实验 4：服务 QoS 预测

🔔 **实验内容：**

1. 服务 QoS 预测算法代码理解
   a. 参考 WS-DREAM 文档（https://github.com/icsme2020/WS-DREAM），查看 /home/WS-DREAM 目录下至少 5 种服务 QoS 预测算法代码，分析其原理。
2. 服务 QoS 预测算法执行
   a. 执行至少 5 种服务 QoS 预测算法并分析结果。
3. 撰写实验报告

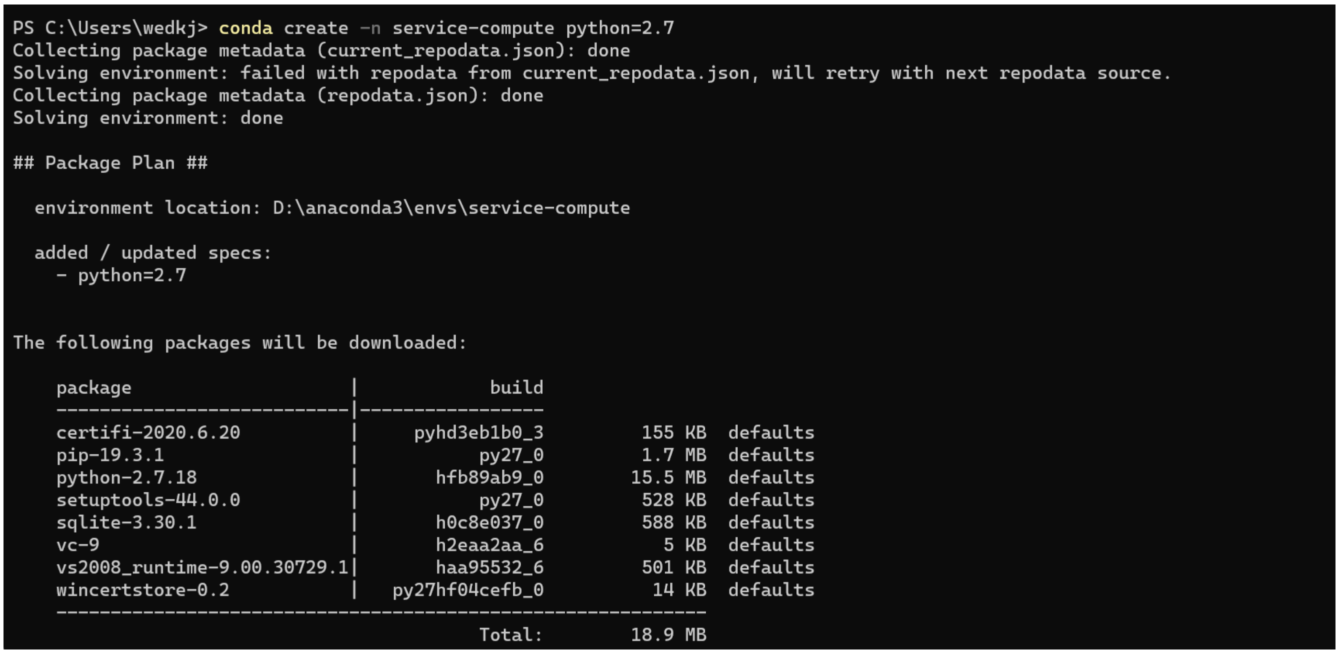
> 王靖皓 BY2221105
> 实验代码及文档：实验 4-github

1. 程序运行环境搭建

WS-DREAM 项目依赖的运行环境为：

- Python 2.7 (https://www.python.org)
- Cython 0.20.1 (http://cython.org)
- numpy 1.8.1 (http://www.scipy.org)
- scipy 0.13.3 (http://www.scipy.org)
- AMF (https://github.com/wsdream/AMF)
- PPCF (https://github.com/wsdream/PPCF)

1. 该项目使用 Python 2.7 版本，为了避免操作系统现在的 Python 版本对其干扰，使用 Anaconda 创建 python 版本为 2.7 的虚拟环境：



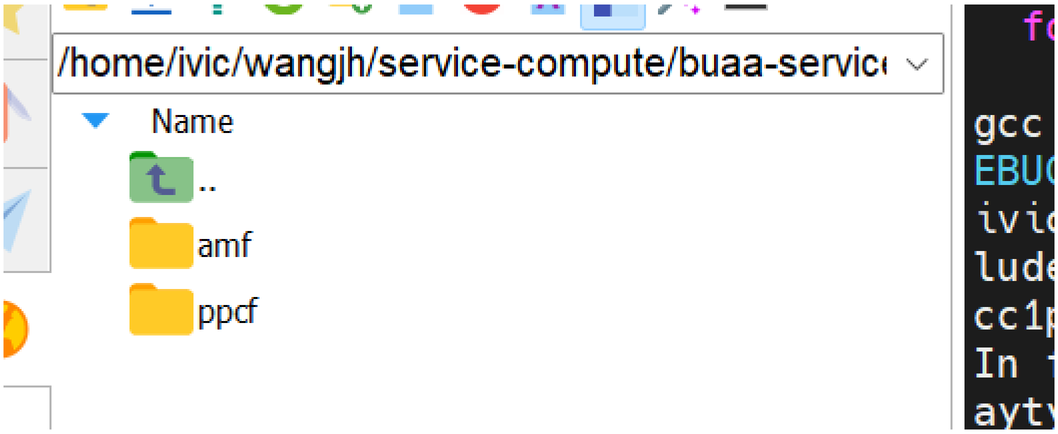
```
PS C:\Users\wedkj> conda create -n service-compute python=2.7
Collecting package metadata (current_repodata.json): done
Solving environment: failed with repodata from current_repodata.json, will retry with next repodata source.
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: D:\anaconda3\envs\service-compute

  added / updated specs:
    - python=2.7


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    certifi-2020.6.20          |       pyhd3eb1b0_3         155 KB  defaults
    pip-19.3.1                 |           py27_0         1.7 MB  defaults
    python-2.7.18              |       hfb89ab9_0        15.5 MB  defaults
    setuptools-44.0.0          |           py27_0         528 KB  defaults
    sqlite-3.30.1              |       h0c8e037_0         588 KB  defaults
    vc-9                       |       h2eaa2aa_6           5 KB  defaults
    vs2008_runtime-9.00.30729.1|       haa95532_6         501 KB  defaults
    wincertstore-0.2           |    py27hf04cefb_0          14 KB  defaults
    ------------------------------------------------------------
                                           Total:        18.9 MB
```

2. 环境创建好后，先安装 Python 的基础依赖

该项目依赖的 Python 包写在项目根目录下的 `requirements.txt` 文件中，

使用命令 `pip install -r requirements.txt` 完成安装

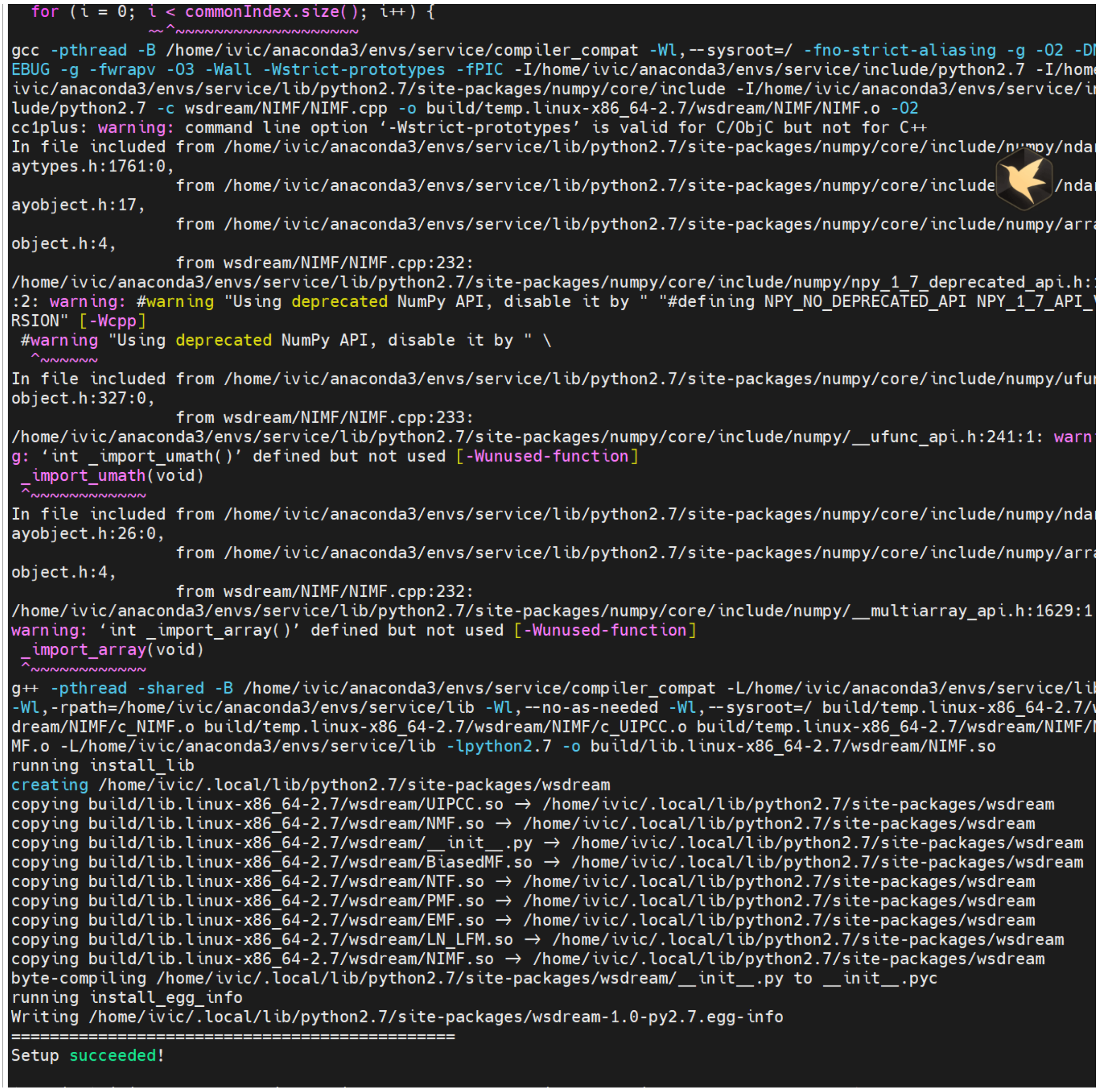
分别在 `src/` 目录下



的两个依赖分别构建。

3. 安装 package:

使用命令 `python setup.py install --user` 编译成功



```
   for (i = 0; i < commonIndex.size(); i++) {
                 ~~^~~~~~~~~~~~~~~~~~~~
gcc -pthread -B /home/ivic/anaconda3/envs/service/compiler_compat -Wl,--sysroot=/ -fno-strict-aliasing -g -O2 -DN
EBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/ivic/anaconda3/envs/service/include/python2.7 -I/home
ivic/anaconda3/envs/service/lib/python2.7/site-packages/numpy/core/include -I/home/ivic/anaconda3/envs/service/in
lude/python2.7 -c wsdream/NIMF/NIMF.cpp -o build/temp.linux-x86_64-2.7/wsdream/NIMF/NIMF.o -O2
cc1plus: warning: command line option '-Wstrict-prototypes' is valid for C/ObjC but not for C++
In file included from /home/ivic/anaconda3/envs/service/lib/python2.7/site-packages/numpy/core/include/numpy/nda
aytypes.h:1761:0,
                 from /home/ivic/anaconda3/envs/service/lib/python2.7/site-packages/numpy/core/include    /nda
ayobject.h:17,
                 from /home/ivic/anaconda3/envs/service/lib/python2.7/site-packages/numpy/core/include/numpy/arra
object.h:4,
                 from wsdream/NIMF/NIMF.cpp:232:
/home/ivic/anaconda3/envs/service/lib/python2.7/site-packages/numpy/core/include/numpy/npy_1_7_deprecated_api.h:
:2: warning: #warning "Using deprecated NumPy API, disable it by " "#defining NPY_NO_DEPRECATED_API NPY_1_7_API_V
RSION" [-Wcpp]
 #warning "Using deprecated NumPy API, disable it by " \
  ^~~~~~~
In file included from /home/ivic/anaconda3/envs/service/lib/python2.7/site-packages/numpy/core/include/numpy/ufun
object.h:327:0,
                 from wsdream/NIMF/NIMF.cpp:233:
/home/ivic/anaconda3/envs/service/lib/python2.7/site-packages/numpy/core/include/numpy/__ufunc_api.h:241:1: warn
g: 'int _import_umath()' defined but not used [-Wunused-function]
 _import_umath(void)
 ^~~~~~~~~~~~~
In file included from /home/ivic/anaconda3/envs/service/lib/python2.7/site-packages/numpy/core/include/numpy/nda
ayobject.h:26:0,
                 from /home/ivic/anaconda3/envs/service/lib/python2.7/site-packages/numpy/core/include/numpy/arra
object.h:4,
                 from wsdream/NIMF/NIMF.cpp:232:
/home/ivic/anaconda3/envs/service/lib/python2.7/site-packages/numpy/core/include/numpy/__multiarray_api.h:1629:1
warning: 'int _import_array()' defined but not used [-Wunused-function]
 _import_array(void)
 ^~~~~~~~~~~~~
g++ -pthread -shared -B /home/ivic/anaconda3/envs/service/compiler_compat -L/home/ivic/anaconda3/envs/service/li
-Wl,-rpath=/home/ivic/anaconda3/envs/service/lib -Wl,--no-as-needed -Wl,--sysroot=/ build/temp.linux-x86_64-2.7/
dream/NIMF/c_NIMF.o build/temp.linux-x86_64-2.7/wsdream/NIMF/c_UIPCC.o build/temp.linux-x86_64-2.7/wsdream/NIMF/N
MF.o -L/home/ivic/anaconda3/envs/service/lib -lpython2.7 -o build/lib.linux-x86_64-2.7/wsdream/NIMF.so
running install_lib
creating /home/ivic/.local/lib/python2.7/site-packages/wsdream
copying build/lib.linux-x86_64-2.7/wsdream/UIPCC.so -> /home/ivic/.local/lib/python2.7/site-packages/wsdream
copying build/lib.linux-x86_64-2.7/wsdream/NMF.so -> /home/ivic/.local/lib/python2.7/site-packages/wsdream
copying build/lib.linux-x86_64-2.7/wsdream/__init__.py -> /home/ivic/.local/lib/python2.7/site-packages/wsdream
copying build/lib.linux-x86_64-2.7/wsdream/BiasedMF.so -> /home/ivic/.local/lib/python2.7/site-packages/wsdream
copying build/lib.linux-x86_64-2.7/wsdream/NTF.so -> /home/ivic/.local/lib/python2.7/site-packages/wsdream
copying build/lib.linux-x86_64-2.7/wsdream/PMF.so -> /home/ivic/.local/lib/python2.7/site-packages/wsdream
copying build/lib.linux-x86_64-2.7/wsdream/EMF.so -> /home/ivic/.local/lib/python2.7/site-packages/wsdream
copying build/lib.linux-x86_64-2.7/wsdream/LN_LFM.so -> /home/ivic/.local/lib/python2.7/site-packages/wsdream
copying build/lib.linux-x86_64-2.7/wsdream/NIMF.so -> /home/ivic/.local/lib/python2.7/site-packages/wsdream
byte-compiling /home/ivic/.local/lib/python2.7/site-packages/wsdream/__init__.py to __init__.pyc
running install_egg_info
Writing /home/ivic/.local/lib/python2.7/site-packages/wsdream-1.0-py2.7.egg-info
==============================================
Setup succeeded!
```

至此，运行环境搭建完成

2. QoS 测试

在此运行 5 种服务 QoS 预测算法代码

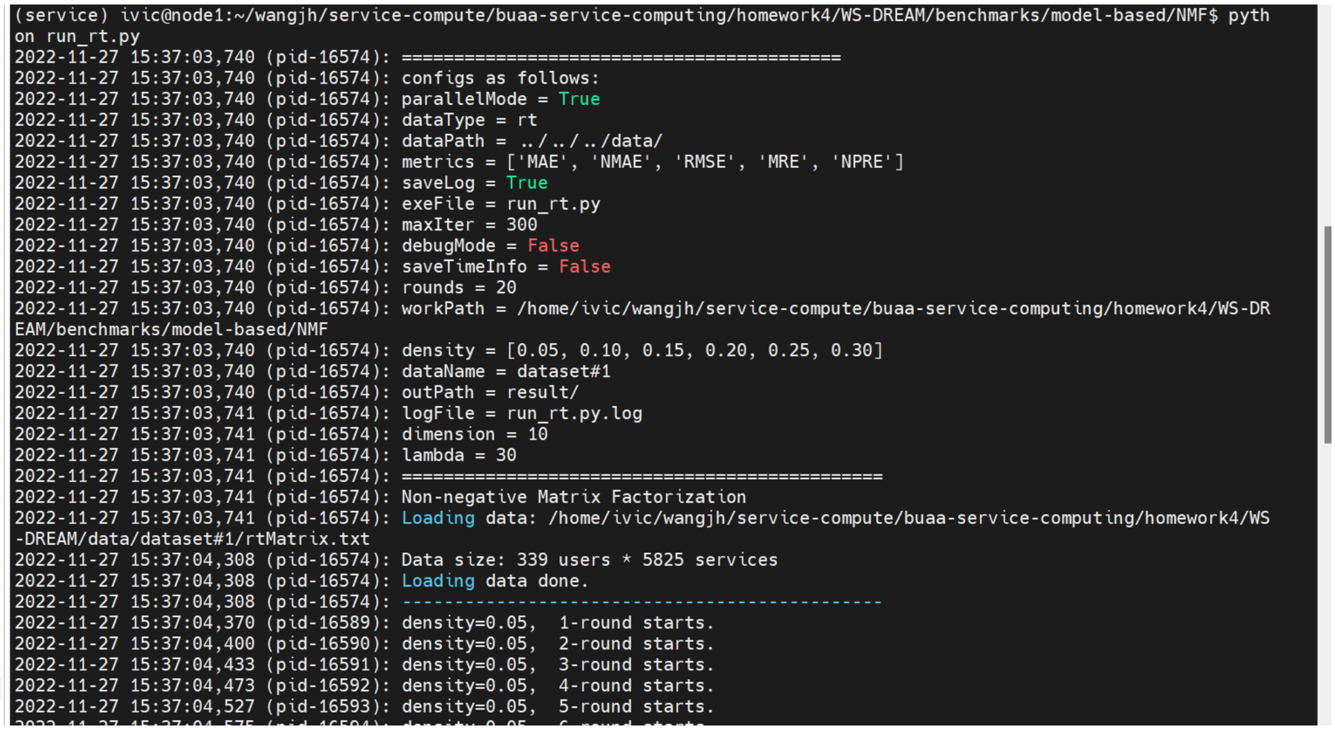
1. NTF 方法(非负矩阵分解)：

💡 **Non-negative Matrix Factorization**

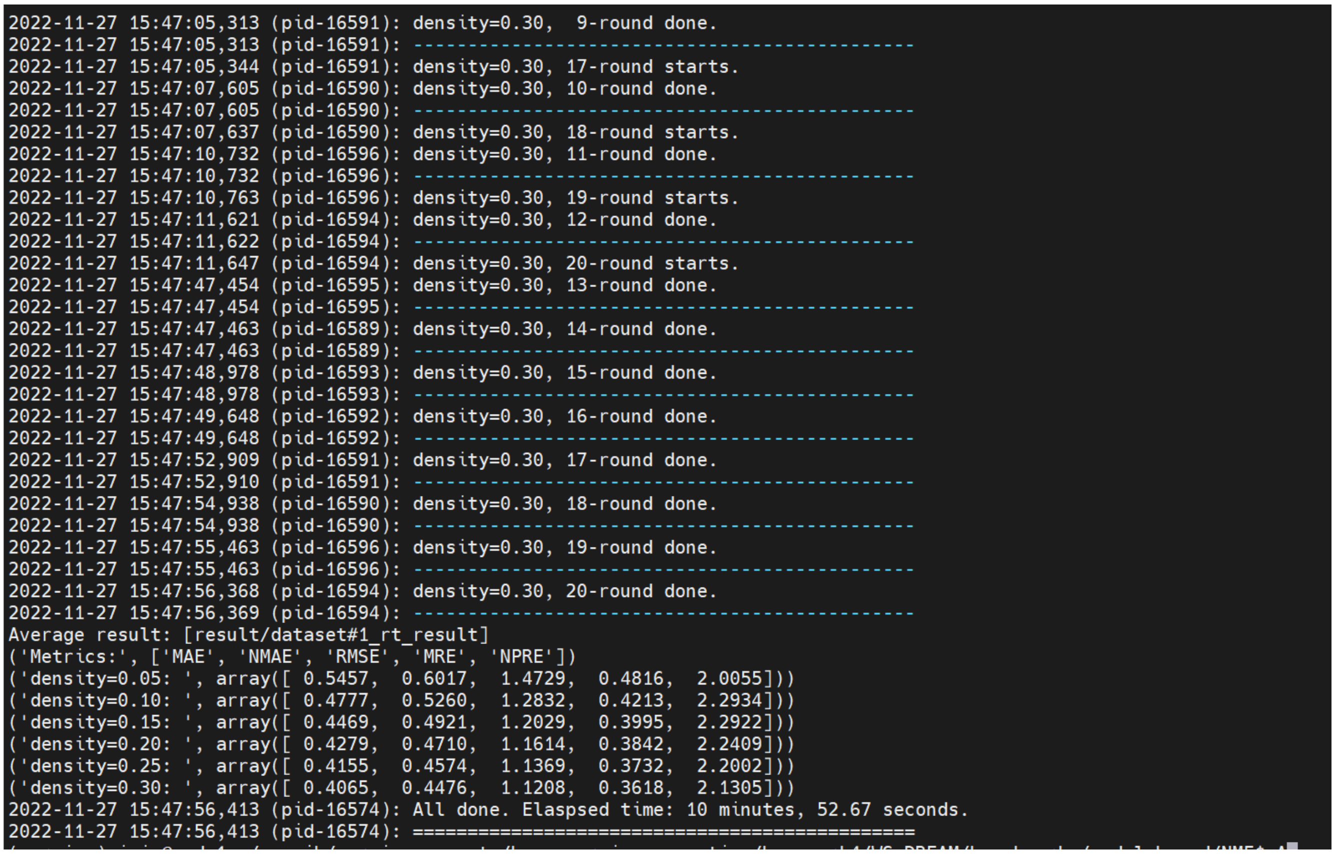
算法原文：Temporal QoS-Aware Web Service Recommendation viaNon-negative Tensor Factorization

NTF 方法为 Model-based 方法

运行结果



```
(service) ivic@node1:~/wangjh/service-compute/buaa-service-computing/homework4/WS-DREAM/benchmarks/model-based/NMF$ pyth
on run_rt.py
2022-11-27 15:37:03,740 (pid-16574): ========================================
2022-11-27 15:37:03,740 (pid-16574): configs as follows:
2022-11-27 15:37:03,740 (pid-16574): parallelMode = True
2022-11-27 15:37:03,740 (pid-16574): dataType = rt
2022-11-27 15:37:03,740 (pid-16574): dataPath = ../../../data/
2022-11-27 15:37:03,740 (pid-16574): metrics = ['MAE', 'NMAE', 'RMSE', 'MRE', 'NPRE']
2022-11-27 15:37:03,740 (pid-16574): saveLog = True
2022-11-27 15:37:03,740 (pid-16574): exeFile = run_rt.py
2022-11-27 15:37:03,740 (pid-16574): maxIter = 300
2022-11-27 15:37:03,740 (pid-16574): debugMode = False
2022-11-27 15:37:03,740 (pid-16574): saveTimeInfo = False
2022-11-27 15:37:03,740 (pid-16574): rounds = 20
2022-11-27 15:37:03,740 (pid-16574): workPath = /home/ivic/wangjh/service-compute/buaa-service-computing/homework4/WS-DR
EAM/benchmarks/model-based/NMF
2022-11-27 15:37:03,740 (pid-16574): density = [0.05, 0.10, 0.15, 0.20, 0.25, 0.30]
2022-11-27 15:37:03,740 (pid-16574): dataName = dataset#1
2022-11-27 15:37:03,740 (pid-16574): outPath = result/
2022-11-27 15:37:03,741 (pid-16574): logFile = run_rt.py.log
2022-11-27 15:37:03,741 (pid-16574): dimension = 10
2022-11-27 15:37:03,741 (pid-16574): lambda = 30
2022-11-27 15:37:03,741 (pid-16574): ==============================================
2022-11-27 15:37:03,741 (pid-16574): Non-negative Matrix Factorization
2022-11-27 15:37:03,741 (pid-16574): Loading data: /home/ivic/wangjh/service-compute/buaa-service-computing/homework4/WS
-DREAM/data/dataset#1/rtMatrix.txt
2022-11-27 15:37:04,308 (pid-16574): Data size: 339 users * 5825 services
2022-11-27 15:37:04,308 (pid-16574): Loading data done.
2022-11-27 15:37:04,308 (pid-16574): ----------------------------------------------
2022-11-27 15:37:04,370 (pid-16589): density=0.05,  1-round starts.
2022-11-27 15:37:04,400 (pid-16590): density=0.05,  2-round starts.
2022-11-27 15:37:04,433 (pid-16591): density=0.05,  3-round starts.
2022-11-27 15:37:04,473 (pid-16592): density=0.05,  4-round starts.
2022-11-27 15:37:04,527 (pid-16593): density=0.05,  5-round starts.
```



```
2022-11-27 15:47:05,313 (pid-16591): density=0.30,  9-round done.
2022-11-27 15:47:05,313 (pid-16591): ----------------------------------------------
2022-11-27 15:47:05,344 (pid-16591): density=0.30, 17-round starts.
2022-11-27 15:47:07,605 (pid-16590): density=0.30, 10-round done.
2022-11-27 15:47:07,605 (pid-16590): ----------------------------------------------
2022-11-27 15:47:07,637 (pid-16590): density=0.30, 18-round starts.
2022-11-27 15:47:10,732 (pid-16596): density=0.30, 11-round done.
2022-11-27 15:47:10,732 (pid-16596): ----------------------------------------------
2022-11-27 15:47:10,763 (pid-16596): density=0.30, 19-round starts.
2022-11-27 15:47:11,621 (pid-16594): density=0.30, 12-round done.
2022-11-27 15:47:11,622 (pid-16594): ----------------------------------------------
2022-11-27 15:47:11,647 (pid-16594): density=0.30, 20-round starts.
2022-11-27 15:47:47,454 (pid-16595): density=0.30, 13-round done.
2022-11-27 15:47:47,454 (pid-16595): ----------------------------------------------
2022-11-27 15:47:47,463 (pid-16589): density=0.30, 14-round done.
2022-11-27 15:47:47,463 (pid-16589): ----------------------------------------------
2022-11-27 15:47:48,978 (pid-16593): density=0.30, 15-round done.
2022-11-27 15:47:48,978 (pid-16593): ----------------------------------------------
2022-11-27 15:47:49,648 (pid-16592): density=0.30, 16-round done.
2022-11-27 15:47:49,648 (pid-16592): ----------------------------------------------
2022-11-27 15:47:52,909 (pid-16591): density=0.30, 17-round done.
2022-11-27 15:47:52,910 (pid-16591): ----------------------------------------------
2022-11-27 15:47:54,938 (pid-16590): density=0.30, 18-round done.
2022-11-27 15:47:54,938 (pid-16590): ----------------------------------------------
2022-11-27 15:47:55,463 (pid-16596): density=0.30, 19-round done.
2022-11-27 15:47:55,463 (pid-16596): ----------------------------------------------
2022-11-27 15:47:56,368 (pid-16594): density=0.30, 20-round done.
2022-11-27 15:47:56,369 (pid-16594): ----------------------------------------------
Average result: [result/dataset#1_rt_result]
('Metrics:', ['MAE', 'NMAE', 'RMSE', 'MRE', 'NPRE'])
('density=0.05: ', array([ 0.5457,  0.6017,  1.4729,  0.4816,  2.0055]))
('density=0.10: ', array([ 0.4777,  0.5260,  1.2832,  0.4213,  2.2934]))
('density=0.15: ', array([ 0.4469,  0.4921,  1.2029,  0.3995,  2.2922]))
('density=0.20: ', array([ 0.4279,  0.4710,  1.1614,  0.3842,  2.2409]))
('density=0.25: ', array([ 0.4155,  0.4574,  1.1369,  0.3732,  2.2002]))
('density=0.30: ', array([ 0.4065,  0.4476,  1.1208,  0.3618,  2.1305]))
2022-11-27 15:47:56,413 (pid-16574): All done. Elapsed time: 10 minutes, 52.67 seconds.
2022-11-27 15:47:56,413 (pid-16574): ==============================================
```

方法原理

该文章将两个二维用户服务矩阵扩展为一个以三维张量表示的更复杂的用户服务时间三元关系，并提出了一种基于矩阵分解的广义张量因子分解（TF）。通过考虑不同时间 QoS 值的差异，使用 用户 - 服务 - 时间 关系替换用户 - 服务矩阵，来进行 QoS 预测。在真实世界中，QoS 值永远为非负数，因此，文章提出了一种非负张量因子分解（NTF）方法，以在考虑服务调用时间的情况下预测 Web 服务 QoS 值，并提出了一种基于时间 QoS 的 Web 服务推荐框架。

通过考虑第三种动态上下文信息，提出了一种时间 QoS 感知 Web 服务推荐框架，以预测在各种时间上下文下丢失的 QoS 值。此外，我们将这个问题形式化为广义张量分解模型，并提出了一种能够处理用户服务时间模型的三元关系的非负张量分解（NTF）算法。基于我们在 Planet-Lab 上收集的真实 Web 服务 QoS 数据集进行了大量实验，该数据集由 32 个时间段内 5,817 个 Web 服务上的 343 个用户的服务调用响应时间和吞吐量值组成。

下图展示了如何将二维的矩阵，增加时间维度，构建 用户 - 服务 - 时间 矩阵：

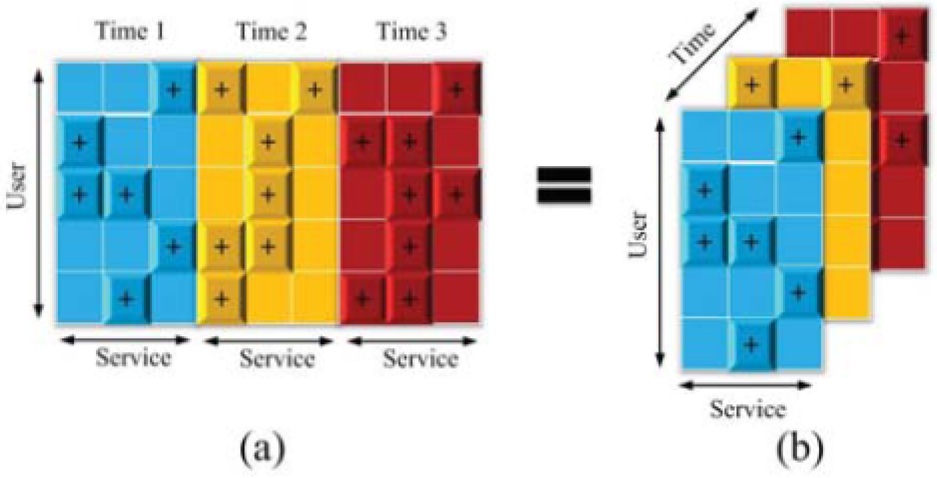
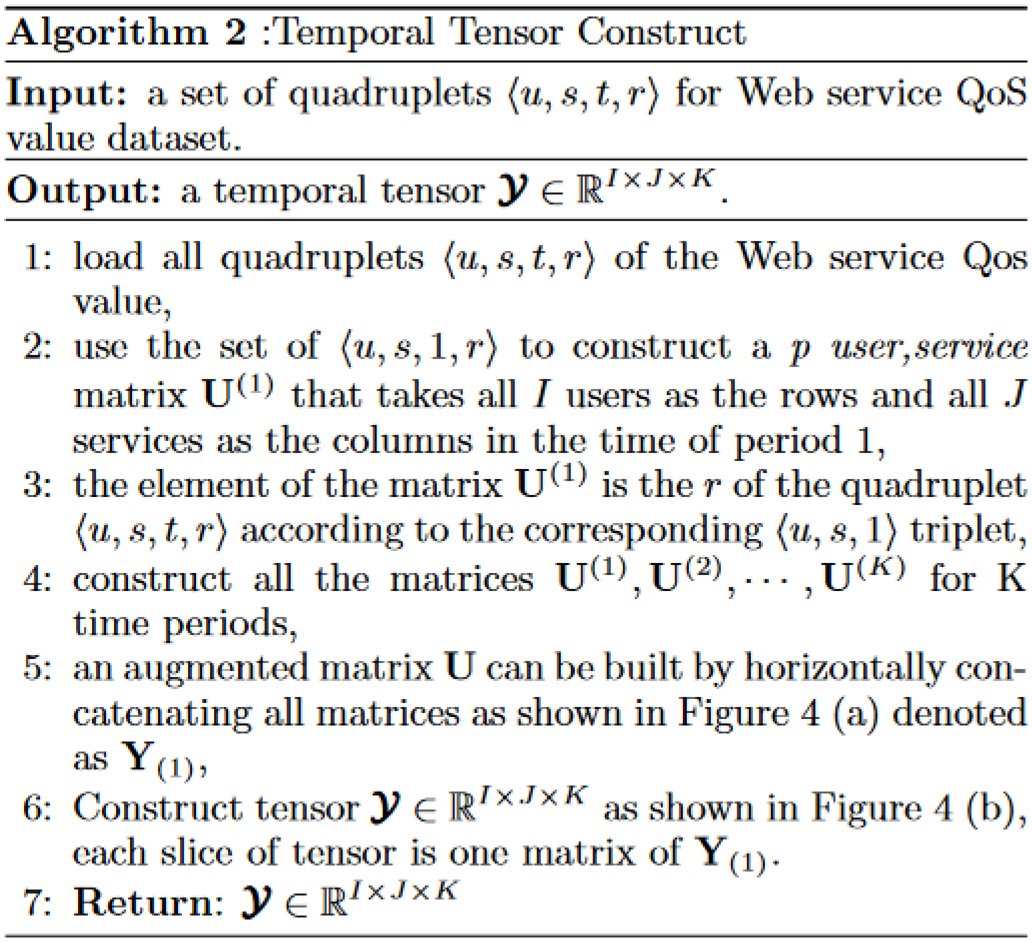


Figure 4: Slices of time-specific matrices with users and services are transformed into a temporal tensor

该三维矩阵的构造方法如下：



Algorithm 2 :Temporal Tensor Construct

Input: a set of quadruplets $\langle u, s, t, r\rangle$ for Web service QoS value dataset.

Output: a temporal tensor $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{I\times J\times K}$.

```
1: load all quadruplets $\langle u, s, t, r\rangle$ of the Web service Qos value,
2: use the set of $\langle u, s, 1, r\rangle$ to construct a $p$ $user, service$ matrix $\mathbf{U}^{(1)}$ that takes all $I$ users as the rows and all $J$ services as the columns in the time of period 1,
3: the element of the matrix $\mathbf{U}^{(1)}$ is the $r$ of the quadruplet $\langle u, s, t, r\rangle$ according to the corresponding $\langle u, s, 1\rangle$ triplet,
4: construct all the matrices $\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \cdots, \mathbf{U}^{(K)}$ for K time periods,
5: an augmented matrix $\mathbf{U}$ can be built by horizontally concatenating all matrices as shown in Figure 4 (a) denoted as $\mathbf{Y}_{(1)}$,
6: Construct tensor $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{I\times J\times K}$ as shown in Figure 4 (b), each slice of tensor is one matrix of $\mathbf{Y}_{(1)}$.
7: **Return**: $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{I\times J\times K}$
```

NTF 方法流程如下：

Algorithm 3 : Non-negative CP decomposition algorithm

Input: the tensor $\boldsymbol{\mathcal{Y}} \in \mathbb{R}_{+}^{I\times J\times K}$, the rank R of tensor $\boldsymbol{\mathcal{Y}}$.

Output: three non-negative factor matrices $\hat{\mathbf{U}}, \hat{\mathbf{S}}, \hat{\mathbf{T}}$.

```
 1: Procedure $[\hat{\mathbf{U}}, \hat{\mathbf{S}}, \hat{\mathbf{T}}] = \mathrm{NNCP}(\boldsymbol{\mathcal{Y}}, R)$
 2: Initialize: $\mathbf{U} \in \mathbb{R}_{+}^{I\times R}, \mathbf{S} \in \mathbb{R}_{+}^{J\times R}$, and $\mathbf{T} \in \mathbb{R}_{+}^{K\times R}$ by
    small non-negative value.
 3: Repeat
 4: for $l = 1, \cdots, I$ do
 5:     $\hat{\mathbf{U}} \leftarrow Eq.(15)$
 6: end for
 7: for $l = 1, \cdots, J$ do
 8:     $\hat{\mathbf{S}} \leftarrow Eq.(16)$
 9: end for
10: for $l = 1, \cdots, K$ do
11:     $\hat{\mathbf{T}} \leftarrow Eq.(17)$
12: end for
13: Until convergence or maximum iterations exhausted.
14: Return: $\hat{\mathbf{U}}, \hat{\mathbf{S}}, \hat{\mathbf{T}}$
15: EndProcedure
```

其中:

$$\mathbf{u}_l^{(i+1)} \leftarrow \frac{\mathbf{u}_l^{(i)} \sum_{j,k} \boldsymbol{\mathcal{Y}}_{ijk} \mathbf{s}_l^{(j)} \mathbf{t}_l^{(k)}}{\sum_{r=1}^{R_{\boldsymbol{\mathcal{Y}}}} \mathbf{u}_r^{(i)} (\mathbf{s}_r \cdot \mathbf{s}_l)(\mathbf{t}_r \cdot \mathbf{t}_l)}, \tag{15}$$

$$\mathbf{s}_l^{(j+1)} \leftarrow \frac{\mathbf{s}_l^{(j)} \sum_{i,k} \boldsymbol{\mathcal{Y}}_{ijk} \mathbf{u}_l^{(i)} \mathbf{t}_l^{(k)}}{\sum_{r=1}^{R_{\boldsymbol{\mathcal{Y}}}} \mathbf{s}_r^{(j)} (\mathbf{u}_r \cdot \mathbf{u}_l)(\mathbf{t}_r \cdot \mathbf{t}_l)}; \tag{16}$$

$$\mathbf{t}_l^{(k+1)} \leftarrow \frac{\mathbf{t}_l^{(k)} \sum_{i,j} \boldsymbol{\mathcal{Y}}_{ikl} \mathbf{u}_l^{(i)} \mathbf{s}_l^{(j)}}{\sum_{r=1}^{R_{\boldsymbol{\mathcal{Y}}}} \mathbf{t}_r^{(k)} (\mathbf{u}_r \cdot \mathbf{u}_l)(\mathbf{s}_r \cdot \mathbf{s}_l)}, \tag{17}$$

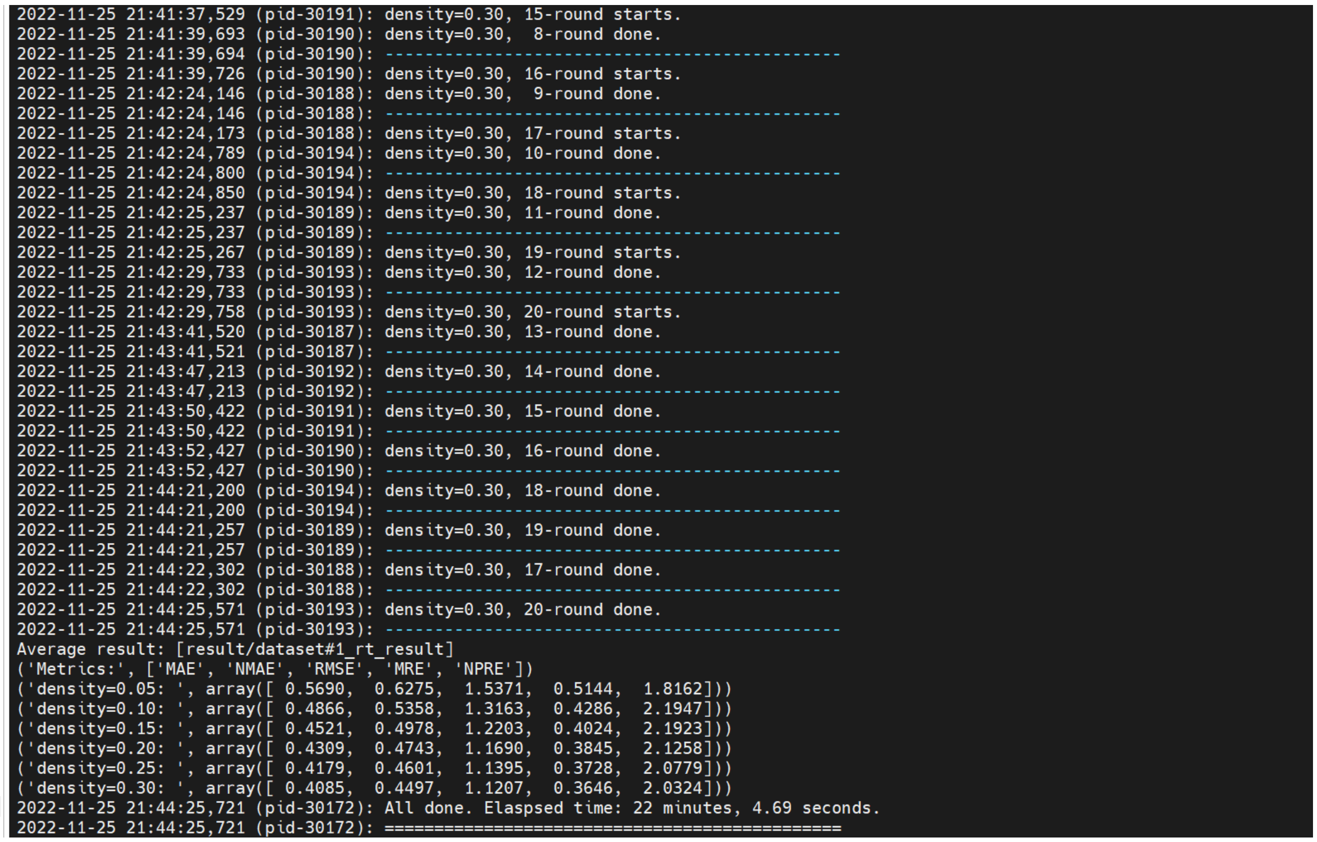
2. PMF 方法（概率矩阵分解）：

👋 PMF 为 Model-based 方法。

文章：Personalized Reliability Prediction of Web Services

该方法会根据可用的 Web 服务故障数据建立一个因子模型，并使用该因子模型进行进一步的可靠性预测。

运行结果：



```
2022-11-25 21:41:37,529 (pid-30191): density=0.30, 15-round starts.
2022-11-25 21:41:39,693 (pid-30190): density=0.30,  8-round done.
2022-11-25 21:41:39,694 (pid-30190): ----------------------------------------------
2022-11-25 21:41:39,726 (pid-30190): density=0.30, 16-round starts.
2022-11-25 21:42:24,146 (pid-30188): density=0.30,  9-round done.
2022-11-25 21:42:24,146 (pid-30188): ----------------------------------------------
2022-11-25 21:42:24,173 (pid-30188): density=0.30, 17-round starts.
2022-11-25 21:42:24,789 (pid-30194): density=0.30, 10-round done.
2022-11-25 21:42:24,800 (pid-30194): ----------------------------------------------
2022-11-25 21:42:24,850 (pid-30194): density=0.30, 18-round starts.
2022-11-25 21:42:25,237 (pid-30189): density=0.30, 11-round done.
2022-11-25 21:42:25,237 (pid-30189): ----------------------------------------------
2022-11-25 21:42:25,267 (pid-30189): density=0.30, 19-round starts.
2022-11-25 21:42:29,733 (pid-30193): density=0.30, 12-round done.
2022-11-25 21:42:29,733 (pid-30193): ----------------------------------------------
2022-11-25 21:42:29,758 (pid-30193): density=0.30, 20-round starts.
2022-11-25 21:43:41,520 (pid-30187): density=0.30, 13-round done.
2022-11-25 21:43:41,521 (pid-30187): ----------------------------------------------
2022-11-25 21:43:47,213 (pid-30192): density=0.30, 14-round done.
2022-11-25 21:43:47,213 (pid-30192): ----------------------------------------------
2022-11-25 21:43:50,422 (pid-30191): density=0.30, 15-round done.
2022-11-25 21:43:50,422 (pid-30191): ----------------------------------------------
2022-11-25 21:43:52,427 (pid-30190): density=0.30, 16-round done.
2022-11-25 21:43:52,427 (pid-30190): ----------------------------------------------
2022-11-25 21:44:21,200 (pid-30194): density=0.30, 18-round done.
2022-11-25 21:44:21,200 (pid-30194): ----------------------------------------------
2022-11-25 21:44:21,257 (pid-30189): density=0.30, 19-round done.
2022-11-25 21:44:21,257 (pid-30189): ----------------------------------------------
2022-11-25 21:44:22,302 (pid-30188): density=0.30, 17-round done.
2022-11-25 21:44:22,302 (pid-30188): ----------------------------------------------
2022-11-25 21:44:25,571 (pid-30193): density=0.30, 20-round done.
2022-11-25 21:44:25,571 (pid-30193): ----------------------------------------------
Average result: [result/dataset#1_rt_result]
('Metrics:', ['MAE', 'NMAE', 'RMSE', 'MRE', 'NPRE'])
('density=0.05: ', array([ 0.5690,  0.6275,  1.5371,  0.5144,  1.8162]))
('density=0.10: ', array([ 0.4866,  0.5358,  1.3163,  0.4286,  2.1947]))
('density=0.15: ', array([ 0.4521,  0.4978,  1.2203,  0.4024,  2.1923]))
('density=0.20: ', array([ 0.4309,  0.4743,  1.1690,  0.3845,  2.1258]))
('density=0.25: ', array([ 0.4179,  0.4601,  1.1395,  0.3728,  2.0779]))
('density=0.30: ', array([ 0.4085,  0.4497,  1.1207,  0.3646,  2.0324]))
2022-11-25 21:44:25,721 (pid-30172): All done. Elaspsed time: 22 minutes, 4.69 seconds.
2022-11-25 21:44:25,721 (pid-30172): ==============================================
```

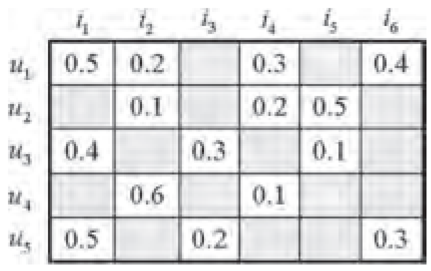
方法原理

1. 对 user-item 矩阵 P ，进行分解，分解为两个矩阵 W 和 H

$$P \approx WH$$

W 中的每一行是用户的特定系数，H 中的每一列是 Web 服务的因子向量

例如对于矩阵 P：



	i_1	i_2	i_3	i_4	i_5	i_6
u_1	0.5	0.2		0.3		0.4
u_2		0.1		0.2	0.5	
u_3	0.4		0.3		0.1	
u_4		0.6		0.1		
u_5	0.5		0.2			0.3

Fig. 1. A 5 × 6 user-item matrix.

可以将其分解为:

$$W = \begin{bmatrix} 0.135 & -0.747 & -0.430 & 0.245 \\ 0.281 & -1.049 & -0.527 & 0.489 \\ -0.029 & -0.622 & -0.224 & -1.135 \\ -0.434 & -0.404 & 0.637 & 0.921 \\ 0.209 & -0.423 & -0.995 & -0.303 \end{bmatrix},$$

$$H = \begin{bmatrix} -0.028 & -0.457 & -0.234 & 0.265 & 0.087 & -0.041 \\ 0.142 & 1.020 & 0.407 & 0.935 & 0.481 & 0.248 \\ -0.117 & 0.894 & 0.813 & -0.280 & 0.156 & 0.574 \\ 0.260 & -0.028 & 0.346 & -1.030 & 1.116 & 0.155 \end{bmatrix},$$

通过矩阵 W 和 H 的乘积，计算 P 中空缺的部分，完成 Qos 预测。

$$P = \begin{bmatrix} 0.5 & 0.2 & & 0.3 & & 0.4 \\ & 0.1 & & 0.2 & 0.5 & \\ 0.4 & & 0.3 & & 0.1 & \\ & 0.6 & & 0.1 & & \\ 0.5 & & 0.2 & & & 0.3 \end{bmatrix} \approx \begin{bmatrix} 0.501 & 0.229 & 0.354 & 0.311 & 0.465 & 0.401 \\ 0.508 & 0.157 & 0.320 & 0.221 & 0.496 & 0.378 \\ 0.412 & 0.312 & 0.306 & 0.655 & 0.168 & 0.388 \\ 0.530 & 0.582 & 0.684 & 0.165 & 0.710 & 0.605 \\ 0.493 & 0.197 & 0.243 & 0.562 & 0.337 & 0.325 \end{bmatrix} = WH.$$

矩阵 W 和 H 通常不唯一，通过使 WH 和 P 之间距离最小化来确定，文章中使用平方误差的方法，计算 P 和 WH 的差异:

$$\sum_{i=1}^{m}\sum_{j=1}^{n} I_{ij}^{P}(p_{ij} - W_i H_j)^2, \tag{18}$$

但为了避免过拟合，需要增加惩罚函数，最终可以将优化问题归结为如下目标函数

$$\min_{W,H} \mathcal{L}(W,H) = \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{n} I_{ij}^{P}(p_{ij} - W_i H_j)^2 + \frac{\gamma}{2}\|W\|_F^2 + \frac{\gamma}{2}\|H\|_F^2, \tag{19}$$

2. 使用梯度下降的方法，找到目标函数的局部最小值

算法流程为:

Algorithm 1: Gradient Descent Algorithm

```
Input: User-item matrix: $P$; number of factors: $l$
Output: User-specified coefficient matrix: $W$; Factor matrix: $H$
Initialize $W$ and $H$ with small random numbers;
repeat
    for each factor $l$ do
        for each available $p_{ij}$ in $P$ do
            $w_{il}^{t+1} = w_{il}^{t} - \alpha \frac{\partial \mathcal{L}}{\partial w_{il}^{t}}$;
            $h_{lj}^{t+1} = h_{lj}^{t} - \alpha \frac{\partial \mathcal{L}}{\partial h_{lj}^{t}}$;
        end
    end
until Converge ;
```

其中，梯度可以通过下式得到：

$$\frac{\partial \mathcal{L}}{\partial w_{il}} = \gamma w_{il} - \sum_{j=1}^{n} I_{ij}^{P}(p_{ij} - W_i H_j) h_{lj}, \tag{20}$$

$$\frac{\partial \mathcal{L}}{\partial h_{lj}} = \gamma h_{lj} - \sum_{i=1}^{m} I_{ij}^{P}(p_{ij} - W_i H_j) w_{il}, \tag{21}$$

首先。用小的随机数初始化矩阵 W 和 H;

之后，通过梯度下降的方法，迭代更新矩阵 W 和 H，参数 α 为学习率，控制迭代的速度。

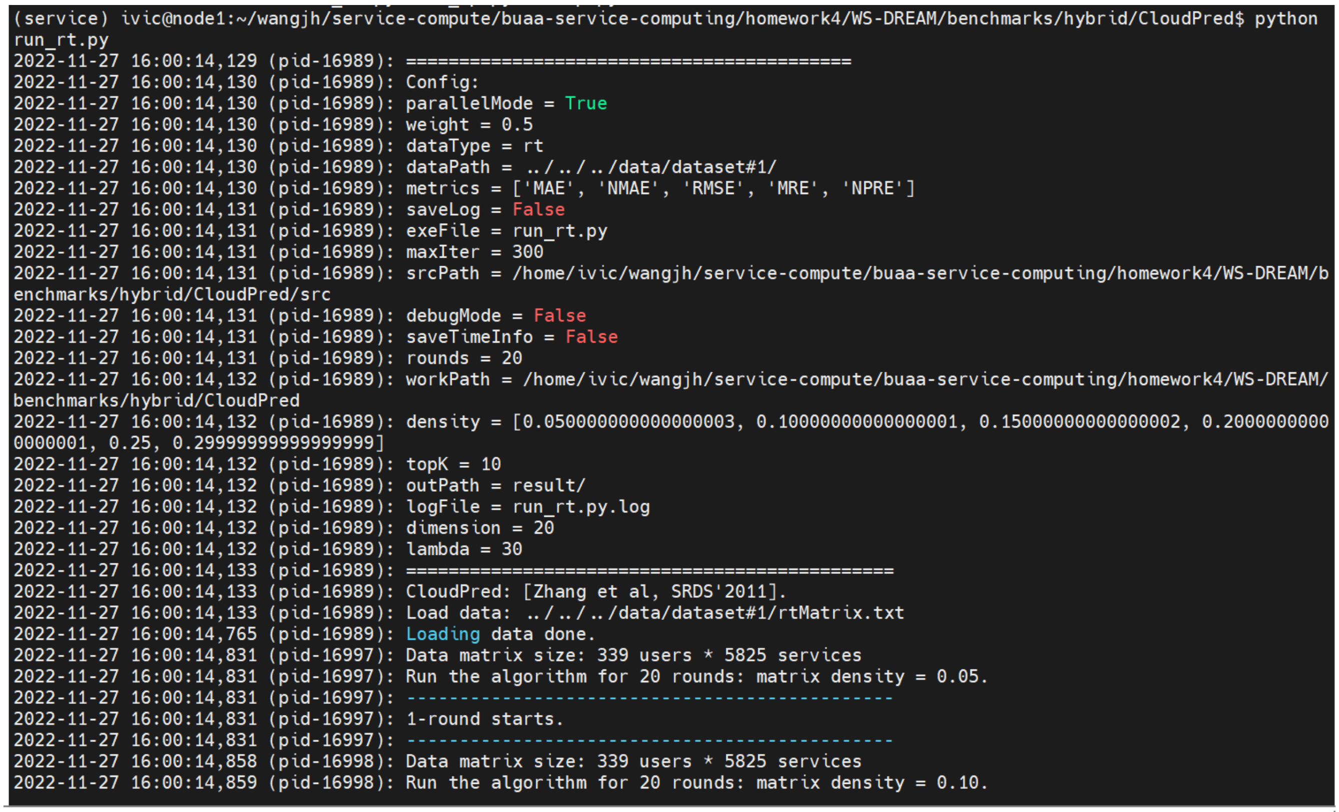
迭代停止时，计算出目标函数取得极小值处的 W 和 H，进行 QoS 预测

3. CloudPred 方法(基于邻域的方法)

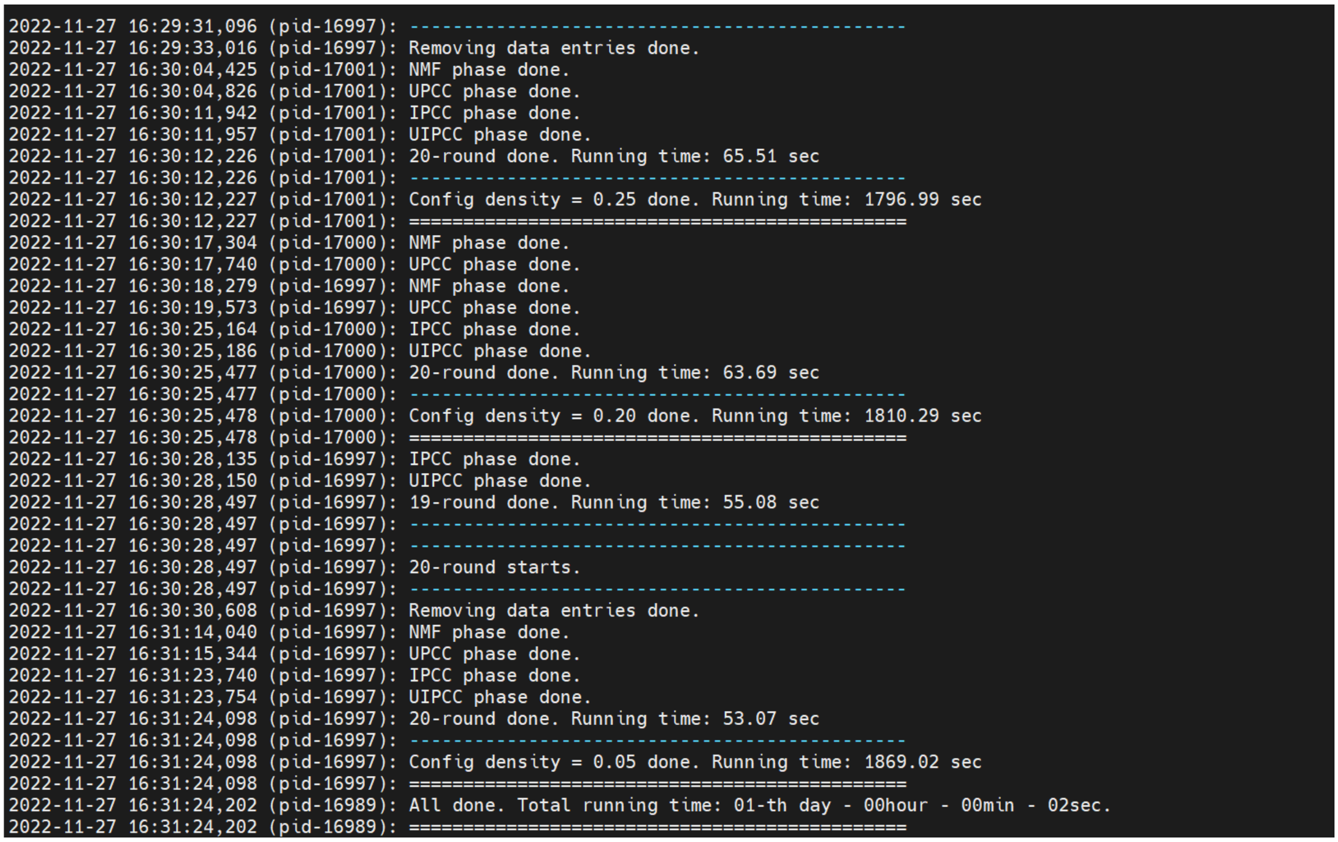
CloudPred 方法

文章链接：Exploring Latent Features for Memory-Based QoS Prediction in Cloud Computing

运行结果



```
(service) ivic@node1:~/wangjh/service-compute/buaa-service-computing/homework4/WS-DREAM/benchmarks/hybrid/CloudPred$ python
run_rt.py
2022-11-27 16:00:14,129 (pid-16989): ==========================================
2022-11-27 16:00:14,130 (pid-16989): Config:
2022-11-27 16:00:14,130 (pid-16989): parallelMode = True
2022-11-27 16:00:14,130 (pid-16989): weight = 0.5
2022-11-27 16:00:14,130 (pid-16989): dataType = rt
2022-11-27 16:00:14,130 (pid-16989): dataPath = ../../../data/dataset#1/
2022-11-27 16:00:14,130 (pid-16989): metrics = ['MAE', 'NMAE', 'RMSE', 'MRE', 'NPRE']
2022-11-27 16:00:14,131 (pid-16989): saveLog = False
2022-11-27 16:00:14,131 (pid-16989): exeFile = run_rt.py
2022-11-27 16:00:14,131 (pid-16989): maxIter = 300
2022-11-27 16:00:14,131 (pid-16989): srcPath = /home/ivic/wangjh/service-compute/buaa-service-computing/homework4/WS-DREAM/b
enchmarks/hybrid/CloudPred/src
2022-11-27 16:00:14,131 (pid-16989): debugMode = False
2022-11-27 16:00:14,131 (pid-16989): saveTimeInfo = False
2022-11-27 16:00:14,131 (pid-16989): rounds = 20
2022-11-27 16:00:14,132 (pid-16989): workPath = /home/ivic/wangjh/service-compute/buaa-service-computing/homework4/WS-DREAM/
benchmarks/hybrid/CloudPred
2022-11-27 16:00:14,132 (pid-16989): density = [0.050000000000000003, 0.10000000000000001, 0.15000000000000002, 0.2000000000
0000001, 0.25, 0.29999999999999999]
2022-11-27 16:00:14,132 (pid-16989): topK = 10
2022-11-27 16:00:14,132 (pid-16989): outPath = result/
2022-11-27 16:00:14,132 (pid-16989): logFile = run_rt.py.log
2022-11-27 16:00:14,132 (pid-16989): dimension = 20
2022-11-27 16:00:14,132 (pid-16989): lambda = 30
2022-11-27 16:00:14,133 (pid-16989): ==============================================
2022-11-27 16:00:14,133 (pid-16989): CloudPred: [Zhang et al, SRDS'2011].
2022-11-27 16:00:14,133 (pid-16989): Load data: ../../../data/dataset#1/rtMatrix.txt
2022-11-27 16:00:14,765 (pid-16989): Loading data done.
2022-11-27 16:00:14,831 (pid-16997): Data matrix size: 339 users * 5825 services
2022-11-27 16:00:14,831 (pid-16997): Run the algorithm for 20 rounds: matrix density = 0.05.
2022-11-27 16:00:14,831 (pid-16997): ----------------------------------------------
2022-11-27 16:00:14,831 (pid-16997): 1-round starts.
2022-11-27 16:00:14,831 (pid-16997): ----------------------------------------------
2022-11-27 16:00:14,858 (pid-16998): Data matrix size: 339 users * 5825 services
2022-11-27 16:00:14,859 (pid-16998): Run the algorithm for 20 rounds: matrix density = 0.10.
```



```
2022-11-27 16:29:31,096 (pid-16997): ----------------------------------------------
2022-11-27 16:29:33,016 (pid-16997): Removing data entries done.
2022-11-27 16:30:04,425 (pid-17001): NMF phase done.
2022-11-27 16:30:04,826 (pid-17001): UPCC phase done.
2022-11-27 16:30:11,942 (pid-17001): IPCC phase done.
2022-11-27 16:30:11,957 (pid-17001): UIPCC phase done.
2022-11-27 16:30:12,226 (pid-17001): 20-round done. Running time: 65.51 sec
2022-11-27 16:30:12,226 (pid-17001): ----------------------------------------------
2022-11-27 16:30:12,227 (pid-17001): Config density = 0.25 done. Running time: 1796.99 sec
2022-11-27 16:30:12,227 (pid-17001): ==============================================
2022-11-27 16:30:17,304 (pid-17000): NMF phase done.
2022-11-27 16:30:17,740 (pid-17000): UPCC phase done.
2022-11-27 16:30:18,279 (pid-16997): NMF phase done.
2022-11-27 16:30:19,573 (pid-16997): UPCC phase done.
2022-11-27 16:30:25,164 (pid-17000): IPCC phase done.
2022-11-27 16:30:25,186 (pid-17000): UIPCC phase done.
2022-11-27 16:30:25,477 (pid-17000): 20-round done. Running time: 63.69 sec
2022-11-27 16:30:25,477 (pid-17000): ----------------------------------------------
2022-11-27 16:30:25,478 (pid-17000): Config density = 0.20 done. Running time: 1810.29 sec
2022-11-27 16:30:25,478 (pid-17000): ==============================================
2022-11-27 16:30:28,135 (pid-16997): IPCC phase done.
2022-11-27 16:30:28,150 (pid-16997): UIPCC phase done.
2022-11-27 16:30:28,497 (pid-16997): 19-round done. Running time: 55.08 sec
2022-11-27 16:30:28,497 (pid-16997): ----------------------------------------------
2022-11-27 16:30:28,497 (pid-16997): ----------------------------------------------
2022-11-27 16:30:28,497 (pid-16997): 20-round starts.
2022-11-27 16:30:28,497 (pid-16997): ----------------------------------------------
2022-11-27 16:30:30,608 (pid-16997): Removing data entries done.
2022-11-27 16:31:14,040 (pid-16997): NMF phase done.
2022-11-27 16:31:15,344 (pid-16997): UPCC phase done.
2022-11-27 16:31:23,740 (pid-16997): IPCC phase done.
2022-11-27 16:31:23,754 (pid-16997): UIPCC phase done.
2022-11-27 16:31:24,098 (pid-16997): 20-round done. Running time: 53.07 sec
2022-11-27 16:31:24,098 (pid-16997): ----------------------------------------------
2022-11-27 16:31:24,098 (pid-16997): Config density = 0.05 done. Running time: 1869.02 sec
2022-11-27 16:31:24,098 (pid-16997): ==============================================
2022-11-27 16:31:24,202 (pid-16989): All done. Total running time: 01-th day - 00hour - 00min - 02sec.
2022-11-27 16:31:24,202 (pid-16989): ==============================================
```

方法原理

作者提出了一种基于邻域的方法，称为 CloudPred，用于云组件的协作和个性化质量预测。CloudPred 通过对用户和组件进行特征建模得到增强。

CloudPred 是基于邻域的方法，CloudPred 通过非负矩阵分解（NMF）了解用户的特征，并探索类似用户的 QoS 体验，以实现较高的 QoS 值预测准确性。

该方法将 QoS 矩阵分解为两个低秩矩阵 V 和 H，V 中的每一列代表用户的 l 维特征向量，H 中的每一列代表组件的特征向量。使用近似矩阵 $W \approx V^T H$ 来拟合

矩阵 V 和 H 是未知的，通过矩阵 W 中获得的 QoS 值学习，通过近似矩阵和原矩阵的距离，定义成本函数

$$F(W, \tilde{W}) = \|W - \tilde{W}\|_F^2 = \sum_{ij} (w_{ij} - \tilde{w}_{ij})^2, \qquad (2)$$

目标函数为：

$$\begin{aligned} \min_{V,H} \quad & f(V,H) = \sum_{(i,j)\in\Lambda} [\tilde{w}_{ij} - w_{ij} \log \tilde{w}_{ij}], \\ s.t. \quad & \tilde{w}_{i,j} = \sum_{k=1}^{l} v_{ki} h_{kj}, \\ & V \geq 0, \\ & H \geq 0. \end{aligned} \qquad (3)$$

为了使目标函数最小化，使用梯度下降的方法，计算局部最小值

```
Algorithm 1: Latent Features Learning Algorithm
Input: $W$, $l$
Output: $V$, $H$
Initialize $V \in \mathbb{R}^{l\times m}$ and $H \in \mathbb{R}^{l\times n}$ with small random numbers;
repeat
    for all $(i,j) \in \Lambda$ do
        $\tilde{w}_{ij} = \sum_k v_{ki} h_{kj}$;
    end
    for all $(i,j) \in \Lambda$ do
        $v_{ij} \leftarrow v_{ij} \sum_k \frac{w_{ik}}{\tilde{w}_{ik}} h_{jk}$;
        $h_{ij} \leftarrow h_{ij} \sum_k \frac{w_{ik}}{\tilde{w}_{ik}} v_{jk}$;
        $v_{ij} = \frac{v_{ij}}{\sum_k v_{kj}}$;
        $h_{ij} = \frac{h_{ij}}{\sum_k h_{kj}}$;
    end
    for all $(i,j) \in \Lambda$ do
        $\tilde{w}_{ij} = \sum_k v_{ki} h_{kj}$;
    end
until Converge ;
```

相似度计算

得到用户和组件特征矩阵 V 和 H 后，使用 Person 相关系数，计算不同用户和组件的邻域相似度，

$$S(u_i,u_j) = \frac{\sum_{k=1}^{l}(v_{ik}-\overline{v}_i)(v_{jk}-\overline{v}_j)}{\sqrt{\sum_{k=1}^{l}(v_{ik}-\overline{v}_i)^2}\sqrt{\sum_{k=1}^{l}(v_{jk}-\overline{v}_j)^2}}, \quad (8)$$

$$S(c_i,c_j) = \frac{\sum_{k=1}^{l}(h_{ik}-\overline{h}_i)(h_{jk}-\overline{h}_j)}{\sqrt{\sum_{k=1}^{l}(h_{ik}-\overline{h}_i)^2}\sqrt{\sum_{k=1}^{l}(h_{jk}-\overline{h}_j)^2}}, \quad (9)$$

QoS 预测

通过对相似度进行排序来识别当前用户的相似邻居。较少相似或不相似用户的 QoS 可能会大大降低预测准确性。 作者从相似的邻居集中排除 PCC 值为负的那些用户，而仅使用 Top-K 最大 PCC 值的的 QoS 来预测当前用户的 QoS 值。

$$w_{ij} = \overline{w}_i + \sum_{k \in \Psi_i} \frac{S(u_i, u_k)}{\sum_{a \in \Psi_i} S(u_i, u_a)} (w_{kj} - \overline{w}_k), \qquad (12)$$

$$w_{ij} = \overline{w}_j + \sum_{k \in \Phi_j} \frac{S(i_j, i_k)}{\sum_{a \in \Phi_j} S(i_j, i_a)} (w_{ik} - \overline{w}_k), \qquad (13)$$

$$w^*_{ij} = \lambda \times w^u_{ij} + (1 - \lambda) \times w^c_{ij}, \qquad (14)$$

CloudPred 预测算法，混合了基于用户的预测方法和基于组件的方法，其算法如下：

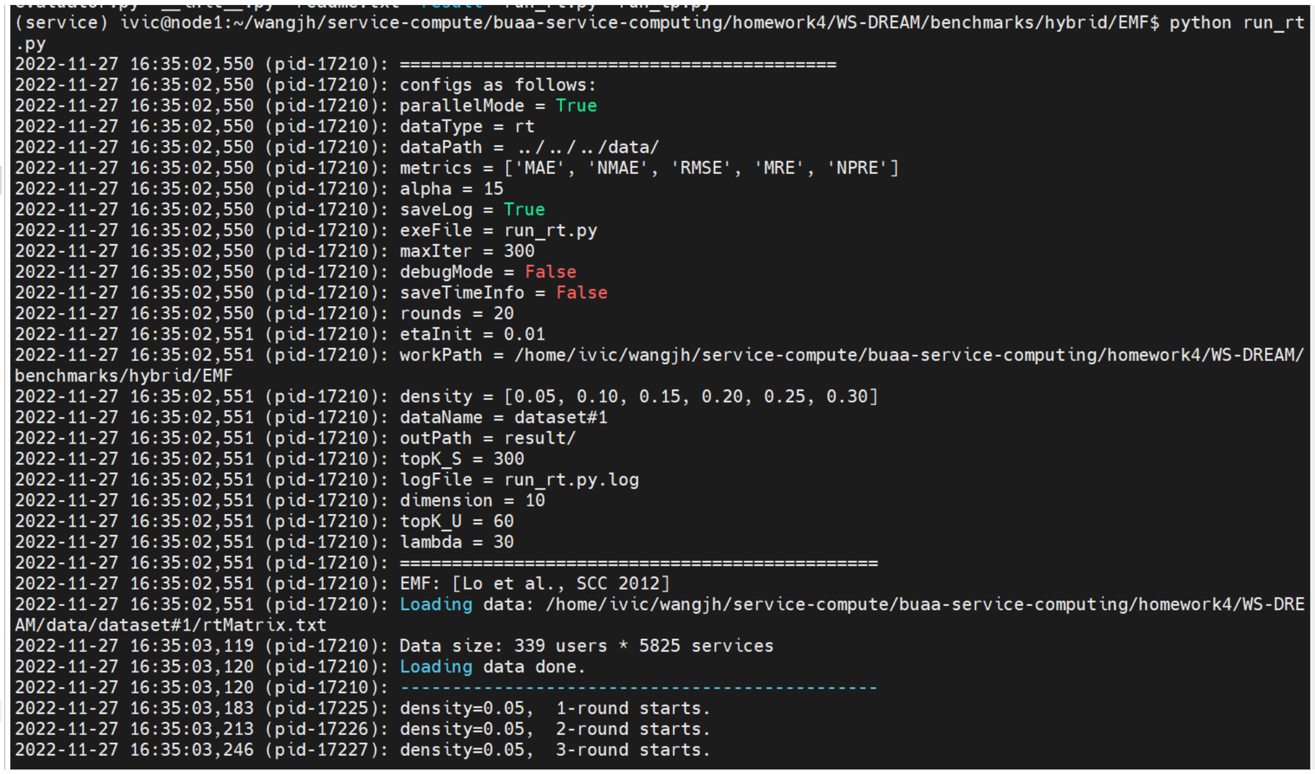
```
Algorithm 2: CloudPred Prediction Algorithm
   Input: $W$, $l$, $\lambda$
   Output: $W^*$
 1 Learn $V$ and $H$ by applying Algorithm 1 on $W$;
 2 for all $(u_i, u_j) \in U \times U$ do
 3     calculate the similarity $S(u_i, u_j)$ by Eq. (8);
 4 end
 5 for all $(c_i, c_j) \in C \times C$ do
 6     calculate the similarity $S(c_i, c_j)$ by Eq. (9);
 7 end
 8 for all $(i, j) \in \Lambda$ do
 9     construct similar user set $\Psi_i$ by Eq. (10);
10     construct similar component set $\Phi_j$ by Eq. (11);
11 end
12 for all $(i, j) \in \Omega - \Lambda$ do
13     calculate $w^u_{ij}$ by Eq. (12);
14     calculate $w^i_{ij}$ by Eq. (13);
15     $w^*_{ij} = \lambda \times w^u_{ij} + (1 - \lambda) \times w^c_{ij}$ ;
16 end
```

4. EMF (扩展矩阵分解)

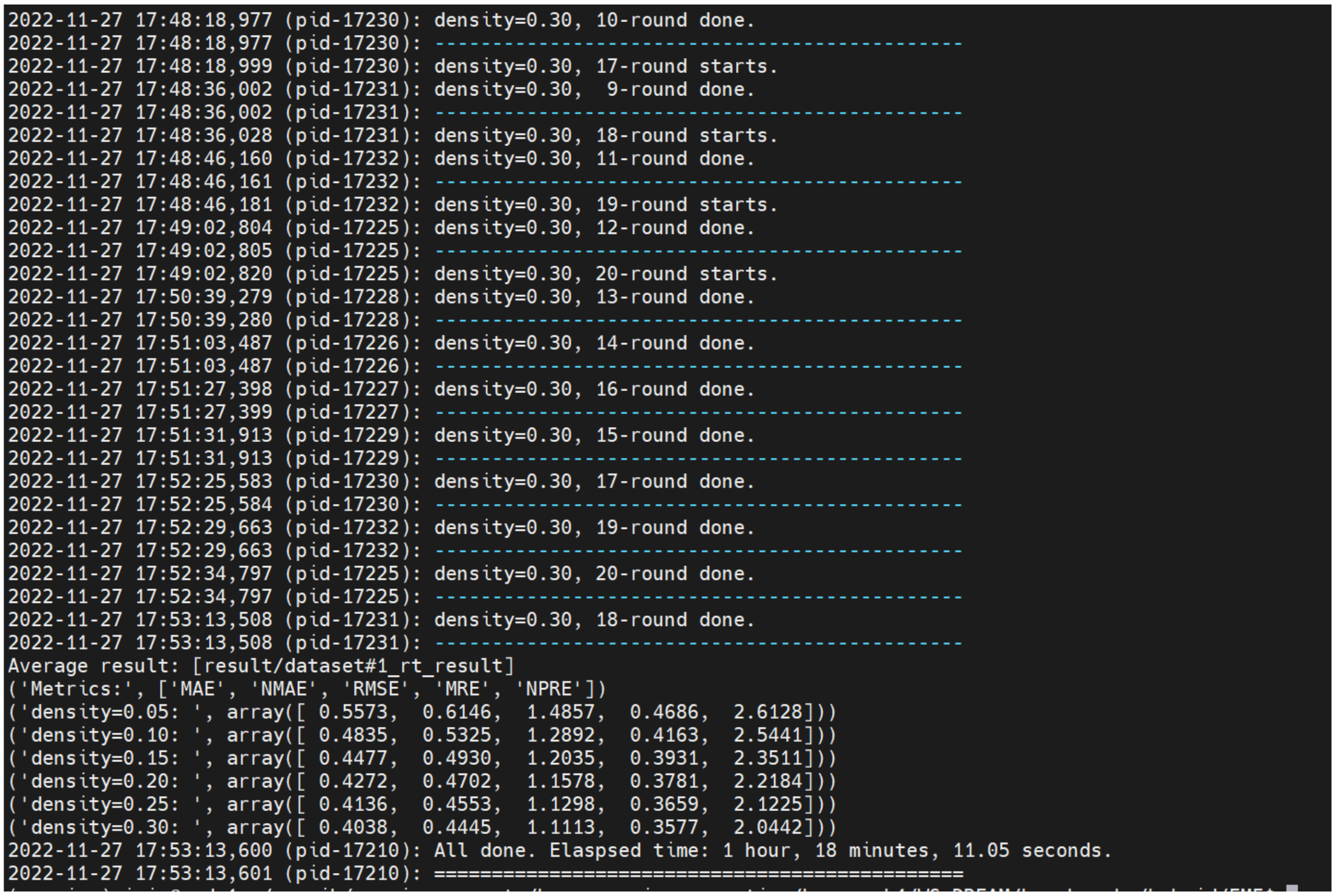
📌 **EMF： Extended Matrix Factorization**

文章：An Extended Matrix Factorization Approach for QoS Prediction in Service Selection

运行结果



```
(service) ivic@node1:~/wangjh/service-compute/buaa-service-computing/homework4/WS-DREAM/benchmarks/hybrid/EMF$ python run_rt
.py
2022-11-27 16:35:02,550 (pid-17210): ========================================
2022-11-27 16:35:02,550 (pid-17210): configs as follows:
2022-11-27 16:35:02,550 (pid-17210): parallelMode = True
2022-11-27 16:35:02,550 (pid-17210): dataType = rt
2022-11-27 16:35:02,550 (pid-17210): dataPath = ../../../data/
2022-11-27 16:35:02,550 (pid-17210): metrics = ['MAE', 'NMAE', 'RMSE', 'MRE', 'NPRE']
2022-11-27 16:35:02,550 (pid-17210): alpha = 15
2022-11-27 16:35:02,550 (pid-17210): saveLog = True
2022-11-27 16:35:02,550 (pid-17210): exeFile = run_rt.py
2022-11-27 16:35:02,550 (pid-17210): maxIter = 300
2022-11-27 16:35:02,550 (pid-17210): debugMode = False
2022-11-27 16:35:02,550 (pid-17210): saveTimeInfo = False
2022-11-27 16:35:02,550 (pid-17210): rounds = 20
2022-11-27 16:35:02,551 (pid-17210): etaInit = 0.01
2022-11-27 16:35:02,551 (pid-17210): workPath = /home/ivic/wangjh/service-compute/buaa-service-computing/homework4/WS-DREAM/
benchmarks/hybrid/EMF
2022-11-27 16:35:02,551 (pid-17210): density = [0.05, 0.10, 0.15, 0.20, 0.25, 0.30]
2022-11-27 16:35:02,551 (pid-17210): dataName = dataset#1
2022-11-27 16:35:02,551 (pid-17210): outPath = result/
2022-11-27 16:35:02,551 (pid-17210): topK_S = 300
2022-11-27 16:35:02,551 (pid-17210): logFile = run_rt.py.log
2022-11-27 16:35:02,551 (pid-17210): dimension = 10
2022-11-27 16:35:02,551 (pid-17210): topK_U = 60
2022-11-27 16:35:02,551 (pid-17210): lambda = 30
2022-11-27 16:35:02,551 (pid-17210): ==============================================
2022-11-27 16:35:02,551 (pid-17210): EMF: [Lo et al., SCC 2012]
2022-11-27 16:35:02,551 (pid-17210): Loading data: /home/ivic/wangjh/service-compute/buaa-service-computing/homework4/WS-DRE
AM/data/dataset#1/rtMatrix.txt
2022-11-27 16:35:03,119 (pid-17210): Data size: 339 users * 5825 services
2022-11-27 16:35:03,120 (pid-17210): Loading data done.
2022-11-27 16:35:03,120 (pid-17210): ----------------------------------------------
2022-11-27 16:35:03,183 (pid-17225): density=0.05,  1-round starts.
2022-11-27 16:35:03,213 (pid-17226): density=0.05,  2-round starts.
2022-11-27 16:35:03,246 (pid-17227): density=0.05,  3-round starts.
```



```
2022-11-27 17:48:18,977 (pid-17230): density=0.30, 10-round done.
2022-11-27 17:48:18,977 (pid-17230): ----------------------------------------------
2022-11-27 17:48:18,999 (pid-17230): density=0.30, 17-round starts.
2022-11-27 17:48:36,002 (pid-17231): density=0.30,  9-round done.
2022-11-27 17:48:36,002 (pid-17231): ----------------------------------------------
2022-11-27 17:48:36,028 (pid-17231): density=0.30, 18-round starts.
2022-11-27 17:48:46,160 (pid-17232): density=0.30, 11-round done.
2022-11-27 17:48:46,161 (pid-17232): ----------------------------------------------
2022-11-27 17:48:46,181 (pid-17232): density=0.30, 19-round starts.
2022-11-27 17:49:02,804 (pid-17225): density=0.30, 12-round done.
2022-11-27 17:49:02,805 (pid-17225): ----------------------------------------------
2022-11-27 17:49:02,820 (pid-17225): density=0.30, 20-round starts.
2022-11-27 17:50:39,279 (pid-17228): density=0.30, 13-round done.
2022-11-27 17:50:39,280 (pid-17228): ----------------------------------------------
2022-11-27 17:51:03,487 (pid-17226): density=0.30, 14-round done.
2022-11-27 17:51:03,487 (pid-17226): ----------------------------------------------
2022-11-27 17:51:27,398 (pid-17227): density=0.30, 16-round done.
2022-11-27 17:51:27,399 (pid-17227): ----------------------------------------------
2022-11-27 17:51:31,913 (pid-17229): density=0.30, 15-round done.
2022-11-27 17:51:31,913 (pid-17229): ----------------------------------------------
2022-11-27 17:52:25,583 (pid-17230): density=0.30, 17-round done.
2022-11-27 17:52:25,584 (pid-17230): ----------------------------------------------
2022-11-27 17:52:29,663 (pid-17232): density=0.30, 19-round done.
2022-11-27 17:52:29,663 (pid-17232): ----------------------------------------------
2022-11-27 17:52:34,797 (pid-17225): density=0.30, 20-round done.
2022-11-27 17:52:34,797 (pid-17225): ----------------------------------------------
2022-11-27 17:53:13,508 (pid-17231): density=0.30, 18-round done.
2022-11-27 17:53:13,508 (pid-17231): ----------------------------------------------
Average result: [result/dataset#1_rt_result]
('Metrics:', ['MAE', 'NMAE', 'RMSE', 'MRE', 'NPRE'])
('density=0.05: ', array([ 0.5573,  0.6146,  1.4857,  0.4686,  2.6128]))
('density=0.10: ', array([ 0.4835,  0.5325,  1.2892,  0.4163,  2.5441]))
('density=0.15: ', array([ 0.4477,  0.4930,  1.2035,  0.3931,  2.3511]))
('density=0.20: ', array([ 0.4272,  0.4702,  1.1578,  0.3781,  2.2184]))
('density=0.25: ', array([ 0.4136,  0.4553,  1.1298,  0.3659,  2.1225]))
('density=0.30: ', array([ 0.4038,  0.4445,  1.1113,  0.3577,  2.0442]))
2022-11-27 17:53:13,600 (pid-17210): All done. Elaspsed time: 1 hour, 18 minutes, 11.05 seconds.
2022-11-27 17:53:13,601 (pid-17210): ==============================================
```

方法原理

作者提出了一种具有关系正则化的扩展矩阵分解 (EMF) 框架来进行 QoS 值缺失预测。首先从一般角度阐述矩阵分解（MF）模型。为了准确收集人群的智慧，我们在用户端和服务端采用不同的相似性度量来识别邻域。然后作者在邻域内系统地设计了两个新的关系正则化项。最后，作者将这两个术语组合成一个统一的 MF 框架来预测丢失的 QoS 值。在传统的矩阵分解方法的基础上通过增加**相关用户**和**相关服务**这两个正则项来预测缺失的 QoS 值

文章中，定义 user-service 矩阵为 R，目标将矩阵 R 分解为 U 和 S 两个矩阵：

$$R \approx U^T S$$

目标使得估计的矩阵 R 更逼近原始矩阵，归结为最小化问题：

$$\min_{U,S} \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{n} \| R_{ij} - U_i^T S_j \|_F^2, \qquad (2)$$

现实情况下，原始矩阵 R 是一个稀疏矩阵，使用 I_{ij} 标记用户 u_i，与服务 s_j 是否进行了交互，原始问题修改为以下问题：

$$\min_{U,S} \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{n} I_{ij}(R_{ij} - U_i^T S_j)^2, \qquad (3)$$

为了避免过拟合，进行正则化：

$$\min_{U,S} \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{n} I_{ij}(R_{ij} - U_i^T S_j)^2 + \frac{\lambda_1}{2} \|U\|_F^2 + \frac{\lambda_2}{2} \|S\|_F^2, \quad (4)$$

EMF 融合了 user 和 service 进行 QoS 预测，

$$\begin{aligned} \min_{U,S} \mathcal{L}_3(R,U,S) = & \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{n} I_{ij}(R_{ij} - U_i^T S_j)^2 \\ & + \frac{\lambda_1}{2} \|U\|_F^2 + \frac{\lambda_2}{2} \|S\|_F^2 \\ & + \frac{\alpha_1}{2} \sum_{i=1}^{m} \|U_i - \sum_{f \in TU(i)} PU_{if} \cdot U_f\|_F^2 \\ & + \frac{\alpha_2}{2} \sum_{j=1}^{n} \|S_j - \sum_{h \in TS(j)} PS_{jh} \cdot S_h\|_F^2, \end{aligned} \qquad (19)$$

$\tau\mu(i)$ 表示用户 i 的基于 Top-k 的相邻用户，矩阵 PU 表示用户之间的经过归一化后的相似度，

$\tau s(j)$ 表示服务 j 的基于 Top-k 的相邻服务，矩阵 PS 表示服务之间的经过归一化后的相似度

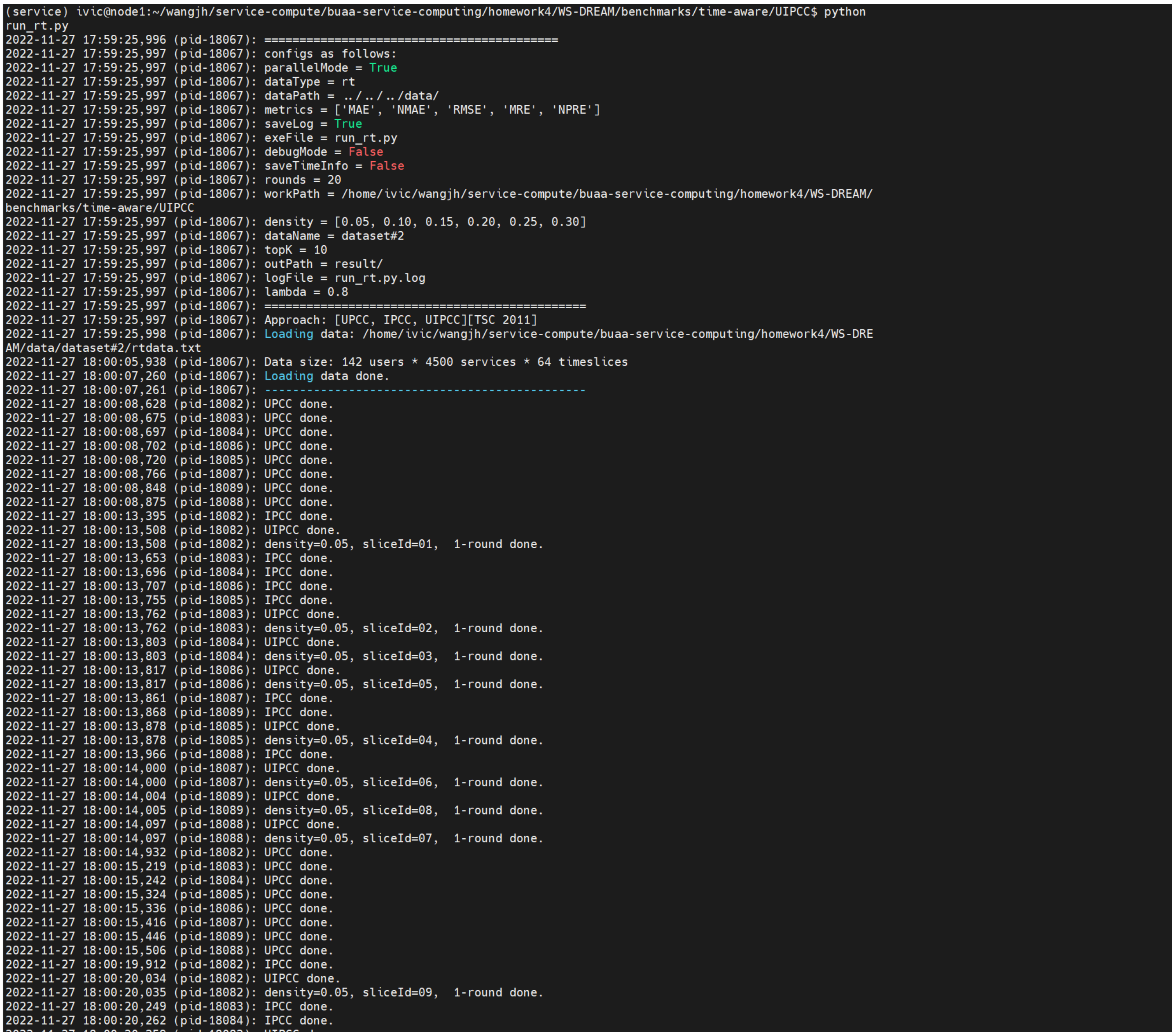
之后使用梯度下降的方法，寻找局部最优解。

5. UIPCC

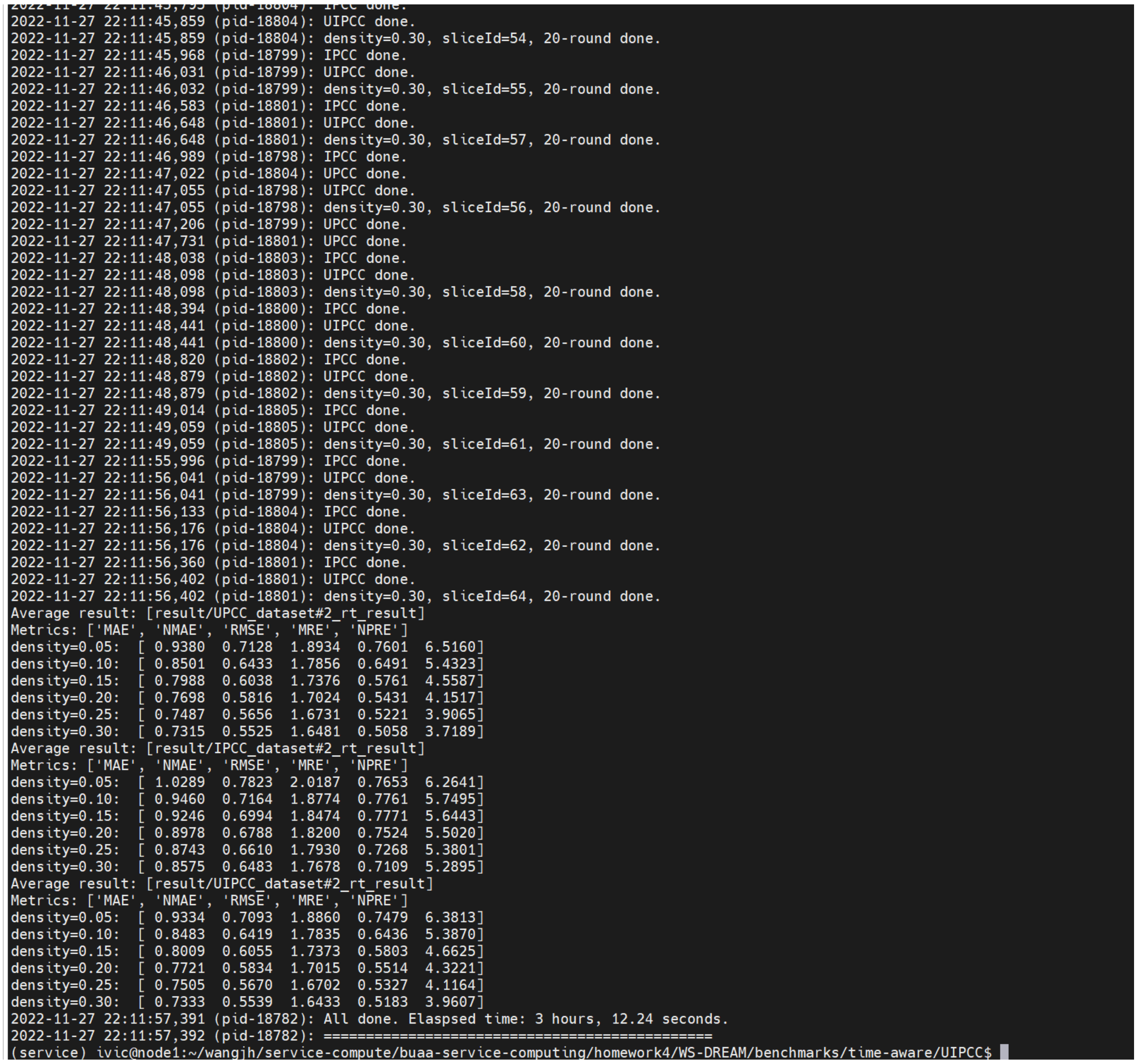
🔔 the user-based PCC approach and the item-based PCC approach

文章：QoS-Aware Web Service Recommendation by Collaborative Filtering

运行结果



```
(service) ivic@node1:~/wangjh/service-compute/buaa-service-computing/homework4/WS-DREAM/benchmarks/time-aware/UIPCC$ python
run_rt.py
2022-11-27 17:59:25,996 (pid-18067): ==========================================
2022-11-27 17:59:25,997 (pid-18067): configs as follows:
2022-11-27 17:59:25,997 (pid-18067): parallelMode = True
2022-11-27 17:59:25,997 (pid-18067): dataType = rt
2022-11-27 17:59:25,997 (pid-18067): dataPath = ../../../data/
2022-11-27 17:59:25,997 (pid-18067): metrics = ['MAE', 'NMAE', 'RMSE', 'MRE', 'NPRE']
2022-11-27 17:59:25,997 (pid-18067): saveLog = True
2022-11-27 17:59:25,997 (pid-18067): exeFile = run_rt.py
2022-11-27 17:59:25,997 (pid-18067): debugMode = False
2022-11-27 17:59:25,997 (pid-18067): saveTimeInfo = False
2022-11-27 17:59:25,997 (pid-18067): rounds = 20
2022-11-27 17:59:25,997 (pid-18067): workPath = /home/ivic/wangjh/service-compute/buaa-service-computing/homework4/WS-DREAM/
benchmarks/time-aware/UIPCC
2022-11-27 17:59:25,997 (pid-18067): density = [0.05, 0.10, 0.15, 0.20, 0.25, 0.30]
2022-11-27 17:59:25,997 (pid-18067): dataName = dataset#2
2022-11-27 17:59:25,997 (pid-18067): topK = 10
2022-11-27 17:59:25,997 (pid-18067): outPath = result/
2022-11-27 17:59:25,997 (pid-18067): logFile = run_rt.py.log
2022-11-27 17:59:25,997 (pid-18067): lambda = 0.8
2022-11-27 17:59:25,997 (pid-18067): ==========================================
2022-11-27 17:59:25,997 (pid-18067): Approach: [UPCC, IPCC, UIPCC][TSC 2011]
2022-11-27 17:59:25,998 (pid-18067): Loading data: /home/ivic/wangjh/service-compute/buaa-service-computing/homework4/WS-DRE
AM/data/dataset#2/rtdata.txt
2022-11-27 18:00:05,938 (pid-18067): Data size: 142 users * 4500 services * 64 timeslices
2022-11-27 18:00:07,260 (pid-18067): Loading data done.
2022-11-27 18:00:07,261 (pid-18067): ------------------------------------------
2022-11-27 18:00:08,628 (pid-18082): UPCC done.
2022-11-27 18:00:08,675 (pid-18083): UPCC done.
2022-11-27 18:00:08,697 (pid-18084): UPCC done.
2022-11-27 18:00:08,702 (pid-18086): UPCC done.
2022-11-27 18:00:08,720 (pid-18085): UPCC done.
2022-11-27 18:00:08,766 (pid-18087): UPCC done.
2022-11-27 18:00:08,848 (pid-18089): UPCC done.
2022-11-27 18:00:08,875 (pid-18088): UPCC done.
2022-11-27 18:00:13,395 (pid-18082): IPCC done.
2022-11-27 18:00:13,508 (pid-18082): UIPCC done.
2022-11-27 18:00:13,508 (pid-18082): density=0.05, sliceId=01,  1-round done.
2022-11-27 18:00:13,653 (pid-18083): IPCC done.
2022-11-27 18:00:13,696 (pid-18084): IPCC done.
2022-11-27 18:00:13,707 (pid-18086): IPCC done.
2022-11-27 18:00:13,755 (pid-18085): IPCC done.
2022-11-27 18:00:13,762 (pid-18083): UIPCC done.
2022-11-27 18:00:13,762 (pid-18083): density=0.05, sliceId=02,  1-round done.
2022-11-27 18:00:13,803 (pid-18084): UIPCC done.
2022-11-27 18:00:13,803 (pid-18084): density=0.05, sliceId=03,  1-round done.
2022-11-27 18:00:13,817 (pid-18086): UIPCC done.
2022-11-27 18:00:13,817 (pid-18086): density=0.05, sliceId=05,  1-round done.
2022-11-27 18:00:13,861 (pid-18087): IPCC done.
2022-11-27 18:00:13,868 (pid-18089): IPCC done.
2022-11-27 18:00:13,878 (pid-18085): UIPCC done.
2022-11-27 18:00:13,878 (pid-18085): density=0.05, sliceId=04,  1-round done.
2022-11-27 18:00:13,966 (pid-18088): IPCC done.
2022-11-27 18:00:14,000 (pid-18087): UIPCC done.
2022-11-27 18:00:14,000 (pid-18087): density=0.05, sliceId=06,  1-round done.
2022-11-27 18:00:14,004 (pid-18089): UIPCC done.
2022-11-27 18:00:14,005 (pid-18089): density=0.05, sliceId=08,  1-round done.
2022-11-27 18:00:14,097 (pid-18088): UIPCC done.
2022-11-27 18:00:14,097 (pid-18088): density=0.05, sliceId=07,  1-round done.
2022-11-27 18:00:14,932 (pid-18082): UPCC done.
2022-11-27 18:00:15,219 (pid-18083): UPCC done.
2022-11-27 18:00:15,242 (pid-18084): UPCC done.
2022-11-27 18:00:15,324 (pid-18085): UPCC done.
2022-11-27 18:00:15,336 (pid-18086): UPCC done.
2022-11-27 18:00:15,416 (pid-18087): UPCC done.
2022-11-27 18:00:15,446 (pid-18089): UPCC done.
2022-11-27 18:00:15,506 (pid-18088): UPCC done.
2022-11-27 18:00:19,912 (pid-18082): IPCC done.
2022-11-27 18:00:20,034 (pid-18082): UIPCC done.
2022-11-27 18:00:20,035 (pid-18082): density=0.05, sliceId=09,  1-round done.
2022-11-27 18:00:20,249 (pid-18083): IPCC done.
2022-11-27 18:00:20,262 (pid-18084): IPCC done.
```



```
2022-11-27 22:11:45,795 (pid-18804): IPCC done.
2022-11-27 22:11:45,859 (pid-18804): UIPCC done.
2022-11-27 22:11:45,859 (pid-18804): density=0.30, sliceId=54, 20-round done.
2022-11-27 22:11:45,968 (pid-18799): IPCC done.
2022-11-27 22:11:46,031 (pid-18799): UIPCC done.
2022-11-27 22:11:46,032 (pid-18799): density=0.30, sliceId=55, 20-round done.
2022-11-27 22:11:46,583 (pid-18801): IPCC done.
2022-11-27 22:11:46,648 (pid-18801): UIPCC done.
2022-11-27 22:11:46,648 (pid-18801): density=0.30, sliceId=57, 20-round done.
2022-11-27 22:11:46,989 (pid-18798): IPCC done.
2022-11-27 22:11:47,022 (pid-18804): UPCC done.
2022-11-27 22:11:47,055 (pid-18798): UIPCC done.
2022-11-27 22:11:47,055 (pid-18798): density=0.30, sliceId=56, 20-round done.
2022-11-27 22:11:47,206 (pid-18799): UPCC done.
2022-11-27 22:11:47,731 (pid-18801): UPCC done.
2022-11-27 22:11:48,038 (pid-18803): IPCC done.
2022-11-27 22:11:48,098 (pid-18803): UIPCC done.
2022-11-27 22:11:48,098 (pid-18803): density=0.30, sliceId=58, 20-round done.
2022-11-27 22:11:48,394 (pid-18800): IPCC done.
2022-11-27 22:11:48,441 (pid-18800): UIPCC done.
2022-11-27 22:11:48,441 (pid-18800): density=0.30, sliceId=60, 20-round done.
2022-11-27 22:11:48,820 (pid-18802): IPCC done.
2022-11-27 22:11:48,879 (pid-18802): UIPCC done.
2022-11-27 22:11:48,879 (pid-18802): density=0.30, sliceId=59, 20-round done.
2022-11-27 22:11:49,014 (pid-18805): IPCC done.
2022-11-27 22:11:49,059 (pid-18805): UIPCC done.
2022-11-27 22:11:49,059 (pid-18805): density=0.30, sliceId=61, 20-round done.
2022-11-27 22:11:55,996 (pid-18799): IPCC done.
2022-11-27 22:11:56,041 (pid-18799): UIPCC done.
2022-11-27 22:11:56,041 (pid-18799): density=0.30, sliceId=63, 20-round done.
2022-11-27 22:11:56,133 (pid-18804): IPCC done.
2022-11-27 22:11:56,176 (pid-18804): UIPCC done.
2022-11-27 22:11:56,176 (pid-18804): density=0.30, sliceId=62, 20-round done.
2022-11-27 22:11:56,360 (pid-18801): IPCC done.
2022-11-27 22:11:56,402 (pid-18801): UIPCC done.
2022-11-27 22:11:56,402 (pid-18801): density=0.30, sliceId=64, 20-round done.
Average result: [result/UPCC_dataset#2_rt_result]
Metrics: ['MAE', 'NMAE', 'RMSE', 'MRE', 'NPRE']
density=0.05:  [ 0.9380  0.7128  1.8934  0.7601  6.5160]
density=0.10:  [ 0.8501  0.6433  1.7856  0.6491  5.4323]
density=0.15:  [ 0.7988  0.6038  1.7376  0.5761  4.5587]
density=0.20:  [ 0.7698  0.5816  1.7024  0.5431  4.1517]
density=0.25:  [ 0.7487  0.5656  1.6731  0.5221  3.9065]
density=0.30:  [ 0.7315  0.5525  1.6481  0.5058  3.7189]
Average result: [result/IPCC_dataset#2_rt_result]
Metrics: ['MAE', 'NMAE', 'RMSE', 'MRE', 'NPRE']
density=0.05:  [ 1.0289  0.7823  2.0187  0.7653  6.2641]
density=0.10:  [ 0.9460  0.7164  1.8774  0.7761  5.7495]
density=0.15:  [ 0.9246  0.6994  1.8474  0.7771  5.6443]
density=0.20:  [ 0.8978  0.6788  1.8200  0.7524  5.5020]
density=0.25:  [ 0.8743  0.6610  1.7930  0.7268  5.3801]
density=0.30:  [ 0.8575  0.6483  1.7678  0.7109  5.2895]
Average result: [result/UIPCC_dataset#2_rt_result]
Metrics: ['MAE', 'NMAE', 'RMSE', 'MRE', 'NPRE']
density=0.05:  [ 0.9334  0.7093  1.8860  0.7479  6.3813]
density=0.10:  [ 0.8483  0.6419  1.7835  0.6436  5.3870]
density=0.15:  [ 0.8009  0.6055  1.7373  0.5803  4.6625]
density=0.20:  [ 0.7721  0.5834  1.7015  0.5514  4.3221]
density=0.25:  [ 0.7505  0.5670  1.6702  0.5327  4.1164]
density=0.30:  [ 0.7333  0.5539  1.6433  0.5183  3.9607]
2022-11-27 22:11:57,391 (pid-18782): All done. Elaspsed time: 3 hours, 12.24 seconds.
2022-11-27 22:11:57,392 (pid-18782): ==============================================
(service) ivic@node1:~/wangjh/service-compute/buaa-service-computing/homework4/WS-DREAM/benchmarks/time-aware/UIPCC$
```

方法原理

UIPCC 使用**协作过滤**的方法，利用 Web 服务用户过去的使用经验来预测 Web 服务的 QoS 值并提出 Web 服务推荐。 作者首先为过去从不同服务用户收集 Web 服务 QoS 信息提出了一种用户协作机制。 然后，基于收集的 QoS 数据，设计了一种协作过滤方法来预测 Web 服务 QoS 值。 最后，使用以 WSRec 的原型由 Java 语言实现，并部署到 Internet 上以进行实际实验。

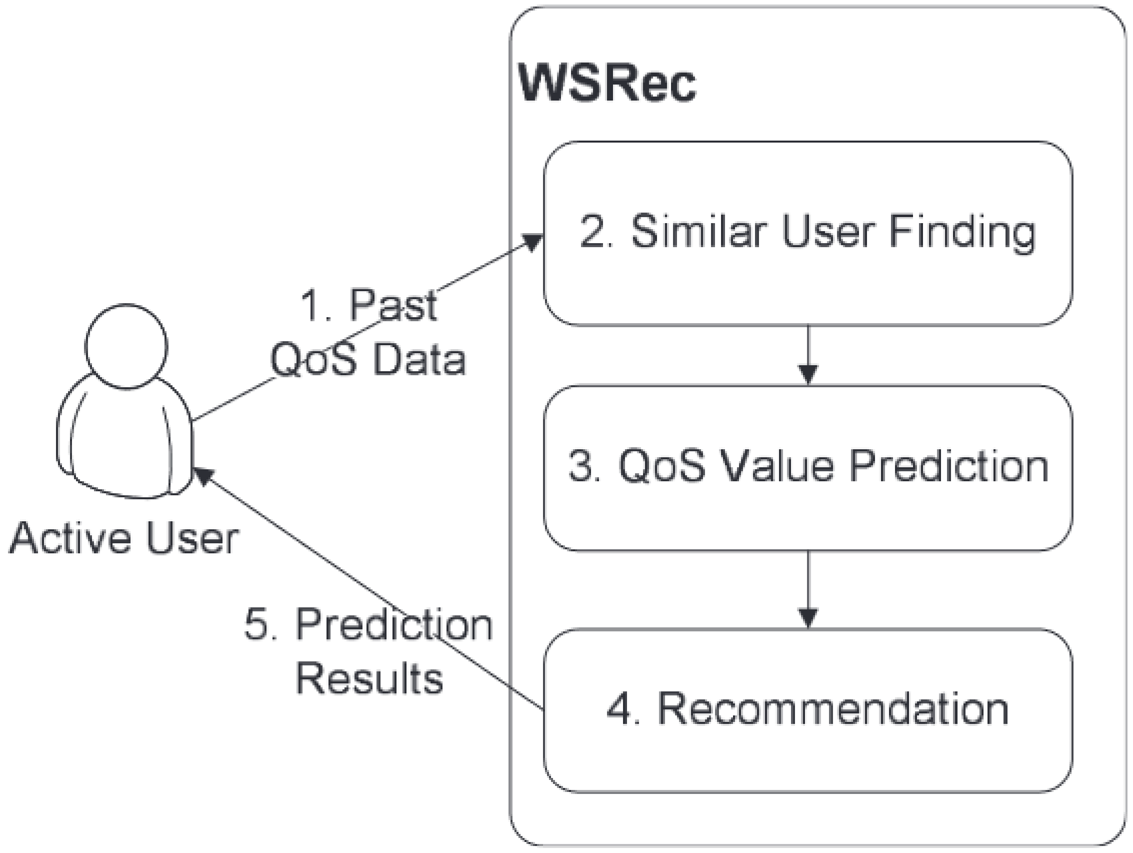


Fig. 1. Procedures of QoS Value Prediction.

流程：

1. 服务用户将过去的 Web 服务 QoS 数据贡献给中央服务器 WSRec 。

2. WSRec 从培训用户中选择类似用户作为活动用户。训练用户代表其 QoS 值存储在 WSRec 服务器中并用于为活动用户进行值预测的服务用户。

3. WSRec 预测活动用户的 Web 服务的 QoS 值。

4. WSRec 根据不同 Web 服务的预测 QoS 值提出 Web 服务推荐。

5. 服务用户接收预测的 QoS 值以及推荐结果，其可以被用于辅助决策（例如，服务选择，复合服务性能预测等。

1. 使用 Person 相关系数来计算两个服务用户 a、u 之间的相似度：

$$Sim(a,u)=\frac{\sum_{i\in I}(r_{a,i}-\overline{r}_a)(r_{u,i}-\overline{r}_u)}{\sqrt{\sum_{i\in I}(r_{a,i}-\overline{r}_a)^2}\sqrt{\sum_{i\in I}(r_{u,i}-\overline{r}_u)^2}}, \qquad (1)$$

计算两个 Web 服务之间的相似度：

$$Sim(i,j)=\frac{\sum_{u\in U}(r_{u,i}-\overline{r}_i)(r_{u,j}-\overline{r}_j)}{\sqrt{\sum_{u\in U}(r_{u,i}-\overline{r}_i)^2}\sqrt{\sum_{u\in U}(r_{u,j}-\overline{r}_j)^2}}, \qquad (2)$$

作者认为 PCC 会高估一些实际上并不相似，但恰巧有相似 QoS 的 Web 服务的相似度，因此，作者提出了改进的 PCC 计算定义：

$$Sim'(a,u)=\frac{2\times|I_a\cap I_u|}{|I_a|+|I_u|}Sim(a,u),$$

2. user-item 矩阵通常非常稀疏，这将极大的影响预测进度，文章提出了一种使得矩阵更密集的缺失值预测方法
    - 传统的 Top-K 算法根据 PCC 相似度对邻居进行排名，选择前 k 个相似的邻居进行缺失值预测，但实际上，用户项矩阵中，某些项可能邻居很少，甚至没有邻居，这将大大降低预测精度
    - 文章提出了增强的 Top K 算法，排除 PCC 相似度小于或等于 0 的邻居

$$S(u)=\{u_a|u_a\in T(u), Sim'(u_a,u)>0, u_a\neq u\},$$

    - QoS 值的预测公式：
        - 基于用户的方法：

$$P(r_{u,i}) = \overline{u} + \frac{\sum_{u_a \in S(u)} Sim'(u_a, u)(r_{u_a,i} - \overline{u}_a)}{\sum_{u_a \in S(u)} Sim'(u_a, u)},$$

- 基于项目的方法：

$$P(r_{u,i}) = \overline{i} + \frac{\sum_{i_k \in S(i)} Sim'(i_k, i)(r_{u,i_k} - \overline{i}_k)}{\sum_{i_k \in S(i)} Sim'(i_k, i)},$$

- 两方法结合：

$$\begin{aligned} P(r_{u,i}) = w_u \times \left(\overline{u} + \frac{\sum_{u_a \in S(u)} Sim'(u_a, u)(r_{u_a,i} - \overline{u}_a)}{\sum_{u_a \in S(u)} Sim'(u_a, u)} \right) \\ + w_i \times \left(\overline{i} + \frac{\sum_{i_k \in S(i)} Sim'(i_k, i)(r_{u,i_k} - \overline{i}_k)}{\sum_{i_k \in S(i)} Sim'(i_k, i)} \right), \end{aligned}$$

$$w_u = \frac{con_u \times \lambda}{con_u \times \lambda + con_i \times (1 - \lambda)}$$

$$w_i = \frac{con_i \times (1 - \lambda)}{con_u \times \lambda + con_i \times (1 - \lambda)},$$

con_u和con_i 可以自动平衡基于用户的预测和基于项目的预测，提高了方法适用于不同数据集的可行性。