alpha = np.array([1.0, 1.0, 1e-3])
alpha = alpha / sum(alpha)
rX3 = falrtc(X1, Omega1, max_iter=100, alpha=alpha)
rX4 = silrtc(X1, Omega1, max_iter=100, alpha=alpha)
rX5 = halrtc(X1, Omega1, max_iter=100, alpha=alpha)
self1 = TNCP(X1, Omega1, rank=r)
self1.run()

# Error Testing
from pyten.tools import tenerror

realX = X0
[Err1, ReErr11, ReErr21] = tenerror(rX1, realX, Omega1)
[Err2, ReErr12, ReErr22] = tenerror(rX2, realX, Omega1)
[Err3, ReErr13, ReErr23] = tenerror(rX3, realX, Omega1)
[Err4, ReErr14, ReErr24] = tenerror(rX4, realX, Omega1)
[Err5, ReErr15, ReErr25] = tenerror(rX5, realX, Omega1)
[Err6, ReErr16, ReErr26] = tenerror(self1.X, realX, Omega1)
print '\n', 'The Relative Error of the Six Methods are:', ReErr21, ReErr22, ReErr23
                                                    , ReErr24, ReErr25, ReErr26
```

3.3 Scenario 2: Tensor Completion/Decomposition with Auxiliary Information

3.3.1 Use AirCP Method to solve Tensor Completion With Auxiliary Similarity Matrices

```
from pyten.method import AirCP  # Import AirCP
from pyten.tools import create  # Import the problem creation function

problem = 'auxiliary'  # Define Problem As Basic Tensor Completion Problem
siz = [20, 20, 20]  # Size of the Created Synthetic Tensor
r = [4, 4, 4]  # Rank of the Created Synthetic Tensor
miss = 0.8  # Missing Percentage
tp = 'sim'  # Define Auxiliary Information As 'Similarity Matrices'


# Construct Similarity Matrices (if 'None', then it will use the default Similarity
                                              Matrices)
# aux = [np.diag(np.ones(siz[n]-1), -1)+np.diag(np.ones(siz[n]-1), 1) for n in
                                              range(dims)]
aux = None
[X1, Omega1, sol1, sim_matrices] = create(problem, siz, r, miss, tp, aux=aux)

self = AirCP(X1, Omega1, r, sim_mats=sim_matrices)
self.run()

# Error Testing
from pyten.tools import tenerror

realX = sol1.totensor()
[Err1, ReErr11, ReErr21] = tenerror(self.X, realX, Omega1)
print '\n', 'The Relative Error of the Two Methods are:', ReErr11
```

3.3.2 Use CMTF Method to solve Tensor Completion With Coupled Matrices

```
from pyten.method import cmtf
from pyten.tools import create  # Import the problem creation function
import numpy as np

problem = 'auxiliary'  # Define Problem As Basic Tensor Completion Problem
siz = [20, 20, 20]  # Size of the Created Synthetic Tensor
r = [4, 4, 4]  # Rank of the Created Synthetic Tensor
miss = 0.8  # Missing Percentage
tp = 'couple'  # Define Auxiliary Information As 'Similarity Matrices'


# Construct A Coupled Matrix and Tensor Completion Problem
dims = 3
[X1, Omega1, sol1, coupled_matrices] = create(problem, siz, r, miss, tp)

[T1, Rec1, V1] = cmtf(X1, coupled_matrices, [1, 2, 3], r, Omega1, maxiter=500)
fit_coupled_matrices_1 = [np.dot(T1.Us[n], V1[n].T) for n in range(dims)]

# Error Testing
from pyten.tools import tenerror

realX = sol1.totensor()
[Err1, ReErr11, ReErr21] = tenerror(Rec1, realX, Omega1)
print '\n', 'The Relative Error of the Two Methods are:', ReErr11, ReErr12
```

3.4 Scenario 3: Dynamic Tensor Decomposition/Completion

```
from pyten.method import onlineCP, OLSGD
from pyten.tools import create  # Import the problem creation function
from pyten.tools import tenerror
import numpy as np

problem = 'dynamic'  # Define Problem As Dynamic Tensor Completion Problem
time_steps = 10  # Define the Number of Total Time Steps
siz = np.array([[1, 50, 50] for t in range(time_steps)])
r = [4, 4, 4]  # Rank of the Created Synthetic Tensor
miss = 0.8  # Missing Percentage
# Create a Dynmaic Tensor Completion Problem
[X1, Omega1, sol1, siz, time_steps] = create(problem, siz, r, miss, timestep=
                                          time_steps)

for t in range(time_steps):
    if t == 0:  # Initial Step
        print('Initial Step\n')
        self1 = OLSGD(rank=r, mu=0.01, lmbda=0.1)  # OLSGD assume time is the first
                                                  mode.
        self1.update(X1[t], Omega1[t])  # Complete the initial tensor using OLSGD
                                         method.
        # onlineCP assume time is the last mode.
        self = onlineCP(X1[t].permute([1, 2, 0]), rank=r, tol=1e-8, printitn=0)  #
                                         Just decompose without completion
                                          using onlineCP
    else:
        if t==1:
            print('Update Step\n')
        self1.update(X1[t], Omega1[t])  # Update Decomposition as well as
                                         Completion using OLSGD.
        self.update(X1[t].permute([1, 2, 0]))  # Update Decomposition of onlineCP.
    # Test Current Step OLSGD Completion Error
    realX = sol1[t].totensor()
    [Err1, ReErr11, ReErr21] = tenerror(self1.recx, realX, Omega1[t])
    print 'OLSGD Recover Error at Current Step:', Err1, ReErr11, ReErr21
```

3.5 Scenario 4: Scalable Tensor Completion/Decomposition

```
# 1. Solve Synthetic Completion Problem
from pyten.tools import create  # Import the problem creation function

problem = 'basic'  # Define Problem As Basic Tensor Completion Problem
siz = [20, 20, 20]  # Size of the Created Synthetic Tensor
r = [4, 4, 4]  # Rank of the Created Synthetic Tensor
miss = 0.8  # Missing Percentage
tp = 'CP'  # Define Solution Format of the Created Synthetic Tensor As 'CP
                                          decomposition'
[X1, Omega1, sol1] = create(problem, siz, r, miss, tp)

# Basic Tensor Completion with methods: CP-ALS,Tucker-ALS, FaLRTC, SiLRTC, HaLRTC,
                                          TNCP
from pyten.method import *

r = 4  # Rank for CP-based methods
R = [4, 4, 4]  # Rank for tucker-based methods

# Distributed CP_ALS
self0 = TensorDecompositionALS()
self0.dir_data = X1  # Could also be '.csv' or '.txt' format, e.g. 'test/syntensor.
                                          csv', 'test/tensor_10x10x10_101.txt'
self0.rank = r
self0.run()

# DistTensorADMM
self1 = DistTensorADMM()
```

```
self1.dir_data = X1  # Could also be '.csv' or '.txt' format, e.g. 'test/syntensor.
                                          csv', 'test/tensor_10x10x10_101.txt'
self1.rank = r
self1.run()

# DistTensorCompletionADMM
self2 = DistTensorCompletionADMM()
self2.dir_data = X1  # Could also be '.csv' or '.txt' format, e.g. 'test/syntensor.
                                          csv', 'test/tensor_10x10x10_101.txt'
self2.rank = r
self2.run()

# Error Testing
from pyten.tools import tenerror
realX = sol1.totensor()
[Err1, ReErr11, ReErr21] = tenerror(self0.ktensor.totensor(), realX, Omega1)
[Err2, ReErr21, ReErr22] = tenerror(self1.ktensor.totensor(), realX, Omega1)
RecTensor = self2.ktensor.totensor().data*(1-Omega1)+X1.data*Omega1
[Err3, ReErr31, ReErr32] = tenerror(RecTensor, realX, Omega1)
print '\n', 'The Relative Error of the Three Distributed Methods are:', ReErr21,
                                          ReErr22, ReErr32
```

3.6 Scenario *: Other Decomposition Method

3.6.1 Parafac2

```
# Create multiset
from pyten.method import parafac2  # Import the problem creation function
from pyten.tools import create  # Import the problem creation function

problem = 'basic'  # Define Problem As Basic Tensor Completion Problem
siz = [30, 50, 40]  # Size of the Created Synthetic Tensor
r = 5  # Rank of the Created Synthetic Tensor
miss = 0  # Missing Percentage
tp = 'Parafac2'  # Define Solution Format of the Created Synthetic Tensor As 'CP
                                          decomposition'
[X1, Omega1, sol1] = create(problem, siz, r, miss, tp, share_mode_size=10)
self = parafac2.PARAFAC2(X1, r, printitn=100, maxiter=1000, tol=1e-7)
self.run()
```