数据分析导论

课程论文

基于 LSTM 的共享单车集群短时需求预测

学生姓名: 纪斌斌

学号: 19124418

联系方式: 18725015414

导师姓名: 陈娟

提交日期: 2021 年 11 月 19 日

《 数据分析导论 》课程论文评价表（2021 版）

评审项目	权重	评分标准		得分（百分制）
一、问题描述（A）	10%	90 分以上	选题有重要的理论意义或实用价值	
		75 ~ 89 分	选题有较大的理论意义或实用价值	
		60 ~ 74 分	选题有一定的理论意义或实用价值	
		60 分以下	选题缺乏理论意义或实用价值	
二、数据预处理（B）	30%	90 分以上	数据预处理丰富、全面	
		75 ~ 89 分	数据预处理较丰富、较详细	
		60 ~ 74 分	数据预处理简单	
		60 分以下	阅读量粗糙	
三、理论分析（C）	20%	90 分以上	掌握坚实宽广的理论基础和系统知识	
		75 ~ 89 分	较好地掌握理论基础和系统知识	
		60 ~ 74 分	基本掌握理论基础和系统知识	
		60 分以下	掌握理论基础和系统知识较差	
四、实验结果（D）	20%	90 分以上	实验方案丰富，详细，结果合理	
		75 ~ 89 分	实验方案较丰富，较详细，结果较合理	
		60 ~ 74 分	实验方案较简单，结果一般	
		60 分以下	实验方案粗糙，结果不好	
五、研究成果的创新性（D）	10%	90 分以上	研究成果具有很强的创新性	
		75 ~ 89 分	研究成果具有较强的创新性	
		60 ~ 74 分	研究成果创新性不明显	
		60 分以下	研究成果不具有创新性	
六、写作能力（E）	10%	90 分以上	条理清晰，分析严谨，文笔流畅	
		75 ~ 89 分	条理较好，层次分明，文笔顺通	
		60 ~ 74 分	写作能力尚可	
		60 分以下	写作能力较差	
总分		0.1A+0.3B+0.2C+0.2D++0.1E+0.1F		

目录

1 问题描述 4
1.1 研究背景和意义 4
1.2 目前研究方法综述及问题分析 4
2 数据集介绍 5
2.1 数据预处理方法 6
2.2 数据可视化 7
2.3 特征工程 8
3 方法描述 10
3.1 本研究模型描述 10
3.2 框架描述 10
3.3 评价指标 11
4 模型结果分析 11
4.1 传统机器学习模型和集成学习模型 12
4.1.1 特征构建 12
4.1.2 模型选取及超参数设置 12
4.2 LSTM 12
4.2.1 特征构建 12
4.2.2 模型调参及参数设置 13
4.3 模型结果评价及可视化 14
5 结论 15
6 参考文献 15
7 代码 16
7.1 过程文件 16
7.2 预处理代码 16
7.3 LSTM 代码 23
7.4 传统机器学习代码 28

1 问题描述

1.1 研究背景和意义

随着共享经济的发展，共享经济成为当前的热门话题，其中，共享经济在交通领域的代表就是共享单车，共享单车的出现解决了城市中“最后一公里”的问题。作为更接近点→点的出行方式，自行车交通相比步行方式更加快捷，相比公交、地铁更加灵活，相比机动车出行更加低碳环保。因此，共享单车的出现降低了机动车出行率、出行成本，增强了公共交通微循环并提高市民环保出行意识[1]。

但同时也随之出现了一些问题，黄斌等[2]发现公共自行车系统最突出的问题在于：（1）租赁点无车可借（21%）；（2）租赁点还不进车（15%）；（3）高峰时段借不到车（14%）；（4）自行车损坏难以使用（12%）；（5）服务热线打不通（10%）。其中，第一位和第三位有问题重复的嫌疑存在。但是仍然可以看出公自系统中最亟待解决的问题：借车难、还车难。共享自行车系统的借、还问题需要合理的调配来解决，而需求预测是解决调度问题的第一步，故本文以 24 小时内的有桩共享自行车站点需求预测展开研究。

1.2 目前研究方法综述及问题分析

国内外学者基于机器学习和深度学习理论在公共自行车需求预测方面展开了多项研究。

付俐哲[3] 提出了一种结合 ST-DBSCAN 与 K-means 的时空聚类方法，能够很好的确定单车停靠站点的容量、站点质心的初始位置，以及站点的个数。并通过 LSTM 模型兼顾工作日与非工作日、天气状况、时间段以及各停放点的地理信息等多种影响因素进行需求预测。

姜剑等[4]基于 GeoHash 对共享单车站点进行划分，提出了基于堆叠集成方法的极限梯度提升（XGBoost）流量变化量预测的机器学习模型 SMVP。并通过结合长短期记忆网络（LSTM）、卷积神经网络（CNN）、全连接神经网络（FC）等多种神经网络的深度学习模型 DPNNst 来预测用户的出行的最终目的地，较好地提取了共享单车轨迹数据的时间特征和空间特征。

胡冰华[5] 提出了一种基于残差分析的时空数据预测模型（RAM-TF）用于交通流量预测。将时空数据分类采集并打包成临近性时空数据、周期性时空和趋势性时空数据三部分用于提取时空数据的时间特征，打包好的数据分别用卷积神经网络（CNN）进行处理以提取时空数据集的空间特征。经过初步卷积后的时空数据通过矩阵相加进行融合（Fusion），简化模型，并使用残差网络（ResNet）进行优化。对时空特征进行提取，并优化了网络结构。

施行建教授[6]提出了一种 Convolutional LSTM 模型，一般的 LSTM 用全连接来作为不同状态之间的转换，而 ConvLSTM 不是使用全连接而是卷积。Bao[7]等将

ConvLSTM模型应用于公共自行车需求预测，将数据栅格化，利用LSTM和ConvLSTM的混合深度学习神经网络来预测每一个区域的共享单车的短时流量，使用 LSTM 来获取该区域的时间特征，使用 ConvLSTM 来获取共享单车影响因素，如天气、特征。该模型对早高峰时刻的需求预测性能较好。

综上，传统的方法虽然能对连续时间序列的交通流量进行预测，但大多数交通流量数据同时具有时间和空间两种特性，其中空间特性一般机器学习或基于时间序列的方法很难做到特征提取，并且有些模型前期工作还需要做大量的特征工程，故具有局限性。同时，提出的基于时间和空间的神经网络模型一定程度上解决了这部分问题，但也伴随着由于网络层数过多所带来的训练精度不稳定问题。

2 数据集介绍

本实验数据集选自洛杉矶城市的地铁自行车共享系统。使用其自 2018 年 1 月至 2018 年 9 月三个季度的自行车出行数据，共计 238030 条出行数据，涉及 128 个站点。用户出行数据如表 2-1 所示，站点数据如表 2-2 所示：

表 2-1 用户数据信息

	Columns	Description	Values
1	trip_id	出行编码	112536773
2	duration	行程时长	7
3	start_time	开始时间	2019-01-01 00:07:00
4	end_time	结束时间	2019-01-01 00:14:00
5	start_station	起始站点	3046
6	start_lat	起始站点经度	34.049198
7	start_lon	起始站点维度	-118.252831
8	end_station	结束站点	3051
9	end_lat	结束站点经度	34.043732
10	end_lon	结束站点维度	-118.260139
11	bike_id	单车编码	6468
12	passholder_type	用户类型	1/30/365
13	trip_route_category	行程类型	One Way/ Round Trip
14	bike_type	单车种类	Round Trip/ electric

表 2-2 站点数据信息

	Columns	Description	Values
1	Station_ID	站点编号	3005
2	Station_Name	站点名称	Virtual Station
3	Day of Go_live_date	创建日期	7/7/2016
4	Region	所属地区	DTLA/ Port of LA
5	Status	站点状态	Active/ Inactive

2.1 数据预处理方法

(1)剔除数据异常值。包括①短时间内在同一站点借还车，认为其属于误借等非正常需求。②借车换车点位虚拟站点，由于借还点 GPS 信号若或借还点在洛杉矶之外。③骑行时间位于总时间的六倍四分位距外，认为骑行时间过长或过短属于异常情况。

(2)确定预测粒度为小时级，将时间数据转换为精确到小时级的数据,并生成站点与时刻的对应需求矩阵，矩阵大小为 128×6552(24*273)维。$C_{i,j}$代表站点 i 在 j 时刻的需求量，此处需求量定义为站点的借出量。矩阵格式如表 2-3 所示：

表 2-3 站点-时间的对应需求矩阵

	2018-01-01 00:00:00	……	2018-09-30 22:00:00	2018-09-30 23:00:00
3005	4		0	2
…	…	……	…	…
4267	0		2	1

(3)生成站点间的转换矩阵，矩阵为 128*128 的方阵。$C_{i,j}$代表从站点 i 借出到站点 j 换车的数量，矩阵格式如表 2-4 所示：

表 2-4 站点间的转换矩阵

	3005	3006	……	4254	4267
3005	0	2	……	…	4
…	…	…	……	…	…
4267	5	3	……	…	0

(4)生成站点相似度矩阵，其中$S_{i,j}$表示站点 i 与站点 j 之间的相似度，数值越大表示越不相似，对角线是当 i=j，表示同一个站点，他们之间的相似度距离为 0。$S_{i,j}$由地理位置和租还关系两部分组成，站点间的距离使用 haversine 公式(2.1)计算：

$$d = 2r\arcsin\left(\sqrt{\mathrm{hav}(\varphi_2-\varphi_1)+\cos(\varphi_1)\cos(\varphi_2)\,\mathrm{hav}(\lambda_2-\lambda_1)}\right)$$
$$= 2r\arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2-\varphi_1}{2}\right)+\cos(\varphi_1)\cos(\varphi_2)\sin^2\left(\frac{\lambda_2-\lambda_1}{2}\right)}\right)$$

公式(2.1)

站点之间的租还关系，通过计算在所有数据集上单个站点 i 到其他站点 j 的租还次数$C_{i,j}$,然后根据式(2.2)计算出站点之间的租还关系$p_{i,j}$，值越大，代表关系越近：

$$p_{ij} = \frac{c_{ij}}{\sum_{j=1}^{n} c_{ij}}$$

公式(2.2)

最终矩阵中的每个元素$S_{i,j}$是由式(3.3)计算出来的，其中 α 是一个参数，是用来控制$p_{i,j}$的权重

$$S_{i,j} = d_{i,j}(1-\alpha p_{i,j})$$

公式(2.3)

矩阵格式如表 2-5 所示：

表 2-5 站点相似度矩阵

	3005	3006	……	4254	4267
3005	0	0.003574	……	0.032228	0.030905
…	…	…	……	…	…
4267	0.030899	0.029773	……	0.028509	0

2.2 数据可视化

(1)站点分布，如图 2-1 观察站点分布，站点主要聚集在 4 个区域。

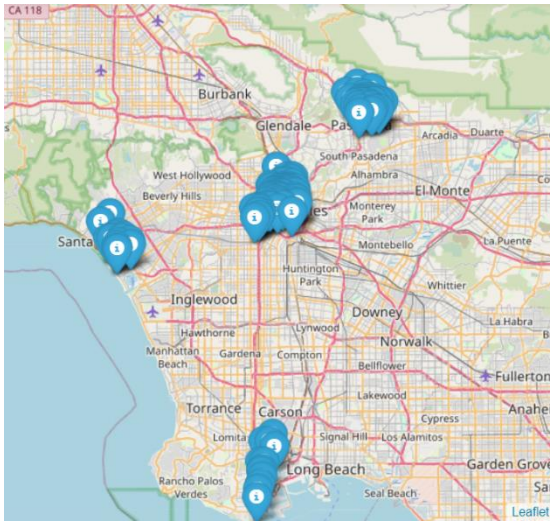


图 2-1 站点分布

(2)自行车需求量随时间的变化

图 2-2 计算的是一天内每个小时的共享自行车流量，可以看出在 17 点左右存在明显的晚高峰，7 点左右存在小幅的早高峰，20 点之后需求量大幅下降。图 2-3 计算的是一周内每天的共享自行车流量，可以看出一周内共享单车的需求量没有太大幅度的波动。

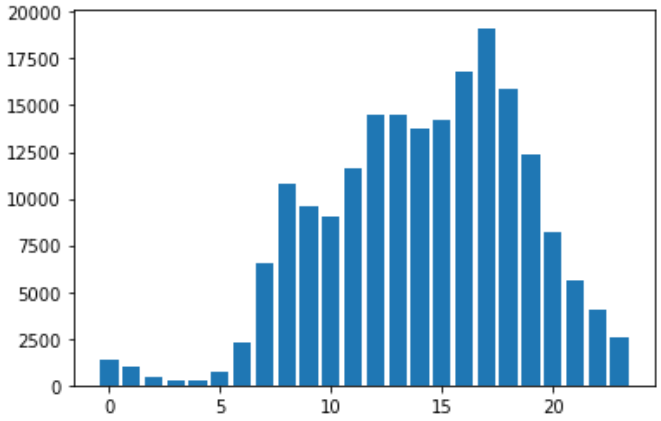


图 2-2 自行车一天的流量变化量分布规律

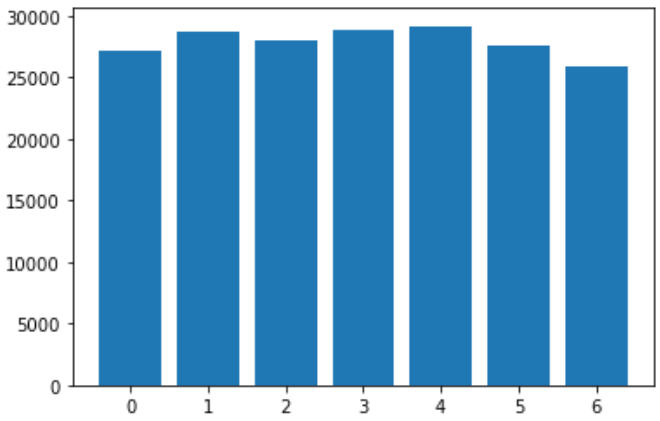


图 2-3 自行车一周内的流量变化规律

2.3 特征工程

(1)使用 K-Medoids 进行站点相似性聚类，K-Medoids 聚类算法是一个与 K-Means 聚类算法相关的算法，K-Means 算法和 K-Medoids 算法都是将数据集中的点分成多个类，试图减少每个类中点之间的距离。但是相比 K-Means 算法而言，K-Medoids 算法是选择数据集中的点作为聚类中心，而不是多个点的均值，这样，在算法执行的过程中，每轮迭代计算的都是每个站点到聚类中心站点之间的距离，这样在 K-Medoids 算法计算中心点 i 和 j 的距离的时候，可以直接读取表 2-5 所计算好的相似度距离矩阵。聚类流程见表 2-6，聚类后的结果如图 2-4 所示：

表 2-6 K-Medoids 算法流程

输入：K，站点聚类数量，n 各站点

输出：站点聚类结果，聚类中心站点

1. 初始化：选择 n 个站点作为 K 个中心站点
2. 计算距离每个站最近的中心站点
3. **while** cost 降低 **do**:

    For 对于每个中心站点 m,每个非中心站点 o **do**:

        交换 m 和 o，重新计算 cost

        If 总的 cost 增加 **then**:

            撤销交换

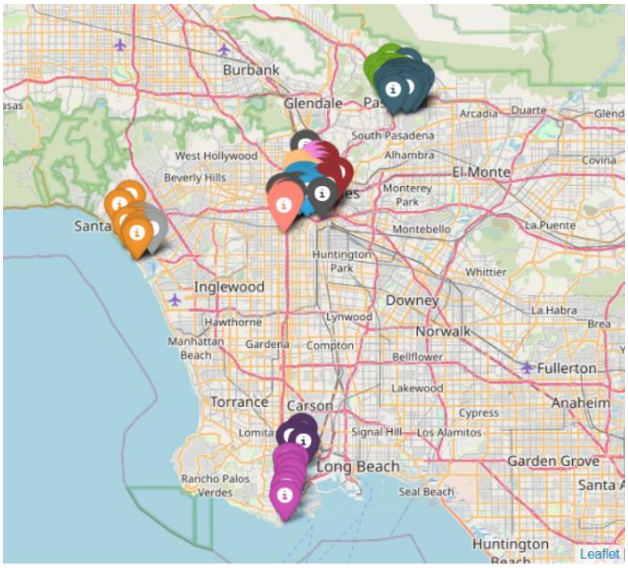


图 2-4 根据相似度矩阵进行聚类

(2)根据聚类结果统计各类别站点在各时间段内的总需求量，并将时间维度细分为月、日、星期、周 4 个维度。如表 2-7 所示：

表 2-7 聚类结果统计

	cluster	month	day	weekday	hour	demands
0	0	1	1	1	0	2
1	0	1	1	1	1	1
…	…	…	…	…	…	…
52414	7	9	30	7	22	0
52415	7	9	30	7	23	0

(3)对数据进行平滑处理，以小时为单位，对所有观测日期的同一时刻进行异常值剔除和平滑处理。主要采取了 1.5 倍四分位距的异常值剔除和均值填补。并将部分中途出现弃用的集群删除，处理前后对比如图 2-5，2-6 所示：

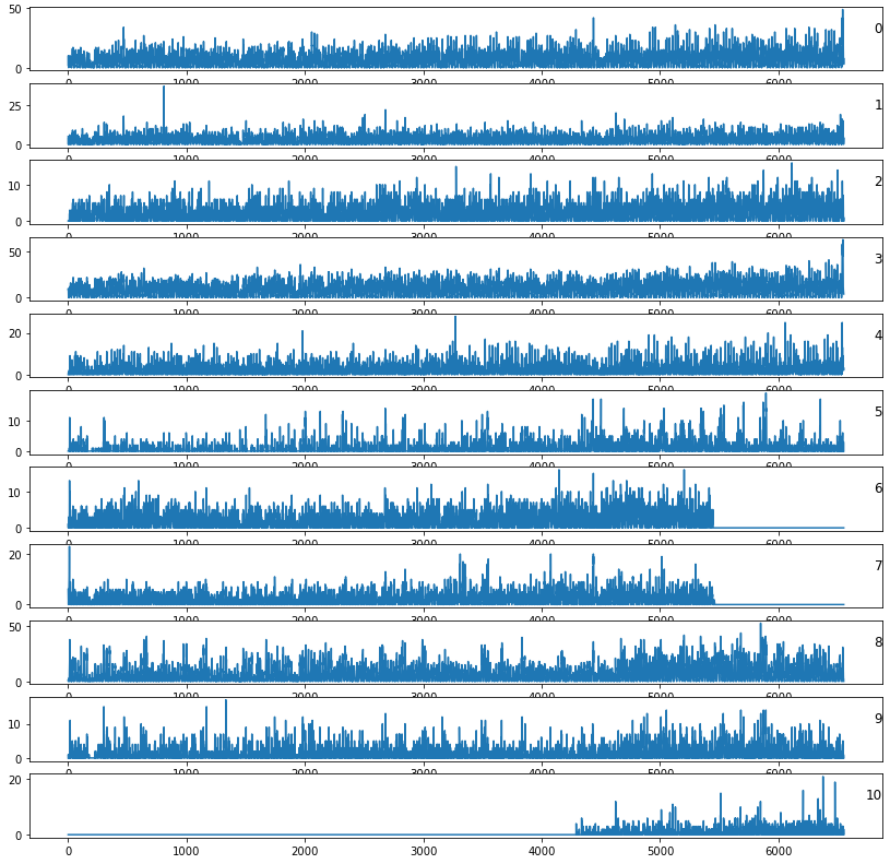


图 2-5 平滑处理前各集群需求量时间序列

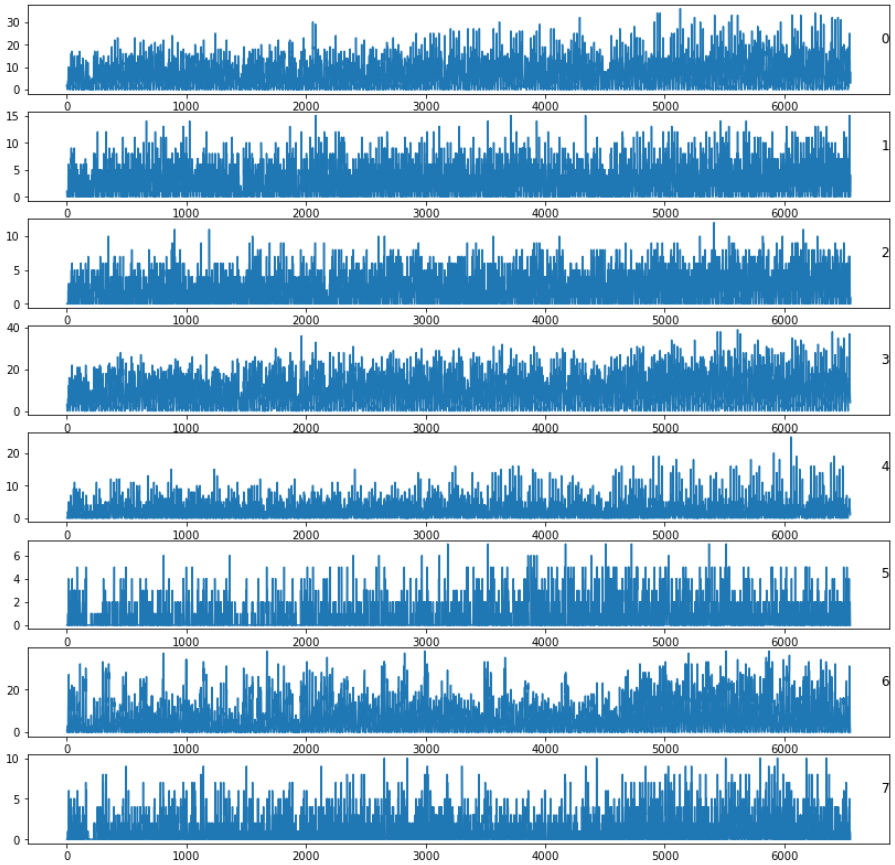


图 2-5 平滑处理后各集群需求量时间序列

3 方法描述

3.1 本研究模型描述

本文主要研究了长短期神经网络模型(LSTM)在处理时空数据方面的方法。

3.2 框架描述

RNN 在处理时间序列问题时，如果节点之间的时间间隔较远，则会带来一些序列的问题，在计算这些时间间隔较远的节点间的对应关系时会因为矩阵的多次相乘造成梯度消失或梯度膨胀爆炸的问题。LSTM 可以解决传统 RNN 的长期依赖问题，即解决当前系统状态可能会受很长时间之前系统状态影响的问题。它对传统 RNN 的隐藏层进行了结构上的改进。

LSTM 和传统 RNN 总体结构类似，内部的具体某些结构有所区别，LSTM 对上一环节信息的记忆。LSTM 的第一步是决定哪些信息是要从单元状态中被丢弃的，这是由 LSTM 中称为“遗忘门”的激活函数层决定。

LSTM 的遗忘门，首先输入的是上一个训练过程中隐藏层的信息，然后输入本过程的要输入的信息，利用选定的激活函数处理，然后就能够得到遗忘门的最终输出f_i。假如选择 sigmoid 函数作为激活函数。由于 sigmoid 的输出f_i是在[0,1]之间，当其取值为 0 时，表示该段信息需要全部丢弃；其取值为 1 时，则表示所有的信息都要完全的保留下来，所以当激活函数选取为 sigmoid 函数时，输出f_i就可以认为是遗忘层对上一层中输入到本层中的信息的遗忘概率。依照公式(3.1)进行计算：

$$f_i = \sigma(W_f \cdot [h_{t-1}, x_i] + b_f) \quad \text{公式(3.1)}$$

输入门(input gate)：输入门所做的工作主要为对当前时刻输入的信息进行一定的处理，输入门由两个部分组成，一是使用 sigmoid 激活函数后的输出i_t，二是使用 tanh 激活函数后的C_t输出，其数学表达式为：

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_i] + b_i) \quad \text{公式(3.2)}$$

$$C_t = tanh(W_i \cdot [h_{t-1}, x_t] + b_c) \quad \text{公式(3.3)}$$

然后是 LSTM 的输出门，最终的状态C_t，是由前面的遗忘门和输入门共同作用后得到的，因此最终的状态C_t，可以理解为是由上一时刻中的状态信息C_{t-1}和遗忘门的输出f_t；的乘积以及输入门得到的i_t和C_t的乘积组成的，经过它们的共同作用，将状态进一步更新为C_t，具体的公式表达为：

$$C_t = f_i * C_{t-1} + i_t * C_t \quad \text{公式(3.4)}$$

最后得到的就是模型的最终输出，其中隐藏状态中的$\mathrm{h_t}$更新是由上一过程中经过 sigmoid 激活的隐藏状态$\mathrm{h_{t-1}}$和本时刻的输入数据x_t，共同作用得到的o_t与经过 tanh 激活函数的激活的隐藏状态C_t，相乘得到的，其公式表达为：

$$o_i = \sigma(W_o \cdot [h_{t-1}, x_i] + b_o) \quad \text{公式(3.5)}$$

$$h_i = o_i * tanh(C_t) \quad \text{公式(3.6)}$$

3.3 评价指标

本研究中的评价指标均基于真实值，即归一前的真实值和反归一后的预测值展开。

(1)均方误差（MSE）

均方误差是指参数的估计值与参数的真实值之差的平方的期望。MSE 可以评价数据的变化程度，MSE 越小，说明模型的拟合实验数据能力强。MSE 的计算公式（3.7）如下。

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y_i})^2 \qquad \text{公式(3.7)}$$

(2)平均绝对误差（MAE）

平均绝对误差表示预测值和观测值之间绝对误差的平均值。只衡量了预测值误差的平均模长，而不考虑方向，取值范围也是从 0 到正无穷，MSE 越小，说明模型的拟合实验数据能力强。MAE 的计算公式（3.8）如下。

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|(y_i - \bar{y_i})| \qquad \text{公式(3.8)}$$

4 模型结果分析

本文研究问题为共享自行车集群短时需求预测，以 2018 年 1 月 1 日至 2018 年 9 月 29 日的数据预测各个集群在 2018 年 9 月 30 日 24 小时内的需求量，9 月 30 日真实需求量如图 4-1 所示：

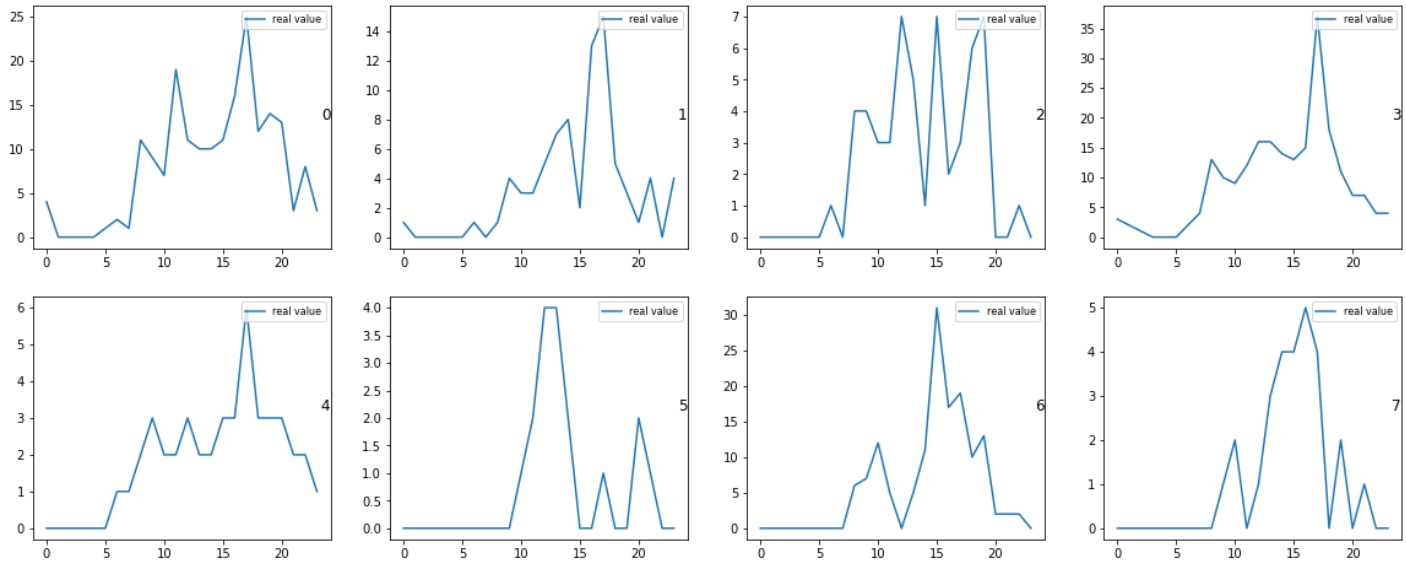


图 4-1 9 月 30 日各集群需求真实量

对实验数据采取 MinMaxScaler 的归一化方法，计算公式（4.1）如下：

$$X_{std} = \frac{X - X_{\min}}{X_{\max} - X_{min}}(axis = 0) \qquad \text{公式（4.1）}$$

4.1 传统机器学习模型和集成学习模型

4.1.1 特征构建

时间特征：从时间方面来看，每天不同时间段的公共自行车流量均存在一定的差异，但又存在一定的规律，因此首先构造特征$\boldsymbol{x_1}$为月，$\boldsymbol{x_2}$为日，$\boldsymbol{x_3}$为星期，$\boldsymbol{x_4}$为小时。

空间/聚类特征：根据第 2.3 部分对站点进行聚类，将结果进行标签化编码，构造特征为聚类特征$\boldsymbol{x_5}$。

4.1.2 模型选取及超参数设置

使用 K 邻近，决策回归树，梯度提升决策树，随机森林，xgboost。使用网格搜索确实超参数，各模型超参数构建见表 4-1

表 4-1 模型超参数信息

模型	参数	设定值
KNN	n_neighbors	10
决策回归树	max_depth	6
xgboost	n_estimators	80
GBDT	n_estimators	50
随机森林	n_estimators	80

4.2 LSTM

4.2.1 特征构建

(1) 构建时间窗，构建取值间隔 gap：①根据时间窗口所设定长度每间隔 24 小时取一条数据（即 gap 为 24），即每一条重建的 LSTM 数据集都为不同日期在同一小时下的需求量。如表 4-2 所示，例如若时间窗取值为 30，在该时间窗口的设定下，第一条数据的序列为不同集群在 1 月 1 日 0 点，1 月 2 日 0 点…1 月 30 日 0 点的需求量，标签为 1 月 31 日 0 点的需求量。②构建时间窗时无间断取值（即 gap=1），即每一条重建的 LSTM 数据集都为小时级连续的需求量，如表 4-3 所示。具体间隔 gap 的选择不同序列依据实验结果而定。

表 4-2 构建时间窗时每 24 小时取一条数据（gap=24）（以窗口大小为 30 为例）

	time	集群 1	集群 2	…	集群 n
序列	**2018-01-01 00:00:00**	2	1	…	3
	2018-01-02 00:00:00	1	0	…	1
	……	…	…	…	…
	2018-01-30 00:00:00	2	1	…	0
标签	**2018-01-31 00:00:00**	1	1	…	0

表 4-3 构建时间窗时无间断取值（gap=1）（以窗口大小为 30 为例）

	time	集群 1	集群 2	…	集群 n
序列	2018-01-01 00:00:00	2	1	…	3
	2018-01-01 01:00:00	0	1	…	1
	……	…	…	…	…
	2018-01-02 06:00:00	2	1	…	0
标签	2018-01-02 07:00:00	5	8	…	4

(2) 构建时间相关协变量，构造特征$\boldsymbol{x_1}$为月，$\boldsymbol{x_2}$为日，$\boldsymbol{x_3}$为星期，$\boldsymbol{x_4}$为小时 4 个变量作为协变量输入，参与计算但不作为输出。值得注意的是当选择协变量输入时，应将网络的 **input_size** 在原值基础上加 4，**input_size** 维持不变。具体是否选择该协变量依据不同序列实验结果而定。

(3) 使用 Xavier 初始化网络权重，本研究中的激活函数为 **tanh**，相比 mingwei 初始化，更适宜使用 Xavier 初始化。保证输出和输入尽可能地服从相同的概率分布，输出和输入具有相同的均值和误差，还需考虑反向传播时的情形，反向传播是正向传播的逆过程，此时的输入是前向传播的输出。为满足以上两个条件权重初始化见公式(4.2)，初始化权重让神经网络在训练过程中学习到有用的信息，减少参数梯度为 0 的概率。

$$Var(W_i) = \frac{2}{n_{in} + n_{out}} \qquad \text{公式(4.2)}$$

4.2.2 模型调参及参数设置

LSTM 主要调整参数为：①学习率；②网络拓扑结构；③优化器选择；④时间窗口大小；⑤构建时间窗时的取值间隔 gap；⑥是否使用协变量。实验初期尝试将所有集群的数据同时输入并同时输出进行预测，但结果表明多变量的时间序列预测效果较差，故最终选择进行单变量(单变量+协变量)的预测模式。各个集群模型参数设置见表 4-4：

表 4-4 LSTM 模型参数设置

集群	学习率	网络拓扑结构	优化器	时间窗	时间窗取值间隔 gap	协变量
0	1e-4	1-2-1-2	Adam	240	24	否
1	1e-4	1-2-1-2	Adam	90	24	否
2	1e-4	1-2-1-2	Adam	90	24	否
3	3e-4	1-2-1-2	Adam	180	24	否
4	1e-4	1-2-1-2	Adam	240	24	否
5	5e-4	1-2-1-2	Adam	240	1	否
6	1e-4	1-2-1-2	Adam	30	1	否
7	1e-4	1-2-1-2	Adam	90	1	否

4.3 模型结果评价及可视化

本研究中的评价指标均基于真实值，即归一前的真实值和反归一后的预测值展开。使用 K 邻近，决策回归树，xgboost，梯度提升决策树，随机森林以及 LSTM 模型对同一天（2018 年 9 月 30 日）24 小时各集群需求量展开预测，各模型测试集评价指标对比见表 4-5：

表 4-5 模型试验评价指标

模型	MSE	MAE
KNN	11.73714193	1.84375000
决策回归树(手写)	15.43411289	1.93585134
xgboost	11.36940647	1.89084938
GBDT	11.33727501	2.02056170
随机森林	10.62777500	1.88187500
LSTM	7.52604144	1.65104164

从上表中可以看出，LSTM 对时间序列的预测效果要优于传统机器学习模型，同时在传统机器学习模型中，随机森林预测效果最优。

随机森林和 LSTM 实验效果如下图 4-2 和图 4-3 所示所示：

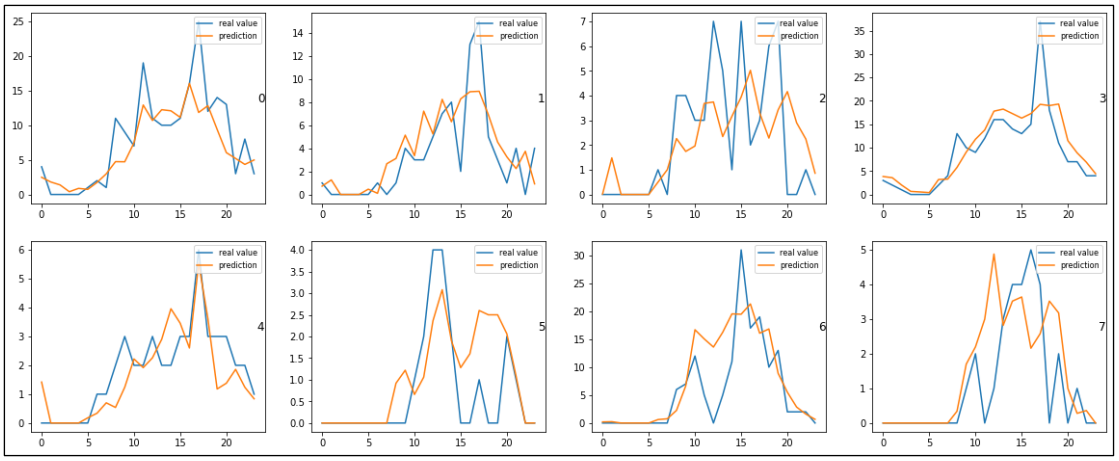


图 4-2 随机森林预测效果

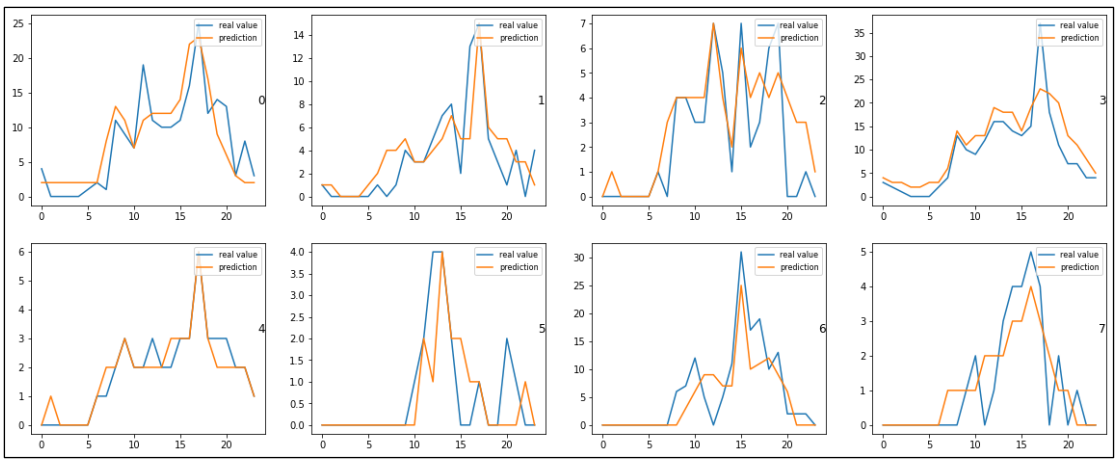


图 4-3 LSTM 预测效果

5 结论

本文使用 LSTM 等机器学习和神经网络方法对站点集群的短时需求预测展开研究，以 K-Medoids 聚类捕获站点空间信息和站点间的转化关系，以 LSTM 提取时空数据的时间特性，实验结果表明该方法具有一定可行性。创新性地使用了时间窗口的取值间隔，提取出不同日期在同一时段内的需求量，表明该方法在某些序列中有较好的效果。

但本研究仍然存在一些不足，普通的 LSTM 模型在同时进行多变量预测时效果较差，且对空间特征的提取仍然不够。在本文研究的基础上，已开展 convlstm 方法的研究，该方法将 LSTM 中的全连接替换为卷积操作，先提取空间特征，再将提取到的空间特征作为输入继续提取空间特征。目前，已完成该改进的数据预处理部分，将数据按照地理位置划分为 4 个区域，分别进行了 5*5 的栅格化处理；并正在基于 LSTMcell 重写 CONVLSTMcell 的结构，形成先空间在时间的网络构架；后续会继续针对此类时空数据、视频数据展开研究。

6 参考文献

[1] 何承韡. 公共自行车短时需求预测与车辆调度方法研究[D].苏州科技大学,2016.

[2] 黄彬.杭州市公共自行车系统运行状况调查分析与展望[J].城市规划学刊,2010(6):72-79

[3] 付俐哲. 基于时空聚类与 LSTM 神经网络的共享单车需求预测模型[D].西北师范大学,2021.

[4] 姜剑. 共享自行车调度需求预测与用户分流关键技术研究[D].杭州电子科技大学,2018.

[5] 胡冰华. 基于残差分析的时空数据预测模型的研究[D].华东交通大学,2020.

[6] Shi XJ, Chen Z, Wang H, Y DY. Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting[J]. nips,19 Sep 2015

[7] Jie Bao, Yu Hao, Wu Jiaming. Short-term FFBS demand prediction with multi-source data in a hybrid deep learning framework[J].IET Intelligent Transport Systems, 2019,13(9):1340-1347.

7 代码

7.1 过程文件

	表名	描述
1	out_2018.csv	站点-时间的对应需求矩阵
2	similarity_station.csv	站点相似度矩阵
3	sept_out.csv	聚类后的数据
4	time_based_out_sept_smooth.scv	输入神经网络的数据
5	basic_model_df.csv	输入 KNN 等模型的数据

7.2 预处理代码

1. 实验准备阶段 认识数据

In [47]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import folium
from datetime import datetime
import pyproj
from sklearn.datasets import make_blobs
from matplotlib import pyplot
import numpy as np
import random
from math import radians, cos, sin, asin, sqrt
```

In [48]:
```
data_2018 = pd.read_csv(r'./data/metro-bike-share-trips-2018-q1.csv')

Q=[2,3]
for q in Q:
    df = pd.read_csv(r'./data/metro-bike-share-trips-2018-q'+str(q)+'.csv')
    data_2018 = pd.concat([data_2018,df],axis=0,ignore_index=True)
```

f1:将站点经纬度数据进行整理归总

In [49]:
```
def cal_station(data):
    """
    输入：原始数据df
    输出：归总后的站点信息
    """
    station_id_list = list(data['start_station'].values)
    station_id_list.extend(list(data['end_station'].values))
    lat_list = list(data['start_lat'].values)
    lat_list.extend(list(data['end_lat'].values))
    lon_list = list(data['start_lon'].values)
    lon_list.extend(list(data['end_lon'].values))
    station_dict = {'station_id':station_id_list,'lat':lat_list,'lon':lon_list}
    station_df = pd.DataFrame(station_dict)
    station_df = station_df.drop_duplicates(subset=['station_id'],keep='first').reset_index(drop=True)
    station_df.to_csv('./data/station_info.csv')
    return station_df
```

f2:根据站点信息汇总，输入两站点编号得出两站点间距离

In [50]:
```
def haversine(station_df,id1,id2):
    """
    通过haversine公式计算两经纬度之间的距离
    输入：
    储存经纬度信息的df，两个站点的id
    输出：
    距离，单位（千米）
    """
    lon1 = station_df[station_df['station_id']==id1]['lon']
    lat1 = station_df[station_df['station_id']==id1]['lat']
    lon2 = station_df[station_df['station_id']==id2]['lon']
    lat2 = station_df[station_df['station_id']==id2]['lat']

    # 将十进制度数转化为弧度
    lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])

    # haversine公式
    a = sin((lat2 - lat1)/2)**2 + cos(lat1) * cos(lat2) * sin((lon2 - lon1) /2)**2
    c = 2 * asin(sqrt(a))
    r = 6371 # 地球平均半径，单位为千米
    return c * r  # 输出单位为千米
```

f3:可视化站点分布 ¶

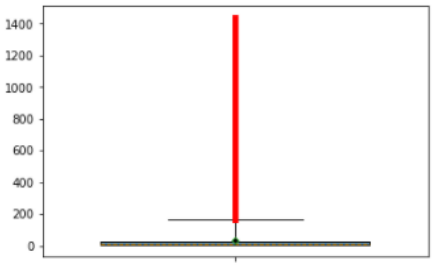
In [51]:
```
def see_station(station_info,mapping,col):
    """
    输入：站点信息
    输出：可视化站点分布
    """
    for i in range(station_info.shape[0]):
        folium.Marker([station_info.iloc[i]['lat'],station_info.iloc[i]['lon']],
                      popup=(str(station_info.iloc[i]['station_id'])), # 点击出现id
                      icon=folium.Icon(color=str(col))).add_to(mapping)
    return mapping
```

2. 预处理：剔除异常值

In [72]:
```
"""
取出异常值的index 并删除异常
"""
# 异常1：短时间内在同一站点借还
abnormal_1=data_2018[(data_2018['duration']==1)&(data_2018['start_station']==data_2018['end_station'])].index
# 异常2：借还站点不明确
abnormal_2=data_2018[(data_2018['end_station']==3000)|(data_2018['start_station']==3000)].index
# 两种异常状态取交集
abnormal=list(set(abnormal_1).union(set(abnormal_2)))
print(len(abnormal))
data_2018_cln=data_2018.drop(index = abnormal,axis=1).reset_index(drop=True)
```

```
7333
```

In [56]:
```
# 绘制箱型图观察骑行时间的异常状态
plt.boxplot(x = data_2018_cln['duration'],
            whis = 6,
            widths = 0.7,
            patch_artist = True,
            showmeans = True,
            boxprops = {'facecolor':'steelblue'},
            flierprops = {'markerfacecolor':'red', 'markeredgecolor':'red', 'markersize':4},
            meanprops = {'marker':'D','markerfacecolor':'black', 'markersize':4},
            medianprops = {'linestyle':'—','color':'orange'},
            labels = [''])
plt.show()
```



In [66]:
```
# 依据6倍的四分位差判断异常
Q1 = data_2018_cln['duration'].quantile(q = 0.25)
Q3 = data_2018_cln['duration'].quantile(q = 0.75)
low_whisker = Q1 - 6*(Q3 - Q1)# 寻找异常点上限
up_whisker = Q3 + 6*(Q3 - Q1)# 寻找异常点下限
data_2018_cln=data_2018_cln[(data_2018_cln['duration']>=low_whisker)&(data_2018_cln['duration']<=up_whisker)]
```

In [67]:
```
"""
将str转为datetime，精度只保留到小时的单位
"""
if type(data_2018_cln['start_time'][0])==str:
    data_2018_cln['start_time']=data_2018_cln['start_time'].apply(lambda x:datetime.strptime(x[:-6],'%Y-%m-%d %H'))
    data_2018_cln['end_time']=data_2018_cln['end_time'].apply(lambda x:datetime.strptime(x[:-6],'%Y-%m-%d %H'))
```

In [68]:
```
"""
创建站点和时间的对应矩阵
"""
date_list=[]
for m in range(1,13):
    if(m==1)|(m==3)|(m==5)|(m==7)|(m==8):
        for d in range(1,32):
            for h in range(0,24):
                temp_date = datetime.strptime('2018-'+str(m)+'-'+str(d)+'-'+str(h),'%Y-%m-%d-%H')
                date_list.append(temp_date)
    elif(m==4)|(m==6)|(m==9):
        for d in range(1,31):
            for h in range(0,24):
                temp_date = datetime.strptime('2018-'+str(m)+'-'+str(d)+'-'+str(h),'%Y-%m-%d-%H')
                date_list.append(temp_date)

    elif m==2:
        for d in range(1,29):
            for h in range(0,24):
                temp_date = datetime.strptime('2018-'+str(m)+'-'+str(d)+'-'+str(h),'%Y-%m-%d-%H')
                date_list.append(temp_date)
```

```
station_df=cal_station(data_2018_cln)
value_count = [[0]*(station_df.shape[0])]*(len(date_list))

out_dict=dict(zip(date_list,value_count))
out_df = pd.DataFrame(out_dict)
out_df['station_id']=station_df['station_id'].astype(int)
out_df.set_index(['station_id'],inplace=True)
out_df.sort_index(ascending=True,inplace=True)
out_df.head(5)
```

Out[68]:

station_id	2018-01-01 00:00:00	2018-01-01 01:00:00	2018-01-01 02:00:00	2018-01-01 03:00:00	2018-01-01 04:00:00	2018-01-01 05:00:00	2018-01-01 06:00:00	2018-01-01 07:00:00	2018-01-01 08:00:00	2018-01-01 09:00:00	...	2018-09-30 14:00:00	2018-09-30 15:00:00	2018-09-30 16:00:00	2018-09-30 17:00:00	2018-09-30 18:00:00
3005	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
3006	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
3007	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
3008	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
3010	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0

5 rows × 6552 columns

In [69]:

```
"""
统计各个站点在每个时间的需求量
"""
# for i in range(data_2018_cln.shape[0]):
#     index=data_2018_cln.loc[i]['start_station']
#     time=data_2018_cln.loc[i]['start_time'].strftime('%Y-%m-%d %H:%M:%S')
#     out_df.loc[index][time]=out_df.loc[index][time]+1

# out_df.to_csv('./data/out_2018.csv')
out_df=pd.read_csv('./data/out_2018.csv')
out_df.head(5)
```

Out[69]:

	station_id	2018-01-01 00:00:00	2018-01-01 01:00:00	2018-01-01 02:00:00	2018-01-01 03:00:00	2018-01-01 04:00:00	2018-01-01 05:00:00	2018-01-01 06:00:00	2018-01-01 07:00:00	2018-01-01 08:00:00	...	2018-09-30 14:00:00	2018-09-30 15:00:00	2018-09-30 16:00:00	2018-09-30 17:00:00	2018-09-30 18:00:00	2018-0 19:00:
0	3005	0	0	1	0	0	1	0	0	0	...	11	2	4	1	4	
1	[illegible]	[illegible]	0	0	0	0	0	0	0	0	...	0	1	3	2	4	
2	3007	0	1	0	0	0	0	0	0	0	...	1	4	1	2	0	
3	3008	0	0	0	0	0	0	0	0	0	...	2	0	4	1	0	
4	3010	0	0	0	0	0	0	0	0	0	...	0	0	0	0	1	

点击展开输出；双击隐藏输出

5 rows × 6553 columns

In [75]:

```
"""
计算相似度矩阵，a代表站点交互关系的权重，矩阵中的值越大，表示相似度越弱，距离越远
站点间的距离和交互关系都进行了无量纲化处理
"""

# 计算每个站点总的出借量
station_df=cal_station(data_2018_cln)
time_based_out = pd.DataFrame(out_df.values.T, index=out_df.columns, columns=out_df.index)#转置
year_count=np.array([time_based_out[column].sum() for column in list(time_based_out.columns)]) # 依据行标题进行求和
station_id=np.array(time_based_out.columns)
station_df['year_count']=year_count

# 制作相似度矩阵
station_df=station_df.sort_values(by='station_id').reset_index(drop=True)
similarity = [[0]*(station_df.shape[0])]*(station_df.shape[0])
similarity_dict=dict(zip(station_df['station_id'],similarity))
similarity_df = pd.DataFrame(similarity_dict)
similarity_df['station_id']=station_df['station_id']
similarity_df.set_index(['station_id'],inplace=True)

# 依据距离和转换关系形成相似度矩阵
for item in range(data_2018_cln.shape[0]):
    i=data_2018_cln.loc[item]['start_station']
    j=data_2018_cln.loc[item]['end_station']
    similarity_df.loc[i][j]=similarity_df.loc[i][j]+1

a=0.5
for i in range(similarity_df.shape[0]):
    similarity_df.iloc[i]= (1-a*(similarity_df.iloc[i]/station_df.iloc[i]['year_count']))

for i in range(similarity_df.shape[0]):
    for j in range(i,similarity_df.shape[0]):
        similarity_df.iloc[i,j]=similarity_df.iloc[i,j]*(0.01*haversine(station_df,int(station_df.iloc[i]['station_id']),int(statio
        similarity_df.iloc[j,i]=similarity_df.iloc[j,i]*(0.01*haversine(station_df,int(station_df.iloc[i]['station_id']),int(statio

# similarity_df = pd.read_csv('./data/bike/similarity_station.csv') # 站点相似性矩阵
# similarity_df.set_index(['station_id'], inplace=True)
# similarity_df.head(5)
```

Out[75]:

station_id	3005	3006	3007	3008	3010	3011	3013	3014	3016	3018	...	4220	4227	4244	4245	
3005	0.000000	0.003574	0.004178	0.004290	0.013066	0.011763	0.298788	0.021365	0.016367	0.005352	...	0.029055	0.023422	0.042644	0.035480	0.03
3006	0.003635	0.000000	0.005770	0.005679	0.009577	0.011458	0.295526	0.021699	0.016130	0.003758	...	0.032537	0.021522	0.041244	0.034330	0.03
3007	0.004137	0.005743	0.000000	0.008529	0.014919	0.015984	0.301062	0.017309	0.012285	0.008758	...	0.027253	0.027167	0.046640	0.039610	0.03
3008	0.004324	0.005664	0.008578	0.000000	0.012857	0.007583	0.296660	0.025706	0.020713	0.003987	...	0.031535	0.019951	0.038690	0.031502	0.02
3010	0.013130	0.009576	0.014932	0.012860	0.000000	0.012899	0.286125	0.027121	0.021465	0.008873	...	0.042161	0.016349	0.036165	0.030016	0.02

5 rows × 127 columns

In [45]:
```
class KMediod():
    """
    KMediod聚类 用相似度矩阵替换欧氏距离计算
    输入：
    记录站点信息的表格
    记录站点间相似度的表格
    聚类个数
    输出：
    各个站点所属的类别
    """

    def __init__(self, data, similarity_df, k):
        self.org_data=data
        x0=list(data['lat'].values)
        x1=list(data['lon'].values)
        self.data = np.array([list(D) for D in(zip(x0,x1))])
        self.similarity = similarity_df
        self.k = k

    def cal_distance(self, x, y):
        # 利用相似度矩阵代替欧式距离的计算
        i = self.org_data[(self.org_data['lat']==x[0])&(self.org_data['lon']==x[1])].index
        j = self.org_data[(self.org_data['lat']==y[0])&(self.org_data['lon']==y[1])].index
        return self.similarity.iloc[i,j].values

    def run_k_center(self, cal_distance):
        print('初始化', self.k, '个中心点')
        indexs = list(range(len(self.data)))
        # 为保证聚类的稳定性及准确性，人为设定初始化中心点
        init_centroids_index = self.org_data[(self.org_data['station_id']==4154)|(self.org_data['station_id']==4144)|
                                             (self.org_data['station_id']==4216)|(self.org_data['station_id']==4204)|
                                             (self.org_data['station_id']==4133)|(self.org_data['station_id']==3064)|
                                             (self.org_data['station_id']==3066)|(self.org_data['station_id']==4245)|
                                             (self.org_data['station_id']==3056)|(self.org_data['station_id']==3054)|
                                             (self.org_data['station_id']==3038)].index
        centroids = self.data[init_centroids_index, :]   # 初始中心点
        # 确定种类编号
        levels = list(range(self.k))
        sample_target = []
        if_stop = False
        while(not if_stop):
            if_stop = True
            classify_points = [[centroid] for centroid in centroids]
            sample_target = []
            # 遍历数据
            for sample in self.data:
                # 计算距离，由距离该数据最近的核心，确定该点所属类别
                distances = [cal_distance(sample, centroid) for centroid in centroids]
                cur_level = np.argmin(distances)
                sample_target.append(cur_level)
                # 统计，方便迭代完成后重新计算中间点
                classify_points[cur_level].append(sample)
            # 重新划分质心
            for i in range(self.k):  # 几类中分别寻找一个最优点
                distances = [cal_distance(point_1, centroids[i]) for point_1 in classify_points[i]]
                #  计算出现在中心点和其他所有点的距离总和
                now_distances = sum(distances)
                for point in classify_points[i]:
                    distances = [cal_distance(point_1, point) for point_1 in classify_points[i]]
                    new_distance = sum(distances)
                    # 计算出该聚簇中各个点与其他所有点的总和，若是有小于当前中心点的距离总和的，中心点去掉
                    if new_distance < now_distances:
                        now_distances = new_distance
                        centroids[i] = point     # 换成该点
                        if_stop = False
        return sample_target
```

In [46]:
```
"""
进行聚类以及可视化
"""
test_one = KMediod(Descartes_df,similarity_df,k=11)
seprate=test_one.run_k_center(test_one.cal_distance)
station_df['sept']=seprate

mapping = folium.Map(location=[34.049198,-118.252831]) # 将地图中心定位到LA
color_list=['blue','green','lightred','orange','pink','darkpurple','gray','beige','purple','darkred','darkgreen']
for i in range(11):
    mapping=see_station(station_df[station_df['sept']==i],mapping,color_list[i])
mapping
```

In [22]:
```
sept_out_df = out_df.copy()
sept_out_df['sept']=station_df['sept'].values
sept_out_df=sept_out_df.groupby('sept').sum() # 基于分好的类统计各时段个站点需求量
sept_out_df=sept_out_df.reset_index()
sept_out_df=sept_out_df.drop(columns=['sept','station_id'])
sept_out_df
```

Out[22]:

	2018-01-01 00:00:00	2018-01-01 01:00:00	2018-01-01 02:00:00	2018-01-01 03:00:00	2018-01-01 04:00:00	2018-01-01 05:00:00	2018-01-01 06:00:00	2018-01-01 07:00:00	2018-01-01 08:00:00	2018-01-01 09:00:00	...	2018-09-30 14:00:00	2018-09-30 15:00:00	2018-09-30 16:00:00	2018-09-30 17:00:00	2018-09-30 18:00:00	2018-0 19:00:
0	2	10	2	0	0	3	1	1	2	2	...	49	31	41	25	28	
1	5	0	4	1	0	0	0	0	0	0	...	8	2	13	15	5	
2	0	0	0	0	0	0	0	0	0	0	...	1	7	2	3	6	
3	9	10	2	0	2	1	2	0	0	3	...	58	45	63	37	18	
4	0	1	1	0	0	0	0	0	2	0	...	2	9	3	6	3	
5	0	0	0	0	0	0	0	0	1	1	...	6	0	0	1	0	
6	1	0	1	0	2	0	3	2	2	0	...	0	0	0	0	0	
7	6	1	0	2	1	3	7	5	11	20	...	0	0	0	0	0	
8	1	0	1	0	0	0	0	2	3	0	...	11	31	17	19	10	
9	0	1	0	0	0	0	0	0	1	0	...	4	4	5	4	0	
10	0	0	0	0	0	0	0	0	0	0	...	3	2	0	0	3	

11 rows × 6552 columns

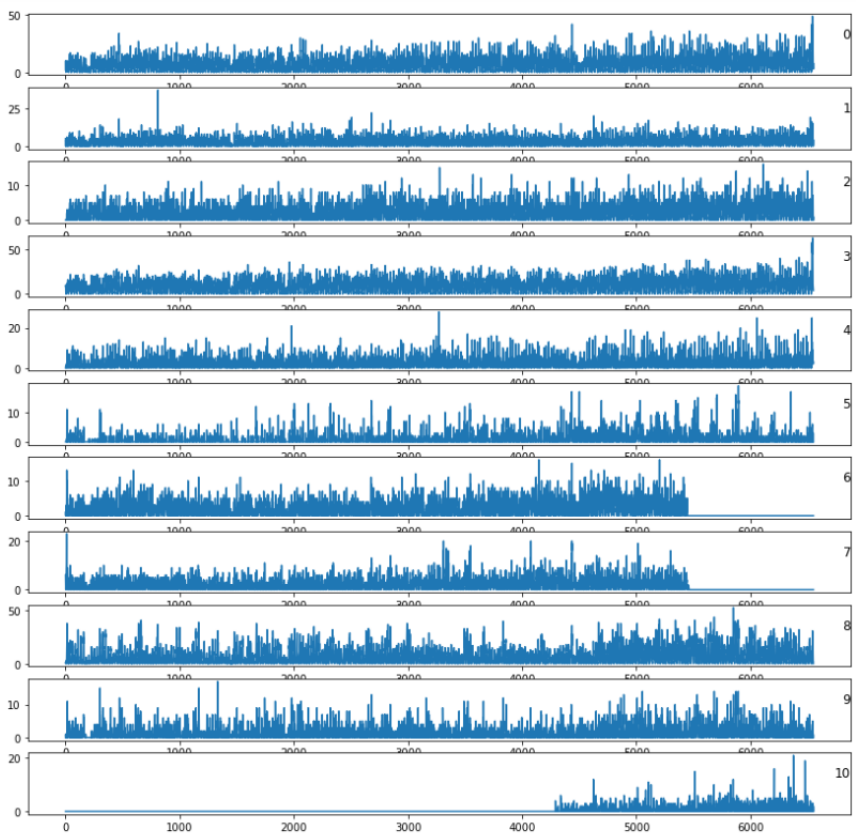
In [138]:
```
time_based_out_sept = pd.DataFrame(sept_out_df.values.T, index=sept_out_df.columns, columns=sept_out_df.index)#转置
time_based_out_sept
```

Out[138]:

	0	1	2	3	4	5	6	7	8	9	10
2018-01-01 00:00:00	2	5	0	9	0	0	1	6	1	0	0
2018-01-01 01:00:00	10	0	0	10	1	0	0	1	0	1	0
2018-01-01 02:00:00	2	4	0	2	1	0	1	0	1	0	0
2018-01-01 03:00:00	0	1	0	0	0	0	0	2	0	0	0
2018-01-01 04:00:00	0	0	0	2	0	0	2	1	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...
2018-09-30 19:00:00	14	3	7	11	3	0	0	0	13	2	3
2018-09-30 20:00:00	13	1	0	7	3	2	0	0	2	0	2
2018-09-30 21:00:00	3	4	0	7	2	3	0	0	2	1	0
2018-09-30 22:00:00	8	0	1	4	2	0	0	0	2	0	2
2018-09-30 23:00:00	3	4	0	4	3	0	0	0	0	0	0

6552 rows × 11 columns

In [141]:
```
def show_graph(dataset):
    values = dataset.values
    columns = [i for i in range(dataset.shape[1])]
    pyplot.figure(figsize=(14,14))
    i=1
    for column in columns:
        pyplot.subplot(len(columns), 1, i)
        pyplot.plot(values[:, column])
        pyplot.title(column, y=0.5, loc='right')
        i += 1
    pyplot.show()

show_graph(time_based_out_sept)
```



In [143]:
```
# 由于第6、7两类站点后期出现弃用，做删除处理
time_based_out_sept.drop(columns=[6,7,10],inplace=True)
time_based_out_sept.rename(columns={8:6,9:7},inplace=True)
time_based_out_sept
```

Out[143]:

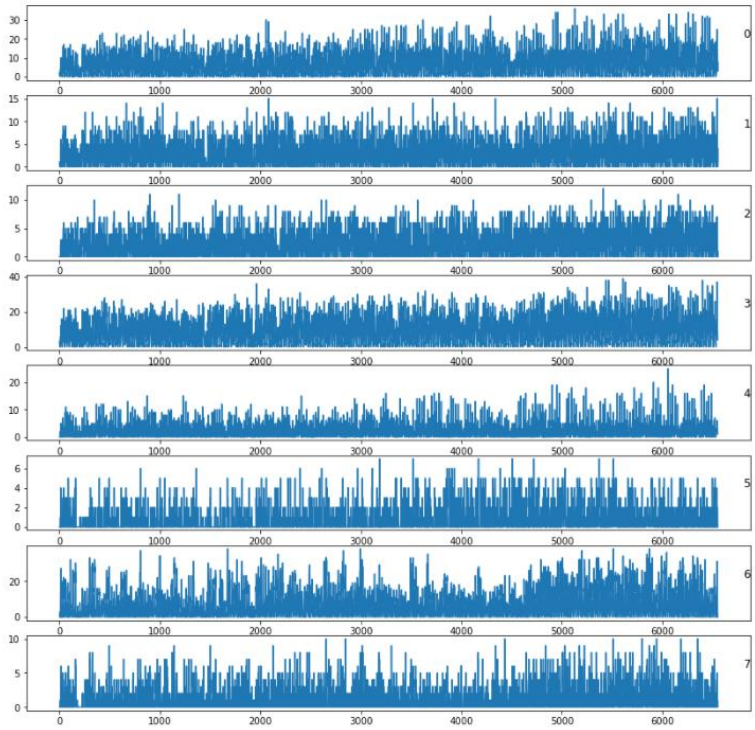
	0	1	2	3	4	5	6	7
2018-01-01 00:00:00	2	5	0	9	0	0	1	0
2018-01-01 01:00:00	10	0	0	10	1	0	0	1
2018-01-01 02:00:00	2	4	0	2	1	0	1	0
2018-01-01 03:00:00	0	1	0	0	0	0	0	0
2018-01-01 04:00:00	0	0	0	2	0	0	0	0
...	...	...	...	...	...	...	...	...
2018-09-30 19:00:00	14	3	7	11	3	0	13	2
2018-09-30 20:00:00	13	1	0	7	3	2	2	0
2018-09-30 21:00:00	3	4	0	7	2	3	2	1
2018-09-30 22:00:00	8	0	1	4	2	0	2	0
2018-09-30 23:00:00	3	4	0	4	3	0	0	0

6552 rows × 8 columns

In [144]:
```
time_based_out_sept.to_csv('./data/bike/time_based_out_sept.csv')
# station_df.to_csv('./data/bike/station_df.csv') # 站点经纬度信息
# similarity_df.to_csv('./data/bike/similarity_station.csv') # 站点相似性矩阵
# out_df.to_csv('./data/bike/out_2018.csv')# 所有站点各个时刻的流出信息
# sept_out_df.to_csv('./data/bike/sept_out.csv') # 聚类后各类的信息
# sept_out_df = pd.read_csv('./data/bike/sept_out.csv')
```

In [147]:
```
"""
对每个集群的各个小时分别做平滑处理，剔除在1.5倍4分位距外的点，使用该集群该小时四舍五入的平均数代替
"""
import math
time_based_out_sept_smooth = time_based_out_sept.copy()
for i in range(time_based_out_sept_smooth.shape[1]):
    for t in range(24):
        Q1 = time_based_out_sept_smooth[i][t::24].quantile(q = 0.25)
        Q3 = time_based_out_sept_smooth[i][t::24].quantile(q = 0.75)
        low_whisker = Q1 - 1.5*(Q3 - Q1)# 寻找异常点上限.
        up_whisker = Q3 + 1.5*(Q3 - Q1)# 寻找异常点下限
        time_based_out_sept_smooth[i][t::24] = time_based_out_sept_smooth[i][t::24].apply(lambda x: x if (x<=up_whisker)&(x>=low_whisker) e
```

In [148]:
```
time_based_out_sept_smooth.to_csv('./data/bike/time_based_out_sept_smooth.csv')
show_graph(time_based_out_sept_smooth)
```



In [150]:
```
"""
提取日期中的月、日、小时作为特征，并将集群类别也化为特征
"""
time_feture_df = pd.DataFrame([[0]*4]*time_based_out_sept.shape[0], index=time_based_out_sept.index, columns=['month','day','weekday','hour'])#
time_feture_df=time_feture_df.reset_index().rename(columns={'index':'datetime'})
time_feture_df['datetime']=time_feture_df['datetime'].apply(lambda x:datetime.strptime(x,'%Y-%m-%d %H:%M:%S'))
time_feture_df['month'] =  time_feture_df['datetime'].dt.month
time_feture_df['day'] =  time_feture_df['datetime'].dt.day
time_feture_df['hour'] =  time_feture_df['datetime'].dt.hour
time_feture_df['weekday'] =  time_feture_df['datetime'].dt.weekday+1

time_based_out_sept_smooth_timeinfro = time_based_out_sept_smooth.copy()
time_based_out_sept_smooth_timeinfro['month'] = time_feture_df['month'].values
time_based_out_sept_smooth_timeinfro['day'] = time_feture_df['day'].values
time_based_out_sept_smooth_timeinfro['hour'] = time_feture_df['hour'].values
time_based_out_sept_smooth_timeinfro['weekday'] = time_feture_df['weekday'].values

time_based_out_sept_smooth_timeinfro.to_csv('./data/bike/time_based_out_sept_smooth_timeinfro.csv')
time_based_out_sept_smooth_timeinfro
```

Out[150]:

	0	1	2	3	4	5	6	7	month	day	hour	weekday
2018-01-01 00:00:00	2	1	0	3	0	0	1	0	1	1	0	1
2018-01-01 01:00:00	1	0	0	2	0	0	0	0	1	1	1	1
2018-01-01 02:00:00	2	0	0	2	0	0	0	0	1	1	2	1
2018-01-01 03:00:00	0	0	0	0	0	0	0	0	1	1	3	1
2018-01-01 04:00:00	0	0	0	2	0	0	0	0	1	1	4	1
...	...	...	...	...	...	...	...	...	...	...	...	...
2018-09-30 19:00:00	14	3	7	11	3	0	13	2	9	30	19	7
2018-09-30 20:00:00	13	1	0	7	3	2	2	0	9	30	20	7
2018-09-30 21:00:00	3	4	0	7	2	1	2	1	9	30	21	7
2018-09-30 22:00:00	8	0	1	4	2	0	2	0	9	30	22	7
2018-09-30 23:00:00	3	4	0	4	1	0	0	0	9	30	23	7

6552 rows × 12 columns

In [26]:
```
time_based_out_sept_smooth_timeinfro=pd.read_csv('./data/bike/time_based_out_sept_smooth_timeinfro.csv')
```

In [27]:
```
basic_model_df = pd.DataFrame([[0]*6]*52416, columns=['cluster','month','day','weekday','hour','demands'])#待置
```

In [28]:
```
import warnings
warnings.filterwarnings("ignore")

for i in range(8):
    basic_model_df['cluster'][i*6552:i*6552+6552] = i
    basic_model_df['month'][i*6552:i*6552+6552] = time_based_out_sept_smooth_timeinfro['month'].values
    basic_model_df['day'][i*6552:i*6552+6552] = time_based_out_sept_smooth_timeinfro['day'].values
    basic_model_df['weekday'][i*6552:i*6552+6552] = time_based_out_sept_smooth_timeinfro['weekday'].values
    basic_model_df['hour'][i*6552:i*6552+6552] = time_based_out_sept_smooth_timeinfro['hour'].values
    basic_model_df['demands'][i*6552:i*6552+6552] = time_based_out_sept_smooth_timeinfro[str(i)].values
```

In [29]:
```
basic_model_df.to_csv('./data/bike/basic_model_df.csv')
basic_model_df
```

Out[29]:

	cluster	month	day	weekday	hour	demands
0	0	1	1	1	0	2
1	0	1	1	1	1	1
2	0	1	1	1	2	2
3	0	1	1	1	3	0
4	0	1	1	1	4	0
...	...	...	...	...	...	...
52411	7	9	30	7	19	2
52412	7	9	30	7	20	0
52413	7	9	30	7	21	1
52414	7	9	30	7	22	0
52415	7	9	30	7	23	0

52416 rows × 6 columns

In [30]:
```
basic_model_df=pd.read_csv('./data/bike/basic_model_df.csv')
```

7.3 LSTM 代码

In [1]:
```
import pandas as pd
import numpy as np
import torch
import torch.nn as nn
import copy
from matplotlib import pyplot as plt
from matplotlib import pyplot
```

In [2]:
```
var_origin_0=pd.read_csv(r'C:\\Users\\JIBINBIN\\DA\\data\\bike\\time_based_out_sept.csv')
var_origin_1=pd.read_csv(r'C:\\Users\\JIBINBIN\\DA\\data\\bike\\time_based_out_sept_smooth.csv')
var_origin_2=pd.read_csv(r'C:\\Users\\JIBINBIN\\DA\\data\\bike\\time_based_out_sept_smooth_timeinfro.csv')
var_origin_0 = var_origin_0.drop(columns=['Unnamed: 0'])
var_origin_1 = var_origin_1.drop(columns=['Unnamed: 0'])
var_origin_2 = var_origin_2.drop(columns=['Unnamed: 0'])
```

In [3]:
```
# 数据归一化
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(var_origin_1)

var = torch.tensor(data=scaled).to(torch.float32)
```

In [4]:
```
def splitData(var,num_val,num_test):
    """
    拆分训练集、验证集、测试集
    输入:
    整体数据集、验证集个数、测试集个数
    输出:
    拆分后的数据集
    """
    train_size=int(len(var)-num_val-num_test)
    train_data=var[0:train_size]
    val_data=var[train_size:train_size+num_val]
    test_data=var[train_size+num_val:train_size+num_val+num_test]
    return train_data,val_data,test_data
```

In [5]:
```
def create_train_sequence(input_data, tw, gap):
    """
    划分训练集序列和标签
    输入:
    训练集数据以及滑动窗口大小
    """
    train_seq_label=[]
    L = len(input_data)
    for i in range(L-tw*gap):
        train_seq = (input_data[i:i+tw*gap:gap])
        train_label = (input_data[i+tw*gap:i+tw*gap+1])
        train_seq_label.append((train_seq,train_label))

    return train_seq_label
```

In [6]:
```
def create_val_sequence(train_data, val_data, tw, gap):
    """
    划分验证集序列和标签，此前拆分数据集所得验证集为验证集标签
    输入：
    训练集数据、验证集数据以及滑动窗口
    """
    temp = torch.cat((train_data, val_data))  #先将训练集和测试集合并
    val_seq_label = []
    L = len(val_data)
    for i in range(L):
        val_seq = (temp[-(tw*gap+L)+i:-L+i:gap])
        val_label = (val_data[i:i+1])
        val_seq_label.append((val_seq, val_label))

    return val_seq_label
```

In [7]:
```
def create_test_sequence(train_data, val_data, test_data, tw, gap):
    """
    划分测试集序列和标签，此前拆分数据集所得验证集为验证集标签
    输入：
    训练集数据、验证集数据、测试集数据以及滑动窗口
    """
    temp = torch.cat((train_data, val_data))   #先将训练集和验证集合并
    temp = torch.cat((temp, test_data)) # 再将其与测试集合并
    test_seq_label = []
    L = len(test_data)
    for i in range(L):
        test_seq = (temp[-(tw*gap+L)+i:-L+i:gap])
        test_label = (test_data[i:i+1])
        test_seq_label.append((test_seq, test_label))

    return test_seq_label
```

In [8]:
```
"""
构建LSTM网络，网络拓扑结构为1-2-1-2
使用Xavier初始化
"""
from torch import nn
import torch.nn.functional as F
import torch.nn.init as init
import math
class LSTM(nn.Module):
    def __init__(self,input_size=1,hidden_layer_size=2,output_size=1,num_layers=2):
        super().__init__()
        self.input_size=input_size
        self.hidden_layer_size=hidden_layer_size
        self.lstm=nn.LSTM(input_size,hidden_layer_size,num_layers)
        self.linear1=nn.Linear(hidden_layer_size,hidden_layer_size)
        self.linear2=nn.Linear(hidden_layer_size,output_size)
        self.hidden_cell=(torch.zeros(num_layers,1,self.hidden_layer_size),torch.zeros(num_layers,1,self.hidden_layer_size))
        init_rnn(self.lstm,'xavier')

    def forward(self,input_seq):
        lstm_out, self.hidden_cell = self.lstm(input_seq.reshape(len(input_seq),1,self.input_size), self.hidden_cell)
        out=self.linear1(lstm_out.view(len(input_seq), -1))
        out=torch.tanh(out)
        predictions = self.linear2(out)
        return predictions[-1]


def init_rnn(x, type='uniform'):
    """
    初始化网络权重
    网络采tanh激活函数，使用Xavier初始化
    """
    for layer in x._all_weights:
        for w in layer:
            if 'weight' in w:
                if type == 'xavier':
                    init.xavier_normal_(getattr(x, w))
                elif type == 'uniform':
                    stdv = 1.0 / math.sqrt(x.hidden_size)
                    init.uniform_(getattr(x, w), -stdv, stdv)
                elif type == 'normal':
                    stdv = 1.0 / math.sqrt(x.hidden_size)
                    init.normal_(getattr(x, w), .0, stdv)
                else:
                    raise ValueError
```

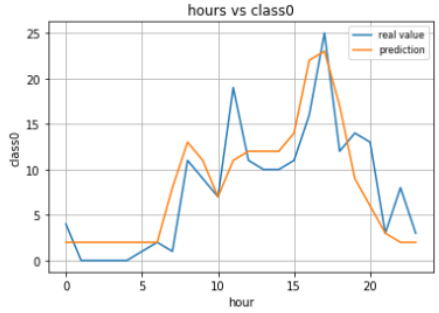
In [9]:
```
def train_model(train_seq_label,class_i, model,learning_r):
    """
    """

    loss_function=nn.MSELoss() # 损失函数
    optimizer = torch.optim.Adam(model.parameters(), lr=learning_r) # 优化器，之前尝试过SGD效果较差且出现了预测试为一条直线的情况
    # 模型进入训练状态
    model.train()
    print('训练',class_i,'模型')
    epochs = 5
    Loss = nn.MSELoss(reduction='mean')
    for i in range(epochs):
        total_loss = 0
        y_pred=[]
        print('<<<<<<<<<<<<<<<<<<<<<<<<<<<<<{}>>>>>>>>>>>>>>>>>>>>>>>>>>>>>'.format(i+1))
        for seq,label in train_seq_label:
            optimizer.zero_grad()
            # 需要清空 LSTM 的隐状态
            # 将其从上个实例的历史中分离出来
            # 重新初始化隐藏层数据，避免受之前运行代码的干扰,如果不重新初始化，会有报错。
            model.hidden_cell = (torch.zeros(2, 1, model.hidden_layer_size),torch.zeros(2, 1, model.hidden_layer_size))
            y_pred_i = model(seq[:,int(class_i[5:])])
            "！！！！！model(data)算出来直接进入Loss函数计算！！！！！ 不要在对其做reshape等运算，否则会出现全网梯度为None网络不更新的情况
            loss = Loss(y_pred_i[0],label[0,int(class_i[5:])])
            total_loss = total_loss + loss
            loss.backward() # 反向传播
            optimizer.step() # 更新梯度
            y_pred.append(y_pred_i[0])
        print("训练集MSE:{}".format(total_loss.data/len(train_seq_label)))
```

In [10]:
```
def val_model(val_seq_label,class_i, model):
    i=0
    loss= 0
    loss_function=nn.MSELoss() # 损失函数
    # 模型当前模式不进行网络更新
    model.eval()
    for seq,label in val_seq_label:
        with torch.no_grad():
            model.hidden = (torch.zeros(1, 1, model.hidden_layer_size),torch.zeros(1, 1, model.hidden_layer_size))
            y_pred_i = model(seq[:,int(class_i[5:])])
            loss_i = loss_function(y_pred_i[0],label[0,int(class_i[5:])])
            loss=loss+loss_i
        i=i+1

    return (loss.data/len(val_seq_label))
```

In [107]:
```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error


def test_model(test_seq_label,class_i, model):
    i=0
    y_pred = copy.deepcopy(test_data)
    # 模型当前模式不进行网络更新
    model.eval()
    for seq,label in test_seq_label:
        with torch.no_grad():
            model.hidden = (torch.zeros(1, 1, model.hidden_layer_size),torch.zeros(1, 1, model.hidden_layer_size))
            y_pred_i = model(seq[:,int(class_i[5:])])
            y_pred[i]=y_pred_i.data
        i=i+1

    actual_predictions = scaler.inverse_transform(np.array(y_pred))
    for i in range(len(actual_predictions[:,int(class_i[5:])])):
         actual_predictions[i,int(class_i[5:])] = math.floor(actual_predictions[i,int(class_i[5:])]+0.5)
    y_true = scaler.inverse_transform(test_data)


    plt.grid(True)

    for i in range(24):
        if(actual_predictions[i,int(class_i[5:])]<0):
            actual_predictions[i,int(class_i[5:])]=0

    plt.plot(np.arange(len(test_data)),y_true[:,int(class_i[5:])],label='real value')
    plt.plot(np.arange(len(test_data)),actual_predictions[:,int(class_i[5:])],label='prediction')

    plt.title('hours vs '+class_i)
    plt.ylabel(class_i)
    plt.xlabel('hour')

    plt.legend(loc='upper right',fontsize=8)


    return actual_predictions[:,int(class_i[5:])], y_true[:,int(class_i[5:])]
```

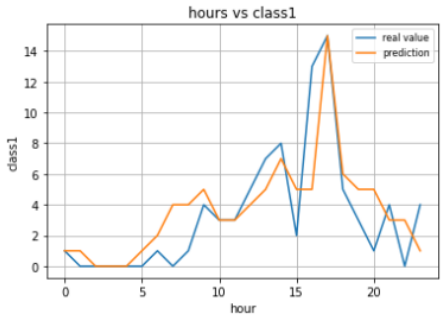
In [39]:
```
train_data,val_data,test_data = splitData(var,48,24)
train_seq_label = create_train_sequence(train_data, 240,24)
print('The total number of train windows is',len(train_seq_label))
val_seq_label = create_val_sequence(train_data,val_data,240,24)
print('The total number of validation windows is',len(val_seq_label))
test_seq_label = create_test_sequence(train_data,val_data,test_data, 240,24)
print('The total number of validation windows is',len(test_seq_label))

model0=LSTM()
train_model(train_seq_label,'class0', model0,0.001)
y_hat_i,y_ture_i = test_model(test_seq_label,'class0', model0)
y_hat[0]=y_hat_i
y_ture[0]=y_ture_i
```

```
The total number of train windows is 720
The total number of validation windows is 48
The total number of validation windows is 24
训练 class0 模型
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<1>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
训练集MSE:0.029504094272851944
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<2>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
训练集MSE:0.014331739395856857
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<3>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
训练集MSE:0.013639123179018497
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<4>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
训练集MSE:0.013543273322284222
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<5>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
训练集MSE:0.013533441349864006
```



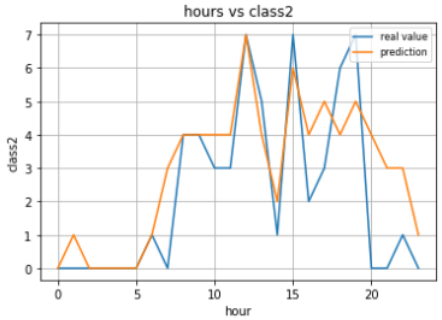
In [57]:
```
model1=LSTM()

train_seq_label = create_train_sequence(train_data,90,24)
print('The total number of train windows is',len(train_seq_label))
val_seq_label = create_val_sequence(train_data,val_data,90,24)
print('The total number of validation windows is',len(val_seq_label))
test_seq_label = create_test_sequence(train_data,val_data,test_data, 90,24)
print('The total number of validation windows is',len(test_seq_label))

train_model(train_seq_label,'class1', model1,0.001)
y_hat_i,y_ture_i = test_model(test_seq_label,'class1', model1)
y_hat[1]=y_hat_i
y_ture[1]=y_ture_i
```

```
The total number of train windows is 4320
The total number of validation windows is 48
The total number of validation windows is 24
训练 class1 模型
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<1>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
训练集MSE:0.024545518681406975
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<2>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
训练集MSE:0.019827950745821
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<3>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
训练集MSE:0.01950506865978241
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<4>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
训练集MSE:0.01943589560687542
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<5>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
训练集MSE:0.019369062036275864
```



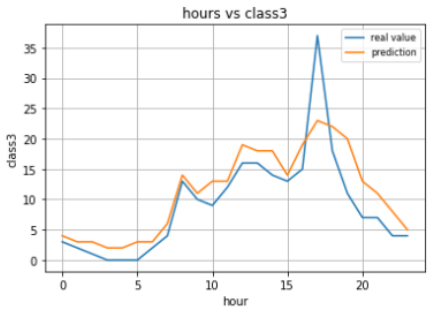
In [87]:
```
model2=LSTM()

train_seq_label = create_train_sequence(train_data, 90,24)
print('The total number of train windows is',len(train_seq_label))
val_seq_label = create_val_sequence(train_data,val_data,90,24)
print('The total number of validation windows is',len(val_seq_label))
test_seq_label = create_test_sequence(train_data,val_data,test_data, 90,24)
print('The total number of validation windows is',len(test_seq_label))

train_model(train_seq_label,'class2', model2,0.001)
y_hat_i,y_ture_i = test_model(test_seq_label,'class2', model2)
y_hat[2]=y_hat_i
y_ture[2]=y_ture_i
```

```
The total number of train windows is 4320
The total number of validation windows is 48
The total number of validation windows is 24
训练 class2 模型
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<1>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
训练集MSE:0.019926294684410095
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<2>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
训练集MSE:0.018806103616952896
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<3>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
训练集MSE:0.018827974796295166
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<4>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
训练集MSE:0.018864808604121208
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<5>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
训练集MSE:0.01888577826321125
```



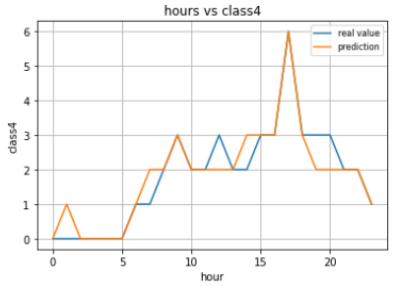
In [49]:
```
model3=LSTM()

train_seq_label = create_train_sequence(train_data, 180,24)
print('The total number of train windows is',len(train_seq_label))
val_seq_label = create_val_sequence(train_data,val_data,180,24)
print('The total number of validation windows is',len(val_seq_label))
test_seq_label = create_test_sequence(train_data,val_data,test_data, 180,24)
print('The total number of validation windows is',len(test_seq_label))

train_model(train_seq_label,'class3', model3,0.001)
y_hat_i,y_ture_i = test_model(test_seq_label,'class3', model3)
y_hat[3]=y_hat_i
y_ture[3]=y_ture_i
```

```
The total number of train windows is 2160
The total number of validation windows is 48
The total number of validation windows is 24
训练 class3 模型
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<1>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
训练集MSE:0.058346688747406006
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<2>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
训练集MSE:0.0131484754383564
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<3>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
训练集MSE:0.012977544218301773
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<4>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
训练集MSE:0.012830703519284725
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<5>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
训练集MSE:0.01270116213709116
```



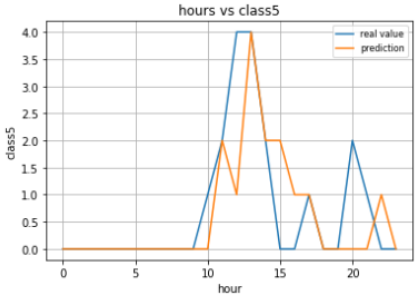
In [109]:
```
model4=LSTM()

train_seq_label = create_train_sequence(train_data, 240,24)
print('The total number of train windows is',len(train_seq_label))
val_seq_label = create_val_sequence(train_data,val_data,240,24)
print('The total number of validation windows is',len(val_seq_label))
test_seq_label = create_test_sequence(train_data,val_data,test_data, 240,24)
print('The total number of validation windows is',len(test_seq_label))

train_model(train_seq_label,'class4', model4,0.003)

y_hat_i,y_ture_i = test_model(test_seq_label,'class4', model4)

y_hat[4]=y_hat_i
y_ture[4]=y_ture_i
```

```
The total number of train windows is 720
The total number of validation windows is 48
The total number of validation windows is 24
训练 class4 模型
<<<<<<<<<<<<<<<<<<<<<<<<<<<<1>>>>>>>>>>>>>>>>>>>>>>>>>>>>
训练集MSE:0.07330556213855743
<<<<<<<<<<<<<<<<<<<<<<<<<<<<2>>>>>>>>>>>>>>>>>>>>>>>>>>>>
训练集MSE:0.01596115343272686
<<<<<<<<<<<<<<<<<<<<<<<<<<<<3>>>>>>>>>>>>>>>>>>>>>>>>>>>>
训练集MSE:0.013476424850523472
<<<<<<<<<<<<<<<<<<<<<<<<<<<<4>>>>>>>>>>>>>>>>>>>>>>>>>>>>
训练集MSE:0.013499435968697071
<<<<<<<<<<<<<<<<<<<<<<<<<<<<5>>>>>>>>>>>>>>>>>>>>>>>>>>>>
训练集MSE:0.013516679406166077
```



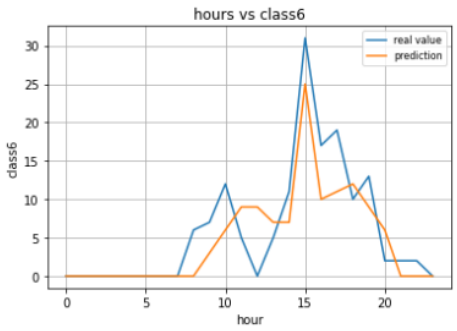
In [73]:
```
model5=LSTM()

train_seq_label = create_train_sequence(train_data, 240,1)
print('The total number of train windows is',len(train_seq_label))
val_seq_label = create_val_sequence(train_data,val_data,240,1)
print('The total number of validation windows is',len(val_seq_label))
test_seq_label = create_test_sequence(train_data,val_data,test_data, 240,1)
print('The total number of validation windows is',len(test_seq_label))

train_model(train_seq_label,'class5', model5,0.001)
y_hat_i,y_ture_i = test_model(test_seq_label,'class5', model5)
y_hat[5]=y_hat_i
y_ture[5]=y_ture_i
```

```
The total number of train windows is 6240
The total number of validation windows is 48
The total number of validation windows is 24
训练 class5 模型
<<<<<<<<<<<<<<<<<<<<<<<<<<<<1>>>>>>>>>>>>>>>>>>>>>>>>>>>>
训练集MSE:0.02446146123111248
<<<<<<<<<<<<<<<<<<<<<<<<<<<<2>>>>>>>>>>>>>>>>>>>>>>>>>>>>
训练集MSE:0.021323131397366524
<<<<<<<<<<<<<<<<<<<<<<<<<<<<3>>>>>>>>>>>>>>>>>>>>>>>>>>>>
训练集MSE:0.02067386545240879
<<<<<<<<<<<<<<<<<<<<<<<<<<<<4>>>>>>>>>>>>>>>>>>>>>>>>>>>>
训练集MSE:0.020458059385418892
<<<<<<<<<<<<<<<<<<<<<<<<<<<<5>>>>>>>>>>>>>>>>>>>>>>>>>>>>
训练集MSE:0.020356489345431328
```



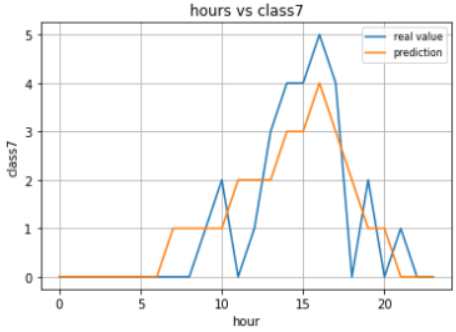
In [69]:
```
model6=LSTM()

train_seq_label = create_train_sequence(train_data, 30,1)
print('The total number of train windows is',len(train_seq_label))
val_seq_label = create_val_sequence(train_data,val_data,30,1)
print('The total number of validation windows is',len(val_seq_label))
test_seq_label = create_test_sequence(train_data,val_data,test_data, 30,1)
print('The total number of validation windows is',len(test_seq_label))

train_model(train_seq_label,'class6', model6,0.005)
y_hat_i,y_ture_i = test_model(test_seq_label,'class6', model6)
y_hat[6]=y_hat_i
y_ture[6]=y_ture_i
```

```
The total number of train windows is 6450
The total number of validation windows is 48
The total number of validation windows is 24
训练 class6 模型
<<<<<<<<<<<<<<<<<<<<<<<<<<<<1>>>>>>>>>>>>>>>>>>>>>>>>>>>>
训练集MSE:0.022914081811904907
<<<<<<<<<<<<<<<<<<<<<<<<<<<<2>>>>>>>>>>>>>>>>>>>>>>>>>>>>
训练集MSE:0.014475001953542233
<<<<<<<<<<<<<<<<<<<<<<<<<<<<3>>>>>>>>>>>>>>>>>>>>>>>>>>>>
训练集MSE:0.014387698844075203
<<<<<<<<<<<<<<<<<<<<<<<<<<<<4>>>>>>>>>>>>>>>>>>>>>>>>>>>>
训练集MSE:0.013783998787403107
<<<<<<<<<<<<<<<<<<<<<<<<<<<<5>>>>>>>>>>>>>>>>>>>>>>>>>>>>
训练集MSE:0.013901213183999062
```



In [78]:
```
model7=LSTM()

train_seq_label = create_train_sequence(train_data, 90,1)
print('The total number of train windows is',len(train_seq_label))
val_seq_label = create_val_sequence(train_data,val_data,90,1)
print('The total number of validation windows is',len(val_seq_label))
test_seq_label = create_test_sequence(train_data,val_data,test_data, 90,1)
print('The total number of validation windows is',len(test_seq_label))

train_model(train_seq_label,'class7', model7,0.001)
y_hat_i,y_ture_i = test_model(test_seq_label,'class7', model7)
y_hat[7]=y_hat_i
y_ture[7]=y_ture_i
```

```
The total number of train windows is 6390
The total number of validation windows is 48
The total number of validation windows is 24
训练 class7 模型
<<<<<<<<<<<<<<<<<<<<<<<<<<<<1>>>>>>>>>>>>>>>>>>>>>>>>>>>>
训练集MSE:0.04367522895336151
<<<<<<<<<<<<<<<<<<<<<<<<<<<<2>>>>>>>>>>>>>>>>>>>>>>>>>>>>
训练集MSE:0.02206074632704258
<<<<<<<<<<<<<<<<<<<<<<<<<<<<3>>>>>>>>>>>>>>>>>>>>>>>>>>>>
训练集MSE:0.02069973386824131
<<<<<<<<<<<<<<<<<<<<<<<<<<<<4>>>>>>>>>>>>>>>>>>>>>>>>>>>>
训练集MSE:0.019750947132706642
<<<<<<<<<<<<<<<<<<<<<<<<<<<<5>>>>>>>>>>>>>>>>>>>>>>>>>>>>
训练集MSE:0.019440559670329094
```



7.4 传统机器学习代码

In [1]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
var_origin = pd.read_csv('./data/bike/basic_model_df.csv')
var_origin = var_origin.drop(columns=['Unnamed: 0'])
var_origin.head(5)
```

Out[2]:

	cluster	month	day	weekday	hour	demands
0	0	1	1	1	0	2
1	0	1	1	1	1	1
2	0	1	1	1	2	2
3	0	1	1	1	3	0
4	0	1	1	1	4	0

In [82]:
```
from sklearn.preprocessing import MinMaxScaler

X = var_origin.drop(labels=['demands'], axis=1).copy()
Y = var_origin['demands'].values.copy()
test = var_origin[(var_origin['month']==9)&(var_origin['day']==30)]
test_index = var_origin[(var_origin['month']==9)&(var_origin['day']==30)].index
train = var_origin.drop(index = test_index,axis=1).reset_index(drop=True)

scaler_X = MinMaxScaler(feature_range=(0, 1))
scaler_Y = MinMaxScaler(feature_range=(0, 1))

scaler_X.fit(X)
scaler_Y.fit(Y.reshape(-1,1))

x_train = scaler_X.transform(train.drop(labels=['demands'], axis=1).copy())
y_train = scaler_Y.transform(train['demands'].values.copy().reshape(-1,1))

x_test = scaler_X.transform(test.drop(labels=['demands'], axis=1).copy())
y_test = scaler_Y.transform(test['demands'].values.copy().reshape(-1,1))
```

In [65]:
```
from matplotlib import pyplot as plt
def show_graph(y_hat,y):
    clusters = [i for i in range(8)]
    plt.figure(figsize=(20,8))
    i=1
    for c in clusters:
        plt.subplot(2, 4, i)
        plt.plot(np.arange(24),y[c*24:c*24+24],label='real value')
        plt.plot(np.arange(24),y_hat[c*24:c*24+24],label='prediction')
        plt.title(c, y=0.5, loc='right')
        plt.legend(loc='upper right',fontsize=8)
        i += 1
    plt.show()
```

决策回归树(手写)

In [ ]:
```
# 决策树
import math

class DecisionTree:
    def __init__(self,depth=99):
        self.depth = depth

    # 计算熵
    def infor_entropy(self,data):
        label = pd.Series(data)
        labelCount = label.value_counts()
        prob = labelCount/np.sum(labelCount)
        Entropy = 0
        for p in prob:
            Entropy = Entropy - (p*math.log2(p))
        return Entropy


    # 寻找最佳的划分方式
    def find_best_feature(self,x,y):
        baseEntropy = self.infor_entropy(y)
        bestInfoGain = 0.0
        col_selected = -1

        for col in x.keys():
            featurelist = x[col]
            uniqueFeature = featurelist.unique()
            newEntropy = 0
            for value in uniqueFeature:
                splitDataSet_y = y[x[col]==value]
                prob = len(splitDataSet_y) / len(y)
                newEntropy += prob * self.infor_entropy(splitDataSet_y)
            infoGain = baseEntropy - newEntropy # 信息增益
            if infoGain >= bestInfoGain:
                bestInfoGain = infoGain
                col_selected = col
        return col_selected

    def majorlabel(self, classlist):
        class_counts = classlist.value_counts()
        most_frequent = class_counts.idxmax()
        return most_frequent
```

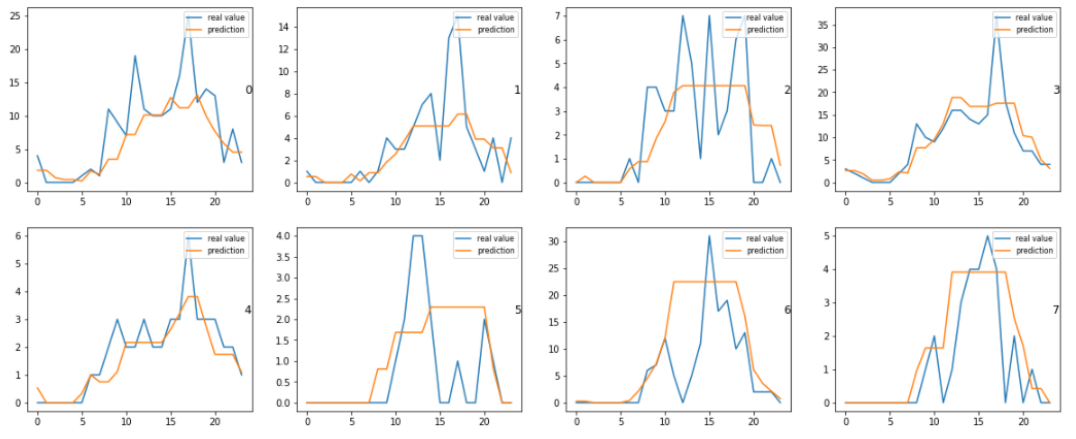
```
    def createTree(self,x,y,mydepth):

        # 剪枝1：类别完全相同则停止划分
        if len(y.value_counts()) == 1:
            return y.iloc[0]

        # 剪枝2：若遍历完所有特征值仍无法完全分类，则返回出现次数最多的
        if len(x.keys())==0:
            return self.majorlabel(y)

        # 剪枝3：层数达到设定上限
        if (mydepth>self.depth):
            return self.majorlabel(y)

        # 选择最好的数据划分方式
        bestFeature = self.find_best_feature(x,y)
        myTree = {bestFeature:{}}
        featurevalue_x = x[bestFeature]
        uniqueFeature_x = featurevalue_x.unique()

        for value in uniqueFeature_x:
            sub_x = x[x[bestFeature]==value].copy()
            sub_x = sub_x.drop(labels=[bestFeature], axis=1)
            sub_y = y[x[bestFeature]==value].copy()
            myTree[bestFeature][value] = self.createTree(sub_x,sub_y,mydepth+1)
        return myTree

    def predict(self,tree,x_test, x_train, y_train):
#     key1:当前属性名称
#     key2:当前属性值
        for key1 in tree.keys():
            secondDict = tree[key1]
            for key2 in secondDict.keys():
                if x_test.loc[key1] == key2:
                    if type(secondDict[key2]).__name__ == "dict":
                        sub_x = x_train[x_train[key1]==key2].copy()
                        sub_y = y_train[x_train[key1]==key2].copy()
                        return self.predict(secondDict[key2], x_test,sub_x,sub_y)
                    else:
                        return secondDict[key2]

            classlabel = self.majorlabel(y_train)

        return classlabel
```

```
    def fit(self, x_train, y_train):
        self.x = x_train
        self.y = y_train
        self.tree = self.createTree(self.x, self.y, 0)

    def score(self, x_test, y_test):
        y_pred=[]

        for i in range(x_test.shape[0]):
            label = self.predict(self.tree, x_test.iloc[i,:],self.x,self.y )
            y_pred.append(label)
        score_list = list(map(lambda x,y: 1 if x==y else 0, y_pred, y_test.values))
        score = round(sum(score_list)/len(score_list), 2)
        return score, y_pred
```

In [85]:
```
param_grid={'max_depth':[i for i in range(1,20)]}
myTree = GridSearchCV(DecisionTree(),param_grid,cv=10)
myTree.fit(x_train, y_train)

y_hat = myTree.predict(x_train)
y_hat_pred = myTree.predict(x_test)

print('Best_param,{}'.format(myTree.best_params_))

print("Train MSE:{:.8f}".format((mean_squared_error(scaler_Y.inverse_transform(y_hat.reshape(-1,1)), scaler_Y.inverse_transform(y_
print("Train MAE:{:.8f}".format((mean_absolute_error(scaler_Y.inverse_transform(y_hat.reshape(-1,1)), scaler_Y.inverse_transform(y
print("Test MSE:{:.8f}".format((mean_squared_error(scaler_Y.inverse_transform(y_hat_pred.reshape(-1,1)), scaler_Y.inverse_transfor
print("Test MAE:{:.8f}".format((mean_absolute_error(scaler_Y.inverse_transform(y_hat_pred.reshape(-1,1)), scaler_Y.inverse_transfo
```

```
Best_param,{'max_depth': 9}
Train MSE:6.99760376
Train MAE:1.56067594
Test MSE:15.43411289
Test MAE:1.93585134
```

In [86]:
```
show_graph(scaler_Y.inverse_transform(y_hat_pred.reshape(-1,1)),scaler_Y.inverse_transform(y_test.reshape(-1,1)))
```

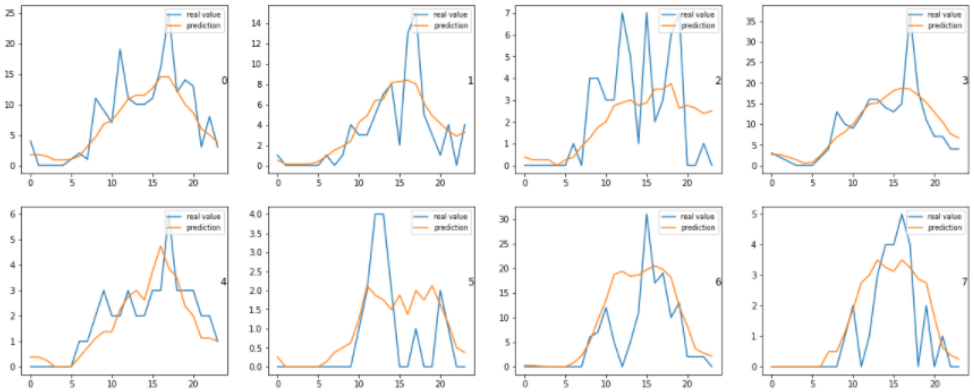


KNN

In [93]:
```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error

param_grid={'n_neighbors':[i for i in range(2,12,2)]}
knn =GridSearchCV(KNeighborsRegressor(),param_grid,cv=10)
knn.fit(x_train,y_train)

y_hat = knn.predict(x_train)
y_hat_pred = knn.predict(x_test)

print('Best_param:{}'.format(knn.best_params_))

print("Train MSE:{:.8f}".format((mean_squared_error(scaler_Y.inverse_transform(y_hat.reshape(-1,1)), scaler_Y.inverse_transform(y_train.reshape(
print("Train MAE:{:.8f}".format((mean_absolute_error(scaler_Y.inverse_transform(y_hat.reshape(-1,1)), scaler_Y.inverse_transform(y_train.reshape
print("Test MSE:{:.8f}".format((mean_squared_error(scaler_Y.inverse_transform(y_hat_pred.reshape(-1,1)), scaler_Y.inverse_transform(y_test.resha
print("Test MAE:{:.8f}".format((mean_absolute_error(scaler_Y.inverse_transform(y_hat_pred.reshape(-1,1)), scaler_Y.inverse_transform(y_test.resh
```

```
Best_param:{'n_neighbors': 8}
Train MSE:6.90026556
Train MAE:1.64149625
Test MSE:11.73714193
Test MAE:1.84375000
```

In [94]:
```
show_graph(scaler_Y.inverse_transform(y_hat_pred.reshape(-1,1)),scaler_Y.inverse_transform(y_test.reshape(-1,1)))
```

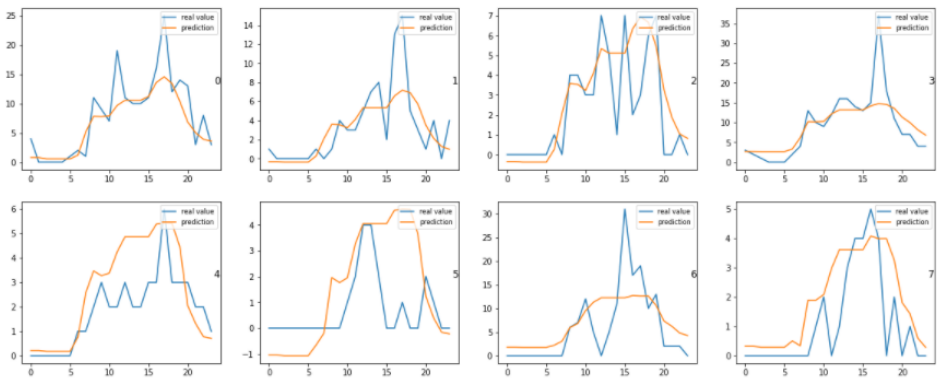


GBDT

In [87]:
```
from sklearn.ensemble import GradientBoostingRegressor

param_grid = {'n_estimators':[50,80,100,300]}

## 随机森林
GBDT = GridSearchCV(GradientBoostingRegressor(),param_grid,  cv=2,scoring='accuracy')
GBDT.fit(x_train,y_train)

y_hat = GBDT.predict(x_train)
y_hat_pred = GBDT.predict(x_test)

print('Best_param:{}'.format(GBDT.best_params_))

print("Train MSE:{:.8f}".format((mean_squared_error(scaler_Y.inverse_transform(y_hat.reshape(-1,1)), scaler_Y.inverse_transform(y_train.reshape(
print("Train MAE:{:.8f}".format((mean_absolute_error(scaler_Y.inverse_transform(y_hat.reshape(-1,1)), scaler_Y.inverse_transform(y_train.reshape
print("Test MSE:{:.8f}".format((mean_squared_error(scaler_Y.inverse_transform(y_hat_pred.reshape(-1,1)), scaler_Y.inverse_transform(y_test.resha
print("Test MAE:{:.8f}".format((mean_absolute_error(scaler_Y.inverse_transform(y_hat_pred.reshape(-1,1)), scaler_Y.inverse_transform(y_test.resh
```

```
Best_param:{'n_estimators': 80}
Train MSE:10.02919221
Train MAE:2.02142982
Test MSE:11.33727501
Test MAE:2.02056170
```

In [88]:
```
show_graph(scaler_Y.inverse_transform(y_hat_pred.reshape(-1,1)),scaler_Y.inverse_transform(y_test.reshape(-1,1)))
```

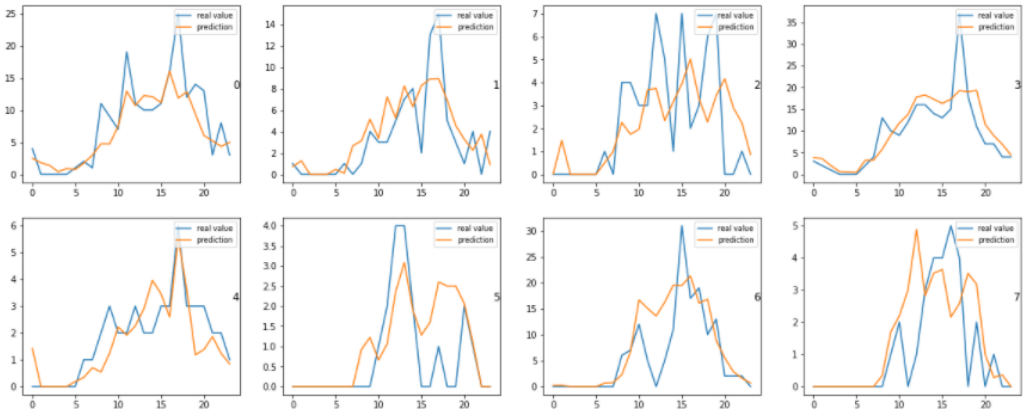


随机森林

In [89]:
```
from sklearn.ensemble import RandomForestRegressor

param_grid = {"n_estimators":[50,80,100,300]}

# 随机森林
clf = GridSearchCV(RandomForestRegressor(),param_grid,  cv=10 ,scoring='accuracy')
clf.fit(x_train,y_train.ravel())

y_hat = clf.predict(x_train)
y_hat_pred = clf.predict(x_test)

print('Best_param:{}'.format(clf.best_params_))

print("Train MSE:{:.8f}".format((mean_squared_error(scaler_Y.inverse_transform(y_hat.reshape(-1,1)), scaler_Y.inverse_transform(y_train.reshape(
print("Train MAE:{:.8f}".format((mean_absolute_error(scaler_Y.inverse_transform(y_hat.reshape(-1,1)), scaler_Y.inverse_transform(y_train.reshape
print("Test MSE:{:.8f}".format((mean_squared_error(scaler_Y.inverse_transform(y_hat_pred.reshape(-1,1)), scaler_Y.inverse_transform(y_test.resha
print("Test MAE:{:.8f}".format((mean_absolute_error(scaler_Y.inverse_transform(y_hat_pred.reshape(-1,1)), scaler_Y.inverse_transform(y_test.resh
```

```
Best_param:{'n_estimators': 50}
Train MSE:0.95086135
Train MAE:0.57732230
Test MSE:10.62777500
Test MAE:1.88187500
```

In [90]:
```
show_graph(scaler_Y.inverse_transform(y_hat_pred.reshape(-1,1)),scaler_Y.inverse_transform(y_test.reshape(-1,1)))
```



xgboost

In [91]:
```
from xgboost import XGBRegressor

param_grid = {"n_estimators":[50,80,100,300]}

# 随机森林
XGBR = GridSearchCV(XGBRegressor(),param_grid,  cv=10 ,scoring='accuracy')
XGBR.fit(x_train,y_train.ravel())

y_hat = XGBR.predict(x_train)
y_hat_pred = XGBR.predict(x_test)

print('Best_param:{}'.format(XGBR.best_params_))

print("Train MSE:{:.8f}".format((mean_squared_error(scaler_Y.inverse_transform(y_hat.reshape(-1,1)), scaler_Y.inverse_transform(y_train.reshape(
print("Train MAE:{:.8f}".format((mean_absolute_error(scaler_Y.inverse_transform(y_hat.reshape(-1,1)), scaler_Y.inverse_transform(y_train.reshape
print("Test MSE:{:.8f}".format((mean_squared_error(scaler_Y.inverse_transform(y_hat_pred.reshape(-1,1)), scaler_Y.inverse_transform(y_test.resha
print("Test MAE:{:.8f}".format((mean_absolute_error(scaler_Y.inverse_transform(y_hat_pred.reshape(-1,1)), scaler_Y.inverse_transform(y_test.resh
```

```
Best_param:{'n_estimators': 80}
Train MSE:5.23166429
Train MAE:1.42731087
Test MSE:11.36940647
Test MAE:1.89084938
```

In [92]:
```
show_graph(scaler_Y.inverse_transform(y_hat_pred.reshape(-1,1)),scaler_Y.inverse_transform(y_test.reshape(-1,1)))
```