深度学习-LSTM网络-代码-示例

原创 2016年07月14日 16:20:26

标签：深度学习 (http://so.csdn.net/so/search/s.do?q=深度学习&t=blog) / LSTM (http://so.csdn.net/so/search/s.do?q=LSTM&t=blog) / 阿里流行音乐预测 (http://so.csdn.net/so/search/s.do?q=阿里流行音乐预测&t=blog)

6021

一、 LSTM网络原理

1. 要点介绍

（1）LSTM网络用来处理带“序列”(sequence)性质的数据，比如时间序列的数据，像每天的股价走势情况，机械振动信号的时域波形，以及类似于自然语言这种本身带有顺序性质的由有序单词组合的数据。

（2）LSTM本身不是一个独立存在的网络结构，只是整个神经网络的一部分，即由LSTM结构取代原始网络中的隐层单元部分。

（3）LSTM网络具有“记忆性”。其原因在于不同“时间点”之间的网络存在连接，而不是单个时间点处的网络存在前馈或者反馈。如下图2中的LSTM单元（隐层单元）所示。图3是不同时刻情况下的网络展开图。图中虚线连接代表时刻，“本身的网络”结构连接用实线表示。

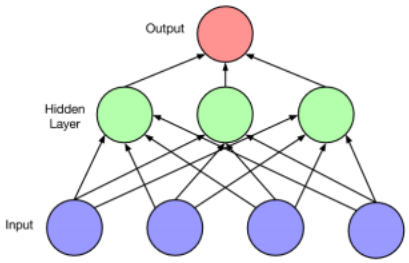


图1.普通神经网络

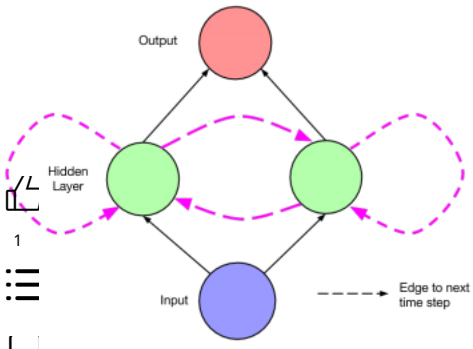


图2.简单的循环神经网络
(Recurrent Network)

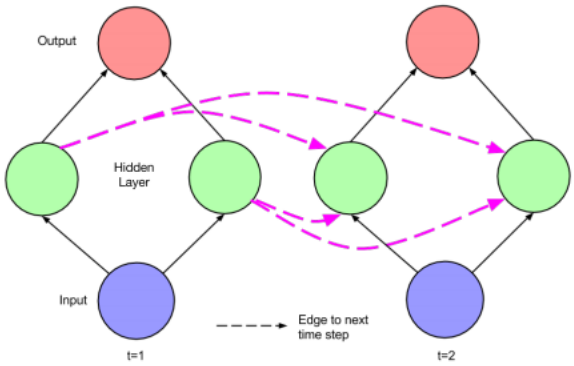


图3.循环神经网络在不同时刻展开图

2.LSTM单元结构图

图4，5是现在比较常用的LSTM单元结构示意图：

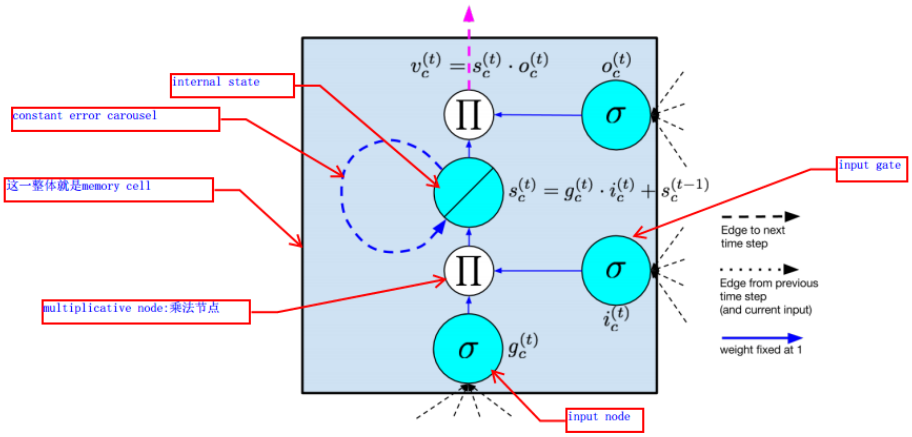


图4. LSTM memory cell as initially described in Hochreiter [Hochreiter and Schmidhuber, 1997]. The self-connected node is the internal state s. The diagonal line indicates that it is linear, i.e. no link function is applied there. Nodes marked "Π" output the product of their inputs. Dashed lines indicate recurrent edges and pink edges have fixed weight of 1.

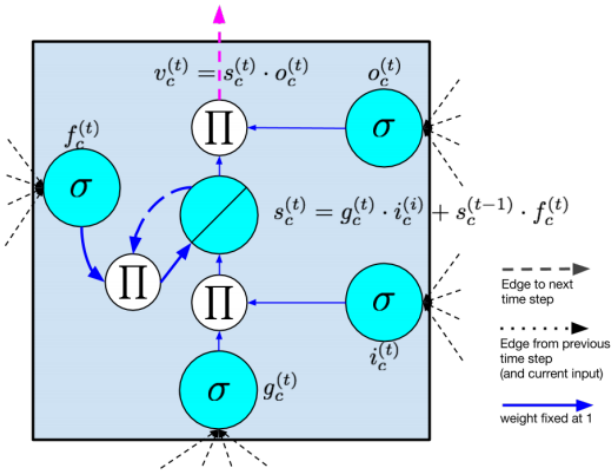


图5. LSTM memory cell with forget gate as described by Felix Gers et al.

其主要结构成分包含如下：

（1）输入节点input node：接受上一时刻隐层单元的输出及当前时刻是样本输入；

（2）输入门input gate：可以看到输入门会和输入节点的值相乘，组成LSTM中internal state单元值的一部分，当门的输出为1时，输入节点的激活值全部流向internal state，当门的输出为0时，输入节点的值对internal state没有影响。

（3）内部状态internal state。

（4）遗忘门forget gate：用于刷新internal state的状态，控制internal state的上一状态对当前状态的影响。

各节点及门与隐藏单元输出的关系参见图4，图5所示。

二、代码示例

1.示例介绍

主要以今年参加的"2016年阿里流行音乐趋势预测"为例。

时间过得很快，今天已是第二赛季的最后一天了，我从5.18开始接触赛题，到6.14上午10点第一赛季截止，这一期间，由于是线下赛，可以用到各种模型，而自己又是做深度学习（deep learning）方向的研究，所以选择了基于LSTM的循环神经网络模型，结果也很幸运，进入到了第二赛季。开始接触深度学习也有大半年了，能够将自己所学用到这次真正的实际生活应用中，结果也还可以，自己感觉很欣慰。突然意识到，自己学习生涯这么多年，我想"学有所成，学有所用"该是我今后努力的方向和动力了吧。

下面我简单的介绍一下今年的赛题：

官方给的"输入"：2张表，一张是用户行为表（时间跨度20150301-20150830）mars_tianchi_user_actions，主要描述用户对歌曲的收藏，下载，播放等行为，一张是歌曲信息表mars_tianchi_songs，主要用来描述歌曲所属的艺人，及歌曲的相关信息，如发行时间，初始热度，语言等。

用户行为表（mars_tianchi_user_actions）

列名	类型	说明	示例
user_id	String	用户唯一标识	7063b3d0c075a4d276c5f06f4327cf4a
song_id	String	歌曲唯一标识	effb071415be51f11e845884e67c0f8c
gmt_create	String	用户播放时间（unix时间戳表示）精确到小时	1426406400
action_type	String	行为类型：1，播放；2，下载，3，收藏	1
Ds	String	记录收集日（分区）	20150315

注：用户对歌曲的任意行为为一行数据。

样例：

	A	B	C	D	E
35	98862006b8c1b7378606652e96db452d	c2c03245b6f8460316fd252f1a2d9e4a	1426395600	1	20150315
36	2d5e3250752aa41f545d74e597900528	488326140 76d8cded18f44ca5a8fd56b	1426381200	1	20150315
37	b10316396c08845627cf7b02031d394a	3c2f8edfdc631f8e3504c98d0bef46cc	1426420800	1	20150315
38	d6535c66f278468d8fad7e0a840178d1	f3ca7994a5e9f200e7dd8273688b79e9	1426388400	2	20150315
39	bcbbacf9df0f2e1911eb5171dd92b9a6	8234f6562e44ef26622a03c01a554f76	1426417200	1	20150315
40	ed8787d6532f5a2245ea7bc470d5a4f1	ddc5c0d45d1ca96a58ad48343040c761	1426348800	1	20150315
41	95f078edbf9cf403794035 4d16bafb51	8346519879ba29da0532b9fa7a7a8df2	1426384800	2	20150315
42	b0593a4550d5778bb499bb63fa5e417a	85b191e5ac39d2e99b9b81f3a0c7548d	1426406400	1	20150315
43	769e0cfbcd519acef424d9a2d334e629	4b50065daa3e433d623dd6a19b347012	1426417200	1	20150315
44	6ddc712848098b048a1d10fc22e01df6	c880db958bcb5efec70edd60cb2739de	1426352400	1	20150315
45	d793da41d050ca8372dc554a44b75e62	7fff79af0bbd8cd3c8be3ff0e03e8c16	1426420800	1	20150315
46	d793da41d050ca8372dc554a44b75e62	7c61629c3eb23bd885c27e54706a01a7	1426420800	1	20150315
47	1b4c2477d391542c0eaa9a75b49828a4	054f864f76ae81ef81dbac2bc82f04b9	1426374000	1	20150315
48	1b4c2477d391542c0eaa9a75b49828a4	054f864f76ae81ef81dbac2bc82f04b9	1426363200	1	20150315
49	dcc50b5608ce5f8f1543271ab8fd93a0	3fe3e19efffc34e68f45fb40fe631568	1426363200	1	20150315
50	cc8fd21fa10f058a1f43a2f6106199f1	1af9707e7f29d6a977f76e3e62eb4ec8	1426410000	1	20150315
51	c8051a85cd608f51d18c6911ba320035	0a9c800f081af3e238885309c0201ace	1426431600	1	20150315
52	fd03286256f8d10e0f07c41f763ab93c	343ba82f88847b2d361396d87bce92fb	1426417200	1	20150315
53	f6ef2623a6804af88b51764d63e81d1f	8a27d9a6c59628c991c154e8d93f412e	1426381200	1	20150315
54	eb732eafbef607d877798a43474ade2b	a047d7747fe722f31a1a793229e1cd39	1426388400	1	20150315
55	eddd8686e564dbebeadafffcdd6cc1fd	b7cf237adcc31164d7d65c2426e9b3fa	1426420800	1	20150315
56	4bf2c26fbaa488a60b20a86ca8b47518	8a27d9a6c59628c991c154e8d93f412e	1426413600	1	20150315
57	3dfd3e98827ce92e10de3f9e0ab9b2d9	12e449b15a0dcc0611215712dda8b399	1426410000	1	20150315
58	08b087618bad921e1d1bb46dfd1b68ba	5fc3eae9ac61fceef0b89d6a95e294f1	1426395600	1	20150315
59	d08d9f96875c23737e31c9ec2ff4b832	b77d71a43f6343e1db9947b677691e32	1426399200	1	20150315
60	15401f388c0ce0f0a37b86893ec80a85	aed2f6db07d0b66b8dcb41b0334a6f36	1426402800	1	20150315
61	a2cc2d0baff09d260a6f2c220a870a72	00005ab3116dc3229230566653b8f77a	1426417200	1	20150315
62	bbc0e6133b9470922802ef1ef7f6a1e1	8a27d9a6c59628c991c154e8d93f412e	1426384800	1	20150315
63	76b01ac34338f841b9df017ffb6a9c51	46e9d5f148610d3b7f379f9d61c2958b	1426428000	1	20150315
64	154dd78bebef4975c2dab7337bb478dc	31486fcca971e974f16bbb0885e38fe3	1426413600	1	20150315
65	e17d560b05e055c29710c6c0cb1d9441	c75eb8b7ef3c0fb605a501ca6bbed98e	1426424400	1	20150315
66	13b2d309bd281e4c84d577612f28d063	2660daee2f1db48a100b359b5a011287	1426420800	1	20150315
67	c2f04cc90ba8982f9dd1a9f2ba541420	b6591390f69171c20816d8acc6ec77b8	1426420800	1	20150315
68	8c0ffdcc57444677db9a305958b42182	bb421c387c984442a2c0deda70aa30bd	1426402800	1	20150315

歌曲艺人（mars_tianchi_songs）

列名	类型	说明	示例
song_id	String	歌曲唯一标识	c81f89cf7edd24930641afa2e411b09c
artist_id	String	歌曲所属的艺人Id	03c6699ea836decbc5c8fc2dbae7bd3b
publish_time	String	歌曲发行时间，精确到天	20150325
song_init_plays	String	歌曲的初始播放数，表明该歌曲的初始热度	0
Language	String	数字表示1,2,3...	100
Gender	String	1,2,3	1

样例：

	A	B	C	D	E	F
1	c81f89cf7edd24930641afa2e411b09c	03c6699ea836decbc5c8fc2dbae7bd3b	20150325	0	100	1
2	c0d7130777c1f1c417e78646946ed909	03c6699ea836decbc5c8fc2dbae7bd3b	20150325	0	100	1
3	200c9131cf929bab418d380356be5f42	03c6699ea836decbc5c8fc2dbae7bd3b	20150325	0	100	1
4	78fedfdf13fc820e363e39986ff91e94	03c6699ea836decbc5c8fc2dbae7bd3b	20110910	1717	1	1
5	95b99faf432d33772d63f828bf2d0921	03c6699ea836decbc5c8fc2dbae7bd3b	20110910	434	1	1
6	95dc772fcc939df641a0eca602e37983	03c6699ea836decbc5c8fc2dbae7bd3b	20110910	1760	1	1
7	7154422d75a916f44869c7c5ef532ba7	03c6699ea836decbc5c8fc2dbae7bd3b	20120601	3853	1	1
8	eff36b5[illegible]ad6ec91223453ff5d9ab1b	03c6699ea836decbc5c8fc2dbae7bd3b	20130817	871	1	1
9	1a2c59[illegible]b52467bf09123eed276a4b665	03c6699ea836decbc5c8fc2dbae7bd3b	20141031	1657	1	1
10	da318831c172bcbea5afd6dbbc634061	03c6699ea836decbc5c8fc2dbae7bd3b	20110601	6615	1	1
11	7cc510a[illegible]49b5a7a056a09c9934c3cdc7	03c6699ea836decbc5c8fc2dbae7bd3b	20141031	2420	1	1
12	5a53b64811cc40a6d700241f520f23fa	03c6699ea836decbc5c8fc2dbae7bd3b	20141031	1830	1	1
13	b1f753[illegible]ef7f966911d121c020c616	03c6699ea836decbc5c8fc2dbae7bd3b	20141031	2697	1	1
14	d12ff2[illegible]293d5fb43409193ff296e2	03c6699ea836decbc5c8fc2dbae7bd3b	20150214	148	1	1
15	56ec869a24b35ce6647b9b0910a7b5de	03c6699ea836decbc5c8fc2dbae7bd3b	20150601	0	1	1
16	17e985a4914713901302c27bbf1c38a3	03c6699ea836decbc5c8fc2dbae7bd3b	20150601	0	1	1
17	84b532[illegible]da4d8bac9417886a8af987ddf	03c6699ea836decbc5c8fc2dbae7bd3b	20120201	2563	100	1
18	10952b[illegible]c69d4f433111d0ddc24f03e	03c6699ea836decbc5c8fc2dbae7bd3b	20141027	36866	1	1
19	49714e6e03981677a4bd851f480ea5b8	03c6699ea836decbc5c8fc2dbae7bd3b	20120104	1942	1	1
20	62ec92[illegible]0e93f8b7f6be799ac1b4143c0	03c6699ea836decbc5c8fc2dbae7bd3b	20150325	0	100	1
21	b0ee23[illegible]0958603a9901963cba427a	03c6699ea836decbc5c8fc2dbae7bd3b	20110910	1178	1	1
22	63b9db[illegible]ebb44676108fc10bf32f0ee	03c6699ea836decbc5c8fc2dbae7bd3b	20120101	9030	1	1
23	9448419ce854ff7e9b58b17a1dbcde14	03c6699ea836decbc5c8fc2dbae7bd3b	20150325	0	100	1
24	bf187333[illegible]755e02fffe176321e92839	03c6699ea836decbc5c8fc2dbae7bd3b	20110910	546	1	1
25	3a9e3d[illegible]2164feebd3620ac90f719c1	03c6699ea836decbc5c8fc2dbae7bd3b	20130815	23744	1	1
26	bd7971816d28420e634052666bc25ab5	03c6699ea836decbc5c8fc2dbae7bd3b	20150325	0	100	1
27	dad213649fc269d8517a4fe20bb8a97b	03c6699ea836decbc5c8fc2dbae7bd3b	20130817	2139	1	1
28	234e8b6184aa9500aae011c756c40767	03c6699ea836decbc5c8fc2dbae7bd3b	20150601	0	1	1
29	096fad8ee853bbe6ee035b7c8448db61	03c6699ea836decbc5c8fc2dbae7bd3b	20150325	0	100	1
30	40a1f75d7bb7ebd00e28242af73e7251	03c6699ea836decbc5c8fc2dbae7bd3b	20141031	1742	1	1
31	a610a88216e17e4f13b81d6391c2d909	03c6699ea836decbc5c8fc2dbae7bd3b	20120104	3430	1	1
32	343880a8016560e68b65acc74e7c061b	03c6699ea836decbc5c8fc2dbae7bd3b	20141027	98427	1	1
33	f3f770e3118cda5345f3f15aaf050d88	03c6699ea836decbc5c8fc2dbae7bd3b	20140901	11147	1	1
34	63b4458b49e8a6bc502562a737091574	03c6699ea836decbc5c8fc2dbae7bd3b	20120101	7050	1	1

官方要求“输出”：预测随后2个月（20150901-20151030）每个歌手每天的播放量。输出格式：

选手提交结果表（mars_tianchi_artist_plays_predict）

列名	类型	说明	示例
artist_id	String	歌曲所属的艺人Id	023406156015ef87f99521f3b343f71f
Plays	String	艺人当天的播放数据	5000
Ds	String	日期	20150901

选手需要预测9月1日至10月30日60天内所有艺人的结果。

2.初赛所用模型思路

由于是对歌手的播放量进行预测，所以直接对每个歌手的“播放量”这一对象进行统计，查看在20150301-20151030这8个月内歌手的播放量变化趋势，并以每天的播放量，连续3天的播放均值，连续3天的播放方差，作为一个时间点的样本，“滑动”构建神经网络的训练集。网络的构成如下：

（1）输入层：3个神经元，分别代表播放量，播放均值，播放方差；

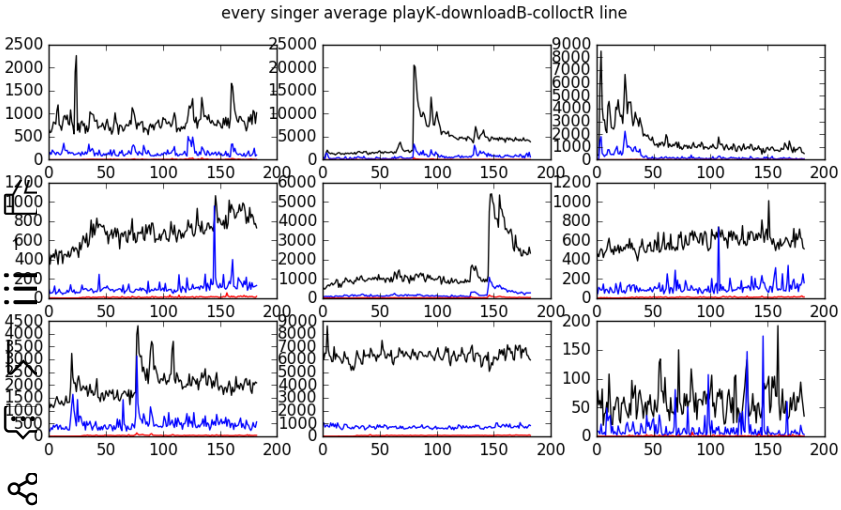
（2）第一隐层：LSTM结构单元，带有35个LSTM单元；

（3）第二隐层：LSTM结构单元，带有10个LSTM单元；

（4）输出层：3个神经元，代表和输入层相同的含义。

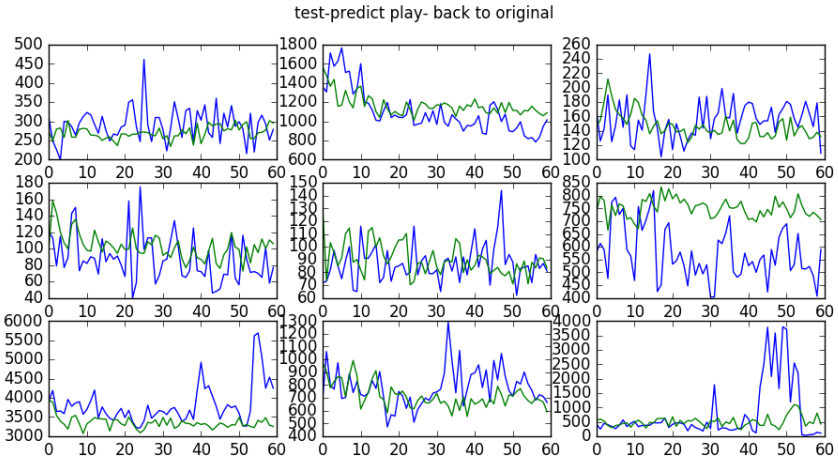
目标函数：重构误差。

下图是某些歌手的播放统计曲线：



2.预测结果

蓝色代表歌手真实的播放曲线，绿色代表预测曲线：



三、代码

运行环境：windows下的spyder

语言：python 2.7，以及Keras深度学习库。

由于看这个赛题前，没有一点Python基础，所以也是边想思路边学Python，对Python中的数据结构不怎么了解，所以代码写得有点烂。但整个代码是可以运行无误的。这也是初赛时代码的最终版本。

```
# -*- coding: utf-8 -*-
"""
Created on Wed Jun 01 16:34:45 2016

@author: Richer
"""
#%%修改记录
#1.将最后一层激活函数改为线性
#2.歌手播放曲线以歌曲量均值化（被第 4 点替换掉了）
#3.加入均值滤波 和 均值特征
#4.分别对每个歌手进行归一化处理（每个歌手之间相差太大了）
#5.对歌手进行聚类(效果不好)

#%% 时间序列及字典
from __future__ import division
import pandas as pd
import pdb
#import time

_DEBUG = False
_ISTEST = False

tempList = pd.date_range(start = '20150301',end = '20150830')
i = 0
dateList = [] #给出的数据集所在的时间序列
while i < len(tempList):
    strTemp = str(tempList[i])[:10]
    strTemp = strTemp.replace('-','')
    dateList.append(strTemp)
    i = i + 1
recDict = {}.fromkeys(dateList,0)  # 给出的数据集所在的时间序列字典
del tempList,i,strTemp

tempList = pd.date_range(start = '20150831', end = '20151030')
i = 0
objDateL = []  #要预测的目标时间序列
while i < len(tempList):
    strTemp = str(tempList[i])[:10]
    strTemp = strTemp.replace('-','')
    objDateL.append(strTemp)
    i += 1
del strTemp, i

## 异常数据信息
newSongExcep = 0                    # 用户表中出现的新歌曲
userDsExcep = 0                     # 用户表行为不在20150301-20150830

#%% 表处理---歌曲艺人数据

from copy import deepcopy
fileSong = open("p2_mars_tianchi_songs.csv")
songData = fileSong.readlines()

bigSongDict = {}  # 以歌曲为中心的大表
for songInfo in songData:
    songInfo = songInfo.replace('\n','')
```

```
    arrayInfo = songInfo.split(',')

    bigSongDict[arrayInfo[0]] = {} # 注：此处需要初始化，否则会出错
    bigSongDict[arrayInfo[0]]['artist_id'] = arrayInfo[1]
    bigSongDict[arrayInfo[0]]['publish_time'] = arrayInfo[2]
    bigSongDict[arrayInfo[0]]['song_init_plays'] = arrayInfo[3]
    bigSongDict[arrayInfo[0]]['Language'] = arrayInfo[4]
    bigSongDict[arrayInfo[0]]['Gender'] = arrayInfo[5]
    bigSongDict[arrayInfo[0]]['nUser'] = 0                          #用户数目
    bigSongDict[arrayInfo[0]]['playRec'] = deepcopy(recDict)          #播放记录
    bigSongDict[arrayInfo[0]]['downloadRec'] = deepcopy(recDict)      #下载记录
    bigSongDict[arrayInfo[0]]['collоctRec'] = deepcopy(recDict)       #收藏记录

fileSong.close()
del songData,arrayInfo,songInfo

# 用户行为数据

fileUser = open("p2_mars_tianchi_user_actions.csv")
userData = fileUser.readlines()

for userInfo in userData:
    userInfo = userInfo.replace('\n','')
    arrUser = userInfo.split(',')

    if (arrUser[1] in bigSongDict):
        bigSongDict[arrUser[1]]['nUser'] += 1
        if arrUser[3] == '1':
            bigSongDict[arrUser[1]]['playRec'][arrUser[4]] += 1
        if arrUser[3] == '2':
            bigSongDict[arrUser[1]]['downloadRec'][arrUser[4]] += 1
        if arrUser[3] == '3':
            bigSongDict[arrUser[1]]['collоctRec'][arrUser[4]] += 1
    else:
        newSongExcep = newSongExcep + 1

fileUser.close()
del userData,userInfo,arrUser


#%%统计每个艺人的播放，下载，收藏的变化曲线（20150301-20150830）

from collections import Counter
singerDict = {}  #歌手信息统计
for songKey in bigSongDict.keys():
    theArtist = bigSongDict[songKey]['artist_id']
    if (theArtist in singerDict):
# dict(Counter())会把 0 值去掉
        # 对应的 key 相加
        singerDict[theArtist]['playRec'] = dict(Counter(singerDict[theArtist]['playRec']) + Counter(bigSongDict[songKey]['playRe
        singerDict[theArtist]['downloadRec'] = dict(Counter(singerDict[theArtist]['downloadRec']) + Counter(bigSongDict[songK
        singerDict[theArtist]['collоctRec'] = dict(Counter(singerDict[theArtist]['collоctRec']) + Counter(bigSongDict[songKey]['co
        singerDict[theArtist]['nSongs'] += 1
    else:
        singerDict[theArtist] = {}
        singerDict[theArtist]['playRec'] = deepcopy(bigSongDict[songKey]['playRec'])
        singerDict[theArtist]['downloadRec'] = deepcopy(bigSongDict[songKey]['downloadRec'])
```

```
        singerDict[theArtist]['colloctRec'] = deepcopy(bigSongDict[songKey]['colloctRec'])
        singerDict[theArtist]['nSongs'] = 1

#%%将singerDict中字典转换为序列-按日期排序

import numpy as np
singerInfoList = {}
tpPlayList = []    # 播放列表
tpDownList = []    # 下载列表
tpCollectList = []  # 收藏列表
artList = []       # 歌手列表

i = 0

for singer in singerDict.keys():
    artList.append(singer)
    singerInfoList[singer] = {}
    #numSongs = singerDict[singer]['nSongs']  #对应歌手的歌曲数量
    while i < len(dateList):
        if (dateList[i] in singerDict[singer]['playRec'].keys()):
            tpPlayList.append(singerDict[singer]['playRec'][dateList[i]])
        else:
            tpPlayList.append(0)
        if (dateList[i] in singerDict[singer]['downloadRec'].keys()):
            tpDownList.append(singerDict[singer]['downloadRec'][dateList[i]])
        else:
            tpDownList.append(0)
        if(dateList[i] in singerDict[singer]['colloctRec'].keys()):
            tpCollectList.append(singerDict[singer]['colloctRec'][dateList[i]])
        else:
            tpCollectList.append(0)
        i += 1
    i = 0

    meanPlays = np.mean(tpPlayList)
    stdPlays = np.std(tpPlayList)
    singerInfoList[singer]['meanPlay'] = meanPlays
    singerInfoList[singer]['stdPlay'] = stdPlays
    singerInfoList[singer]['maxPlay'] = (abs((np.array(tpPlayList) - meanPlays) / stdPlays)).max()

    singerInfoList[singer]['playRec'] = deepcopy(tpPlayList)
    singerInfoList[singer]['downloadRec'] = deepcopy(tpDownList)
    singerInfoList[singer]['colloctRec'] = deepcopy(tpCollectList)



    del tpPlayList, tpDownList, tpCollectList
    tpPlayList = []
    tpDownList = []
    tpCollectList = []

del tpPlayList, tpDownList, tpCollectList, singer,meanPlays,stdPlays

#%%对每个歌手的播放曲线进行FFT变换
import matplotlib.pyplot as plt
import math

```

```
#i = 0
#if _ISTEST == True:
#    while i < len(singerInfoList):
#        flagY = i % 9
#        if flagY ==0:
#            plt.figure(figsize = (10,8), dpi = 150)
#            plt.suptitle('FFT process')
#        plt.subplot(3,3,flagY + 1)
#        fAmp = np.fft.fft(singerInfoList[artList[i]]['playRec']) / len(dateList)
#        plt.stem(abs(fAmp[1:(len(fAmp)/2)]))
#        i += 1
#        del fAmp
#
#pdb.set_trace()

#predictTestFFT = {} #使用FFT回归预测结果
#playLth = 0 #选取播放序列的长度做FFT
#chsNum = np.ones(len(singerInfoList),dtype=np.int) * 1  #选择前10个峰值做趋势预测
##chsNum[0] = 10
##chsNum[5] = 10
##chsNum[7] = 10
##chsNum[8] = 10
##chsNum[10] = 10
##chsNum[17] = 10
##chsNum[21] = 10
##chsNum[22] = 10
#
#if _ISTEST == True:
#    playLth = len(dateList) - len(objDateL)
#else:
#    playLth = len(dateList)
#
#j = 0 #歌手索引
#i = 0 #FFT索引
#while j < len(singerInfoList):
#    i = 0
#    ampFFT = np.fft.fft(singerInfoList[artList[j]]['playRec'][:playLth]) / playLth
#    sortInd = sorted(xrange(len(ampFFT)),key = (abs(ampFFT)).__getitem__,reverse = True) #降序排列
#    chsAmp = np.zeros(chsNum[j])
#    while i < chsNum[j]:
#        chsAmp[i] = ampFFT[sortInd[i]]
#        i += 1
#    dateRcon = np.zeros((playLth + len(objDateL)))
#    ind = np.arange(0,len(dateRcon),1.0) / len(ampFFT) * (2 * np.pi)
#    for k, p in enumerate(chsAmp):
#        if k != 0:
#            p *= 2
#        dateRcon += np.real(p) * np.cos(k * ind)
#        dateRcon -= np.imag(p) * np.sin(k * ind)
#    predictTestFFT[artList[j]] = {}
#    predictTestFFT[artList[j]]['playRec'] = deepcopy((list(dateRcon))[playLth:(playLth + len(objDateL))])
#
#    if _ISTEST == True:
#        flagY = j % 9
#        if flagY == 0:
#            plt.figure(figsize = (10,8),dpi = 150)
#            plt.suptitle('predict test play - use fft')
```

```
#       plt.subplot(3,3,flagY + 1)
#       plt.plot(singerInfoList[artList[j]]['playRec'][playLth:(playLth + len(objDateL))],'b')
#       plt.plot(predictTestFFT[artList[j]]['playRec'],'g')
#    j += 1
#    del ampFFT,sortInd,chsAmp,dateRcon,ind
#
#
#pdb.set_trace()
 1
#%% 绘制歌手播放，下载，收藏曲线

xVal = range(len(dateList)) #x坐标值
i = 0

while i < len(singerInfoList):  # 每个歌手播放曲线
    flagY = i % 9
    if flagY == 0:
        plt.figure(figsize = (10,8), dpi = 150)
        plt.suptitle('every singer average playK-downloadB-collectR line')
    plt.subplot(3,3,flagY + 1)
    plt.plot(singerInfoList[artList[i]]['playRec'],'k')
    plt.plot(singerInfoList[artList[i]]['downloadRec'],'b')
    plt.plot(singerInfoList[artList[i]]['collectRec'],'r')

    i += 1


del flagY

#%%提取歌手的标准差信息并进行排序

#nCls = 1  #分类数
#clsTh = 0 #第几类
#
#nSgrToCls = [] #每类的歌手数量列表
#stdPlayList = [] #所有歌手标准差列表
#indStdList = []  #排序后的数据在原始序列中的索引
#
#i = 0
#while i < len(artList):
#    stdPlayList.append(singerInfoList[artList[i]]['stdPlay'])
#    i += 1
#
#indStdList = sorted(xrange(len(stdPlayList)),key = stdPlayList.__getitem__) #默认降序排列
#
#i = 0
#while i < (nCls - 1):
#    nSgrToCls.append(int(len(singerInfoList) / nCls))
#    i += 1
#if nCls == 1:
#    nSgrToCls.append(int(len(singerInfoList)))
#else:
#    nSgrToCls.append(int(len(singerInfoList) - (nCls - 1) * nSgrToCls[0]))
#
#nObjSgr = nSgrToCls[clsTh]  #目标歌手数量
#objInd = []                 #初始化-对应的索引
#if clsTh == (nCls -1):
```

```
#    objInd = indStdList[( (nCls - 1) * nSgrToCls[0] ):]
#else:
#    objInd = indStdList[(clsTh * nSgrToCls[0]):((clsTh + 1) * nSgrToCls[0])]

nObjSgr = len(singerInfoList)
objInd = range(nObjSgr)

#%% 将singerDict 的 playRec downloadRec colloctRec按时间顺序转换为list
# 且分别对每个歌手数据进行归一化

playList = []   #大播放列表
downList = []   # 大下载列表
collectList = [] #大收藏列表
avePlayList = [] # 播放曲线的均值滤波后曲线
varPlayList = [] #实际上是标准差曲线


i = 0
while i < nObjSgr:
    artSg = artList[objInd[i]]
    meanPlays = singerInfoList[artSg]['meanPlay']
    stdPlays = singerInfoList[artSg]['stdPlay']
    maxPlays = singerInfoList[artSg]['maxPlay']

    playList = playList + list( (np.array(singerInfoList[artSg]['playRec']) - meanPlays) / (stdPlays * maxPlays) )
    downList = downList + singerInfoList[artSg]['downloadRec']
    collectList = collectList + singerInfoList[artSg]['colloctRec']

    i += 1
del meanPlays,stdPlays,maxPlays,artSg

#所有歌手的播放下载收藏曲线放在一起
plt.figure(figsize = (10,8), dpi = 150)
plt.plot(playList,'k')
plt.plot(downList,'b')
plt.plot(collectList,'r')
plt.title('overall playK-downB-colloctR')

#相关参数（影响结果的重要参数）
seqLength = 10                          #序列长度
testSetRate = 0                         #测试集比例
if _ISTEST == True:
    testSetRate = len(objDateL) / len(dateList)
else:
    testSetRate = 0
lenDate = len(dateList)                     #给定的数据集时间长度
nSinger = nObjSgr    #len(singerInfoList)               #艺人数量
batchSize = 50
validRate = 0.2
aveFilter = 4                           # 均值滤波长度

in_out_neurons = 3  #输入输出神经元个数
firLSTM = 35      #第一层神经元个数
secLSTM = 10      #第二层神经元个数
epochD = 600       #迭代次数

```

```
#%%对播放曲线列表 playList 进行均值滤波 及 求取标准差曲线


i = 0
while i < nSinger:
    j = i * lenDate
    fj = i * lenDate         #起点
    ej = (i + 1) * lenDate   #终点
    while j < ej:
        if j < (i * lenDate + aveFilter -1):
            avePlayList.append(np.mean(playList[fj:(j+1)]))
            varPlayList.append(np.std(playList[fj:(j+1)]))
        else:
            avePlayList.append(np.mean(playList[(j-aveFilter+1):(j+1)]))
            varPlayList.append(np.std(playList[(j-aveFilter+1):(j+1)]))
        j +=1
    i +=1

#均值滤波结果显示
i = 0
while i < nSinger:
    flagY = i % 9
    if flagY == 0:
        plt.figure(figsize = (10,8), dpi =150)
        plt.suptitle('average filter-play-originalK filterB')
    plt.subplot(3,3,flagY + 1)
    stPt = i * lenDate
    endPt = (i + 1) * lenDate
    plt.plot(playList[stPt:endPt],'k')
    plt.plot(avePlayList[stPt:endPt],'b')
    i += 1



dateSet = pd.DataFrame({"avePlay":avePlayList,"play":playList,"varPlay":varPlayList}) #全体数据集
dateSet.to_csv("originalDataSet.csv")
dateSetOrigin = deepcopy(dateSet)    # 原始数据集保存一份

# 数据预处理 去均值 方差归一 缩放到[-1 1]
#if _DEBUG == True:
#    pdb.set_trace()

#avePlayMean = dateSet['avePlay'].mean()
##downMean = dateSet['down'].mean()
#playMean = dateSet['play'].mean()
#
#dateSet['avePlay'] = dateSet['avePlay'] - avePlayMean
##dateSet['down'] = dateSet['down'] - downMean
#dateSet['play'] = dateSet['play'] - playMean
#
#avePlayStd = dateSet['avePlay'].std()
##downStd = dateSet['down'].std()
#playStd = dateSet['play'].std()
#
#dateSet['avePlay'] = dateSet['avePlay'] / avePlayStd
##dateSet['down'] = dateSet['down'] / downStd
#dateSet['play'] = dateSet['play'] / playStd
```

```
#
#factorMax = abs(dateSet).max().max() + 0.05
#
#dateSet = dateSet / factorMax
#dateSet.to_csv("preproceeDataSet.csv")


#所有歌手的播放曲线
plt.figure(figsize = (10,8), dpi = 150)
plt.plot(dateSet['play'],'k')
plt.plot(dateSet['avePlay'],'b')
plt.plot(dateSet['varPlay'],'g')
plt.xlabel('index')
plt.ylabel('playK-avePlayB')
plt.title('overall playK-avePlayB-varPlayG - preprocessed')

#%%训练集测试集划分
def load_data(data, n_prev = 14):

    docX, docY = [], []
    for i in range(len(data)-n_prev):
#        pdb.set_trace()
        docX.append(data.iloc[i:i+n_prev].as_matrix())
        docY.append(data.iloc[i+n_prev].as_matrix())
#    alsX = np.array(docX)
#    alsY = np.array(docY)

    return docX, docY

def train_test_split(df, test_size = 1 / 3, seqL = 14):

    ntrn = int(round(len(df) * (1 - test_size)))

    X_train, y_train = load_data(df.iloc[0:ntrn],seqL)
    X_test, y_test = load_data(df.iloc[ntrn:],seqL)

    return (X_train, y_train), (X_test, y_test)

# 训练集 测试集 划分
if _DEBUG == True:
    pdb.set_trace()
#初值
(xTrain,yTrain), (xTest,yTest) = train_test_split(dateSet[0:lenDate],testSetRate,seqLength)

needPredict = [] # 需要被预测的后续序列的真实值
tempIndex = int(round(lenDate * (1 - testSetRate)))
if _ISTEST == True:
    needPredict.append(dateSet[0:lenDate].iloc[tempIndex:].as_matrix()) # 三维数组，每组是一个歌手需要预测的序列

i = 1

while i < nSinger:
    startPt = i * lenDate
    endPt = (i + 1) * lenDate
    tempData = dateSet[startPt:endPt]
    (xTrainTp,yTrainTp), (xTestTp,yTestTp) = train_test_split(tempData,testSetRate,seqLength)
    xTrain = np.vstack((xTrain,xTrainTp))
```

```
    yTrain = np.vstack((yTrain,yTrainTp))
    xTest = np.vstack((xTest,xTestTp))
    yTest = np.vstack((yTest,yTestTp))

    tempIndex = int(round(len(tempData) * (1 - testSetRate)))
    if _ISTEST == True:
        needPredict.append(tempData.iloc[tempIndex:].as_matrix())

    i += 1

X_Train = np.array(xTrain)
Y_Train = np.array(yTrain)
X_Test = np.array(xTest)
Y_Test = np.array(yTest)

del xTrain, yTrain, xTest, yTest

#%%绘制需要被预测的数据之间的差异
if _ISTEST == True:
    i = 0
    plt.figure(figsize = (10,8), dpi = 150)
    while i < nSinger:
        orgValue = pd.DataFrame(needPredict[i])
        plt.plot(orgValue[1])
        i += 1
        del orgValue
    plt.suptitle('need predict test data - preprocess data')


#%% 训练算法模型
if _DEBUG == True:
    pdb.set_trace()

from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.layers.recurrent import LSTM
from keras.callbacks import EarlyStopping

model = Sequential()
# LSTM作为第一层---输入层维度：input_dim，输出层维度：hidden_neurons
model.add(LSTM(firLSTM, input_dim=in_out_neurons, input_length=seqLength,return_sequences=True))
model.add(LSTM(secLSTM,return_sequences=False))
#model.add(LSTM(thiLSTM))
# 标准的一维全连接层---输出：in_out_neurons，输入：input_dim
model.add(Dense(in_out_neurons,activation='linear'))
model.compile(loss="mse", optimizer="rmsprop") # mse  mean_squared_error
#提前中断训练
earlyStopping = EarlyStopping(monitor = 'val_loss', patience = 10)
# X_Train三维数组，每组是一个序列
hist = model.fit(X_Train, Y_Train, batch_size=batchSize, nb_epoch=epochD, verbose=0, shuffle = False,validation_split=valid
#print(hist.history)

#对训练集进行预测-调试用
predictTrain = model.predict(X_Train) # 二维数组，每一行是一组预测值
predictDF = pd.DataFrame(predictTrain)
Y_TrainDF = pd.DataFrame(Y_Train)

```

```
plt.figure(figsize = (10,8), dpi = 150)
plt.plot(list(predictDF[1]),'g')
plt.plot(list(Y_TrainDF[1]),'b')
plt.title('train set predict check')


if _DEBUG == True:
    pdb.set_trace()
 1
#%%预测
i = 0
j = 0
predictTest = {} # 所有歌手最终预测结果
while j < nSinger:
    artSg = artList[objInd[j]]
    predictTest[artSg] = {}
    predictTest[artSg]['playRec'] = []
    predictTest[artSg]['avePlay'] = []
    predictTest[artSg]['varPlay'] = []

    j += 1
del artSg

if _DEBUG == True:
    pdb.set_trace()
i = 0
j = 0
lastIndex = len(X_Train) / nSinger
while j < nSinger:
    lastData = np.array([X_Train[int(lastIndex * (j+1) -1)]])
    while i < len(objDateL):                #预测天数
        predictTp = model.predict(lastData)
        artSg = artList[objInd[j]]
        predictTest[artSg]['varPlay'].append(predictTp[0][2])
        predictTest[artSg]['playRec'].append(predictTp[0][1])
        predictTest[artSg]['avePlay'].append(predictTp[0][0])

        lastData = np.array([np.vstack((lastData[0][1:],predictTp))])
        i += 1
    j += 1
    i = 0
    del lastData, predictTp
del artSg

# 预测结果分析---数据还原之前
i = 0
xIndex = range(len(objDateL))
if _ISTEST == True:
    while i < nSinger:                          # 播放预测曲线
        flagY = i % 9
        if flagY == 0:
            plt.figure(figsize = (10,8), dpi = 150)
            plt.suptitle('test set: predict play')
        plt.subplot(3,3,flagY + 1)
        orgValue = pd.DataFrame(needPredict[i]) # needPredict三维数组，每组是一个歌手需要预测的序列值
        artSg = artList[objInd[i]]
        plt.plot(xIndex,predictTest[artSg]['playRec'],'g')
```

```
        plt.plot(xIndex,orgValue[1],'b')

        i += 1
        del orgValue
    del artSg


i = 0
if _ISTEST == True:
    while i < nSinger:                          # 平均值预测曲线
        flagY = i % 9
        if flagY == 0:
            plt.figure(figsize = (10,8), dpi = 150)
            plt.suptitle('test-predict avePlay')
        plt.subplot(3,3,flagY + 1)
        orgValue = pd.DataFrame(needPredict[i])
        artSg = artList[objInd[i]]
        plt.plot(xIndex,predictTest[artSg]['avePlay'],'g')
        plt.plot(xIndex,orgValue[0],'b')

        i += 1
        del orgValue
    del artSg


#i = 0
#while i <nSinger:                           # 收藏预测曲线
#    flagY = i % 9
#    if flagY == 0:
#        plt.figure(figsize = (10,8), dpi = 150)
#
#    plt.subplot(3,3,flagY +1)
#    orgValue = pd.DataFrame(needPredict[i])
#    plt.plot(xIndex,predictTest[artList[i]]['colloctRec'],'g')
#    plt.plot(xIndex,orgValue[0],'b')
#
#    i += 1
#    del orgValue
#plt.suptitle('test-predict colloct')

#%%预测---还原到原始数据集
if _ISTEST == True:
    i = 0
    while i < nSinger:
        flagY = i % 9
        if flagY == 0:
            plt.figure(figsize = (10,8), dpi =150)
            plt.suptitle('test-predict play- back to original')
        plt.subplot(3,3,flagY + 1)

        artSg = artList[objInd[i]]
        meanPlays = singerInfoList[artSg]['meanPlay']
        stdPlays = singerInfoList[artSg]['stdPlay']
        maxPlays = singerInfoList[artSg]['maxPlay']

        orgValue = ((pd.DataFrame(needPredict[i]))[1]) * maxPlays * stdPlays + meanPlays
        aftValue = ((pd.DataFrame(predictTest[artSg]['playRec']))[0]) * maxPlays * stdPlays + meanPlays
```

```
627
628         plt.plot(xIndex,orgValue,'b')
629         plt.plot(xIndex,aftValue,'g')
630 
631         i +=1
632 
633     del orgValue, aftValue
634     del artSg
635 
636 
637 #使用 aveplay 预测真实 play
638 if _ISTEST == True:
639     i = 0
640     while i < nSinger:
641         flagY = i % 9
642         if flagY == 0:
643             plt.figure(figsize = (10,8), dpi =150)
644             plt.suptitle('use avePlay to predict real play line')
645         plt.subplot(3,3,flagY + 1)
646 
647         artSg = artList[objInd[i]]
648         meanPlays = singerInfoList[artSg]['meanPlay']
649         stdPlays = singerInfoList[artSg]['stdPlay']
650         maxPlays = singerInfoList[artSg]['maxPlay']
651 
652         orgValue = ((pd.DataFrame(needPredict[i]))[1]) * maxPlays * stdPlays + meanPlays
653         aftValue = ((pd.DataFrame(predictTest[artSg]['avePlay']))[0]) * maxPlays * stdPlays + meanPlays
654 
655         plt.plot(xIndex,orgValue,'b')
656         plt.plot(xIndex,aftValue,'g')
657 
658         i +=1
659 
660         del orgValue, aftValue
661     del artSg
662 
663 #%%融合svr
664 svrResult = {}
665 fileSVR = open("svr.csv")
666 svrData = fileSVR.readlines()
667 
668 for svrInfo in svrData:
669     svrInfo = svrInfo.replace('\n','')
670     arrInfo = svrInfo.split(',')
671 
672     svrResult[arrInfo[0]] = int(arrInfo[1])
673 
674 
675 fileSVR.close()
676 del svrData,svrInfo,arrInfo
677 
678 #%% 评价指标
679 
680 if _ISTEST == True:
681     singerF = [] # 每个歌手的评价指标值 F
682     sumF = 0
683     i = 0
```

```
    while i < nSinger:
        artSg = artList[objInd[i]]
        meanPlays = singerInfoList[artSg]['meanPlay']
        stdPlays = singerInfoList[artSg]['stdPlay']
        maxPlays = singerInfoList[artSg]['maxPlay']

        orgValue = ((pd.DataFrame(needPredict[i]))[1]) * maxPlays * stdPlays + meanPlays
        aftValue = ((pd.DataFrame(predictTest[artSg]['playRec']))[0]) * maxPlays * stdPlays + meanPlays
 1
        tempArr = (np.array(aftValue) - np.array(orgValue)) / (np.array(orgValue))
        tempS = ((tempArr * tempArr).sum()) / len(objDateL)
        theta = math.sqrt(tempS)

        tempFi = math.sqrt((np.array(orgValue)).sum())
        sumF = sumF + (1-theta) * tempFi

        singerF.append((1-theta) * tempFi)

        i += 1
        del orgValue,aftValue,tempArr
    del artSg

if _ISTEST == True:
    singerFA = [] # 每个歌手的评价指标值 F
    sumF = 0
    i = 0
    while i < nSinger:
        artSg = artList[objInd[i]]
        meanPlays = singerInfoList[artSg]['meanPlay']
        stdPlays = singerInfoList[artSg]['stdPlay']
        maxPlays = singerInfoList[artSg]['maxPlay']

        orgValue = ((pd.DataFrame(needPredict[i]))[1]) * maxPlays * stdPlays + meanPlays
        aftValue = (((pd.DataFrame(predictTest[artSg]['playRec']))[0]) * maxPlays * stdPlays + meanPlays) * 0.5 + svrResult[artS

        tempArr = (np.array(aftValue) - np.array(orgValue)) / (np.array(orgValue))
        tempS = ((tempArr * tempArr).sum()) / len(objDateL)
        theta = math.sqrt(tempS)

        tempFi = math.sqrt((np.array(orgValue)).sum())
        sumF = sumF + (1-theta) * tempFi

        singerFA.append((1-theta) * tempFi)

        i += 1
        del orgValue,aftValue,tempArr
    del artSg
#    resF = pd.DataFrame({"singerf":singerF})
#    resF.to_csv("singerF.csv")


#%%使用均值预测后的评价指标值
#singerF_AVG = [] # 每个歌手的评价指标值 F
#sumF = 0
#i = 0
#while i < nSinger:
#    meanPlays = singerInfoList[artList[i]]['meanPlay']
```

```
#    stdPlays = singerInfoList[artList[i]]['stdPlay']
#    maxPlays = singerInfoList[artList[i]]['maxPlay']
#
#    orgValue = ((pd.DataFrame(needPredict[i]))[1]) * maxPlays * stdPlays + meanPlays
#    aftValue = ((pd.DataFrame(predictTest[artList[i]]['avePlay']))[0]) * maxPlays * stdPlays + meanPlays
#
#    tempArr = (np.array(aftValue) - np.array(orgValue)) / (np.array(orgValue))
#    tempS = ((tempArr * tempArr).sum()) / len(objDateL)
#    theta = math.sqrt(tempS)
#
#    tempFi = math.sqrt((np.array(orgValue)).sum())
#    sumF = sumF + (1-theta) * tempFi
#
#    singerF_AVG.append((1-theta) * tempFi)
#
#    j += 1
#    del orgValue,aftValue,tempArr
#sum(singerF_AVG[:36]) + sum(singerF_AVG[37:56]) + sum(singerF_AVG[57:])

#%%写入到预测文件
if _ISTEST == False:
    import csv
    resFile = open("mars_tianchi_artist_plays_predict.csv","wb")
    writerRes = csv.writer(resFile)

    i = 0
    j = 1
    while i < nSinger:
        artSg = artList[objInd[i]]
        meanPlays = singerInfoList[artSg]['meanPlay']
        stdPlays = singerInfoList[artSg]['stdPlay']
        maxPlays = singerInfoList[artSg]['maxPlay']

        aftValue = (((pd.DataFrame(predictTest[artSg]['playRec']))[0]) * maxPlays * stdPlays + meanPlays) * 0.5 + svrResult[art
        while j < len(objDateL):
            oneLineData = [artSg,str(int(aftValue[j])),objDateL[j]]
            writerRes.writerow(oneLineData)

            del oneLineData
            j += 1
        del aftValue
        j = 1
        i += 1
    resFile.close()
    del artSg
```

四、参考文献

1.LSTM入门介绍比较好的文章：A Critical review of rnn for sequence learning
2.LSTM学习思路，参见知乎的一个介绍，很详细：https://www.zhihu.com/question/29411132 (https://www.zhihu.com/question/29411132) 。
3.Python入门视频教程—可看南京大学张莉老师在coursera上的公开课《用Python玩转数据》，有例子介绍，很实用。https://www.coursera.org/learn/hipython/home/welcome (https://www.coursera.org/learn/hipython/home/welcome)。

4.Keras介绍—参看官方文档http://keras.io/ (http://keras.io/)

nuannuanyingying (/nuannuanyingying) 2017-07-01 16:21 1楼

(/nuannuanyingying)

学有成，学有所用！

回复

查看 4 条热评

相关文章推荐

tensorflow笔记：多层LSTM代码分析 (http://blog.csdn.net/u014595019/article/details/52…

tensorflow笔记系列：（一） tensorflow笔记：流程，概念和简单代码注释（二） tensorflow笔记：多层CNN代码分析之前讲过了tensorflow中CNN的示例代码，...

u014595019 (http://blog.csdn.net/u014595019) 2016年10月08日 17:33 50804

LSTM源码分析 (http://blog.csdn.net/u011274209/article/details/53329082)

代码来源：LSTM Networks for Sentiment Analysis ps：markdown真难用啊

u011274209 (http://blog.csdn.net/u011274209) 2016年11月25日 01:09 1745

LSTM推导 源码分析 (http://blog.csdn.net/vsooda/article/details/47958709)

LSTM推导 源码分析

vsooda (http://blog.csdn.net/vsooda) 2015年08月24日 21:36 5586

在keras 上实践,通过keras例子来理解lastm循环神经网络 (http://blog.csdn.net/ma41653…

本文是对这篇博文的翻译和实践： http://machinelearningmastery.com/understanding-stateful-lstm-recurrent-neural-netw...

ma416539432 (http://blog.csdn.net/ma416539432) 2016年12月07日 19:05 7085

详细解读简单的lstm的实例 (http://blog.csdn.net/zjm750617105/article/details/51321889)

本文是初学keras这两天来，自己仿照addition_rnn.py，写的一个实例，数据处理稍微有些不同，但是准确性相比addition_rnn.py 差一点，下面直接贴代码，解释和注释都在代码里...

zjm750617105 (http://blog.csdn.net/zjm750617105) 2016年05月05日 15:31 11451

深度学习**Keras** 库 跑例子 **(http://blog.csdn.net/u014114990/article/details/49765725)**

跑lmdb_lstm.py 因为需要用lstm，所以就先跑 lstm例子， 1、官网下载后，直接运行lmdb_lstm.py。总是提示无法下载，打开程序有看到， 通过load_data来下载数据...

u014114990 (http://blog.csdn.net/u014114990) 2015年11月10日 21:37 18468

如何为**LSTM**重新构建输入数据（**Keras**） **(http://blog.csdn.net/CygqjBABx875u/article/…**

“全球人工智能拥有十多万AI产业用户，10000多名AI技术专家。主要来自：北大，清华，中科院，麻省理工，卡内基梅隆，斯坦福，哈佛，牛津，剑桥...以及谷歌，腾讯，百度，脸谱，微软，阿里，海康威视，...

CygqjBABx875u (http://blog.csdn.net/CygqjBABx875u) 2017年10月14日 00:00 111

keras的一些例子理解 **(http://blog.csdn.net/B_C_Wang/article/details/74885654)**

keras的一些例子理解来自我的github页面：https://github.com/B-C-WANG/AI.Learning/tree/master/AI.Learning.Notes.III....

B_C_Wang (http://blog.csdn.net/B_C_Wang) 2017年07月09日 15:00 644

LSTM实例 **(http://blog.csdn.net/qq_34484472/article/details/77371848)**

LSTM网络（长短期记忆网络）可以理解为是RNN的改进版，它的出现解决了RNN的记忆问题。本博将给出lstm的实现代码，并做出简要讲解。...

qq_34484472 (http://blog.csdn.net/qq_34484472) 2017年08月18日 16:34 579

利用 **Keras** 下的 **LSTM** 进行情感分析 **(http://blog.csdn.net/William_2015/article/details/7…**

~~~~~~~我们用 Keras 提供的 LSTM 层构造和训练一个 many-to-one 的 RNN。 网络的输入是一句话，输出是一个情感值（积极或消极）。 所用数据是来自 Kaggle 的情感分...

William_2015 (http://blog.csdn.net/William_2015) 2017年06月10日 10:35 2968

Keras之**LSTM**源码阅读笔记 **(/silent56_th/article/details/73442391)**

这里目前为止只是博主阅读Keras中LSTM源码的草稿笔记，内容不全，没有清晰的逻辑，只是堆砌个人想法。参考文献： 1. keras的官方相关文档 2. LSTM原论文 3. keras的RNN...

silent56_th (http://blog.csdn.net/silent56_th) 2017-06-18 23:45 1013

keras + lstm 情感分类 **(/weixin_36541072/article/details/53786020)**

负面评论如下：正面评论如下：使用keras配合lstm效果不错。代码：#coding:utf-8 ''' Created on 2016-12-20@author: 刘帅 ''' impo...

weixin_36541072 (http://blog.csdn.net/weixin_36541072) 2016-12-21 15:50 3445

Keras框架下**LSTM**的一种实现 **(/jiehanwang/article/details/48680327)**

第一次发帖，做个测试，以后逐一添上细节和代码。目标：实现以Kinect为sensor的连续手语手别特性：使用非常简单用GPU能加速4倍以上，但是散热需要“格力”牌电风扇，没错。Stack...

jiehanwang (http://blog.csdn.net/jiehanwang) 2015-09-23 14:57 8010
~~~~~~~

LSTM实现详解 **(/appleml/article/details/49923577)**

原文地址：http://www.csdn.net/article/2015-09-14/2825693 英文地址：http://apaszke.github.io/lstm-explained....

u014221266 (http://blog.csdn.net/u014221266) 2015-11-19 09:55 3672

一个基于**TensorFlow**的简单故事生成案例：带你了解**LSTM (/jdbc/article/details/717749...**

在深度学习中，循环神经网络（RNN）是一系列善于从序列数据中学习的神经网络。由于对长期依赖问题的鲁棒性，长短期记忆（LSTM）是一类已经有实际应用的循环神经网络。

jdbc (http://blog.csdn.net/jdbc) 2017-05-13 07:21 1789

LSTM实现详解 **(/zdy0_2004/article/details/49977423)**

http://datauml.org/20778.html 相关阅读：深入浅出LSTM神经网络前言在很长一段时间里，我一直忙于寻找一个实现LSTM网络的好教程。它们似乎很复杂，而...

zdy0_2004 (http://blog.csdn.net/zdy0_2004) 2015-11-22 14:13 21445

递归神经网络**LSTM**原理——结合实例**MATLAB**实现 **(/u010540396/article/details/52797489)**

最近正在看递归神经网络，看了网上很多博文，算是鱼龙混杂，并且基本都是使用Python实现，要不就是使用Matlab中的函数库等。对于使用Matlab的同学，甚为不方便。所以我将结合实例，使用matla...

u010540396 (http://blog.csdn.net/u010540396) 2016-10-12 15:12 7887

用 **LSTM** 做时间序列预测的一个小例子 **(/aliceyangxi1987/article/details/73420583)**

问题：航班乘客预测数据：1949 到 1960 一共 12 年，每年 12 个月的数据，一共 144 个数据，单位是 1000 下载地址目标：预测国际航班未来 1 个月的乘客数import ...

aliceyangxi1987 (http://blog.csdn.net/aliceyangxi1987) 2017-06-18 12:16 2722

LSTM实现详解 **(/real_myth/article/details/51275869)**

LSTM实现详解发表于2015-09-14 16:58| 5021次阅读| 来源Apaszke Github| 3 条评论| 作者Adam Paszke LSTM神经网络RN...

Real_Myth (http://blog.csdn.net/Real_Myth) 2016-04-28 16:58 2421

深度学习与自然语言处理之五：从**RNN**到**LSTM (/malefactor/article/details/50436735)**

本文介绍了RNN和LSTM的基本技术原理及其在自然语言处理的应用。

malefactor (http://blog.csdn.net/malefactor) 2015-12-30 19:01 37614

内容举报

返回顶部