队伍编号	1758
赛道	A

基于 AI 算法的基站流量预测

摘 要

随着移动通信技术的发展，4G、5G 给人们带来了极大便利。移动互联网的飞速发展，使得移动流量呈现爆炸式增长，从而基站的流量负荷问题变得越来越重要。本文基于数据分析和 AI 算法旨在解决由于每个基站的流量高峰和低谷的时段各不相同而造成的能量资源浪费和由于基站的整体流量增加而干扰物理扩容规划的问题。

本文首先将原始数据拆分为题目所提供的提交文件包含的小区数据，然后对数据缺失值、异常值和重复值分别进行处理，得到问题需要的洁净数据。

针对问题一，本文首先将预处理之后的单变量时间序列数据转化为有监督问题，再将数据集划分为训练和验证数据。然后使用 ARMA 作为 benchmark 模型，再使用 Facebook 的 Prophet 模型、传统的树模型 LightGBM、长短时记忆网络 LSTM 和改进的深度前馈时变网络 ForecastNet 分别作为对比模型，最后我们将 5 个模型分别对所有基站小区的验证集进行误差测试，取所有小区的平均验证误差，得到每个小区的最优验证模型(RMSE 误差最小)对未来一周进行预测，并保存为提交文件 Test1。

针对问题二，本文首先将待预测小区的 小时级业务量合并为天级业务量数据，再将天级数据缺失严重的部分进行同周期均值填补，然后再使用问题一得到的预测效果最好的 Prophet 和 ForecastNet 分别进行长期单变量时间序列预测来对比预测结果，最后选取 Prophet 作为本题最终的预测模型来预测 2020 年 11 月 1 日至 11 月 25 日的业务量。而对数据集中存在的空数据小区，本文使用同一天的其他小区预测的业务量数据计算并取均值来填补，最终保存为提交文件 Test2。

针对问题三，我们基于前面对于数据的分析和理解，发现不少基站小区存在极少或 0 业务量现象，分析原因可能是基站小区建设在偏远或人群流量极少的地方，这样就导致总体基站小区的利用率较低。所以基于本题的数据可以研究基站小区之间的空间合作关系，目的是为了得到空间合作关系微弱的小区，即偏远或人群流量极少的基站小区。我们采用格兰杰因果关系检验、K-mean 算法和相关性检验逐级筛选，最终筛选得到的异簇小区来为运营商的基站建设提供参考和引导。

关键词：Prophet　ForecastNet　LSTM　LightGBM　模型融合

目录

一、 问题分析

随着移动通信技术的发展，4G、5G 给人们带来了极大便利。移动互联网的飞速发展，使得移动流量呈现爆炸式增长，从而基站的流量负荷问题变得越来越重要。本文基于数据分析和 AI 算法旨在解决由于每个基站的流量高峰和低谷的时段各不相同而造成的能量资源浪费和由于基站的整体流量增加而干扰物理扩容规划的问题。

本文首先将原始数据拆分为题目所提供的提交文件包含的小区数据，然后对数据缺失值、异常值和重复值分别进行处理，得到问题需要的洁净数据。

1.1 问题一分析

本文首先将预处理之后的单变量时间序列数据转化为有监督问题，再将数据集划分为训练和验证数据。本题使用 ARMA 作为 benchmark，然后再使用 Facebook 的 Prophet 模型、传统的树模型 LightGBM、长短时记忆网络 LSTM 和改进的深度前馈时变网络 ForecastNet 分别作为对比模型，最后我们将 5 个模型分别对所有基站小区的验证集进行误差测试，取所有小区的平均验证误差，得到每个小区的最优验证模型(RMSE 误差最小)对未来一周进行预测。

1.2 问题二分析

本文首先将待预测小区的小时级业务量合并为天级业务量数据，再将天级数据缺失严重的部分进行同周期均值填补，而对数据集中存在的空数据小区，本文使用同一天的其他小区预测的业务量数据计算并取均值来填补。然后再使用问题一得到的预测效果最好的 Prophet 和 ForecastNet 分别进行长期单变量时间序列预测来对比预测结果，最后选取 Prophet 作为本题最终的预测模型来预测 2020 年 11 月 1 日至 11 月 25 日的业务量。

1.3 问题三分析

我们基于前面对于数据的分析和理解，发现不少基站小区存在极少或 0 业务量现象，分析原因可能是基站小区建设在偏远或人群流量极少的地方，这样就导致总体基站小区的利用率较低。所以基于本题的数据可以研究基站小区之间的空间合作关系，目的是为了得到空间合作关系微弱的小区，即偏远或人群流量极少的基站小区。我们采用格兰杰因果关系检验、K-mean 算法和相关性检验逐级筛选，最终筛选得到的异簇小区来为运营商的基站建设提供参考和引导。

二、 数据预处理

2.1 数据拆分

训练数据中含有 132279 个不同基站小区的各种数据。数据量非常大，由于本地机器的算力限制，我们选择只对题目所提供的提交文件包含的小区进行逐基站小区预测。根据短期验证.csv 和长期验证.csv 两个文件中的小区编号，整理出短期验证训练数据和长期验证的训练数据。

2.2 缺失、重复值处理

2.2.1 缺失值处理

部分小区的原数据存在空值，选择前向填充方法对其进行填补。

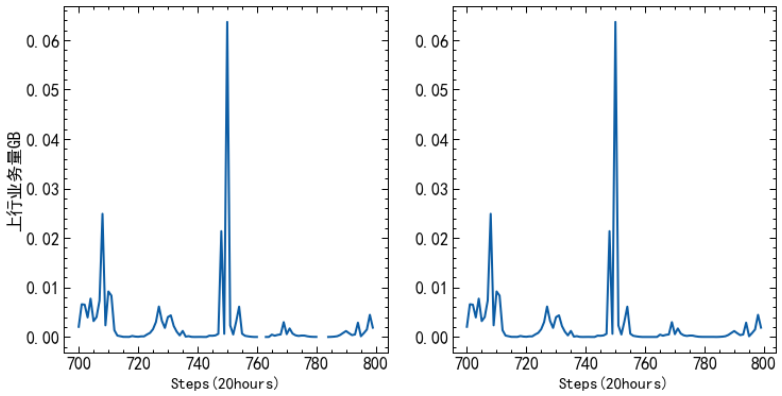

图 1 缺失值填补前(左)与缺失值填补后(右)的部分数据分布

2.2.2 重复值处理

存在某一天的数据量比其他正常天数数据量要多几倍，找出该天数据发现存在 48 条数据，其中 24 条属于重复数据。故而对全部数据进行去重处理。

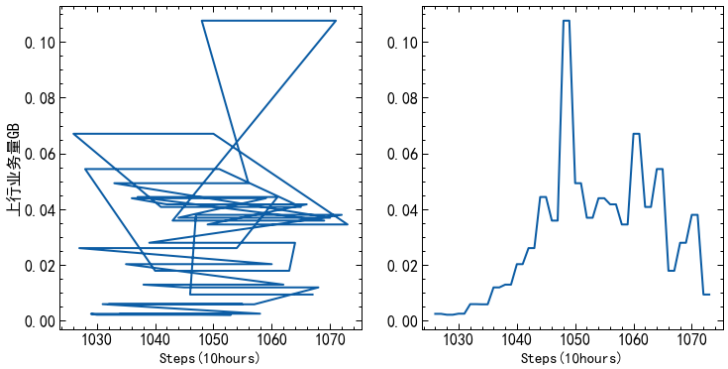

图 2 重复值前(左)与删除重复值后(右)的部分数据分布

2.3 记录每天样本数超出或者不满 24 次/天

按照天数统计，观察某一个小区每一天的流量。可以发现：存在某一天的数据量比其他要少几倍，找出该天数据发现不满 24 条数据。

考虑到数据的充分利用性。我们设置了以 12 为阈值的处理方法，若缺失值大于 12 条，直接剔除一整天的数据(剔除约 2%的天量数据)。如缺失值小于或等于 12 条，选择填充数据(填充约 11%的天量数据)。

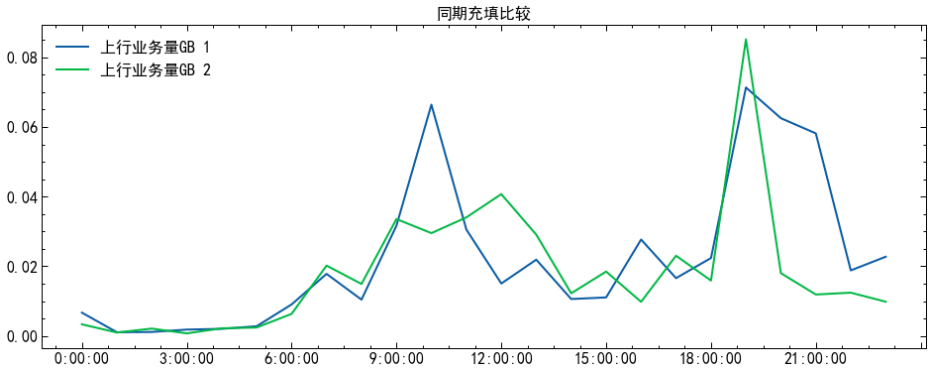

图 3 相邻两天的上行业务量数据分布

在选择填充的方法时，我们绘制了某一个小区 2018 年 3 月 1 日和 2018 年 3 月 2 日的上行业务量 GB 随着时间变化图(如上图 3)，发现不同日期同一时间的数据大多变化不大，而同一日期内数据的时序性比较强。故而采取同一时间后一天的数据向前填充。填充效果如下图 4。

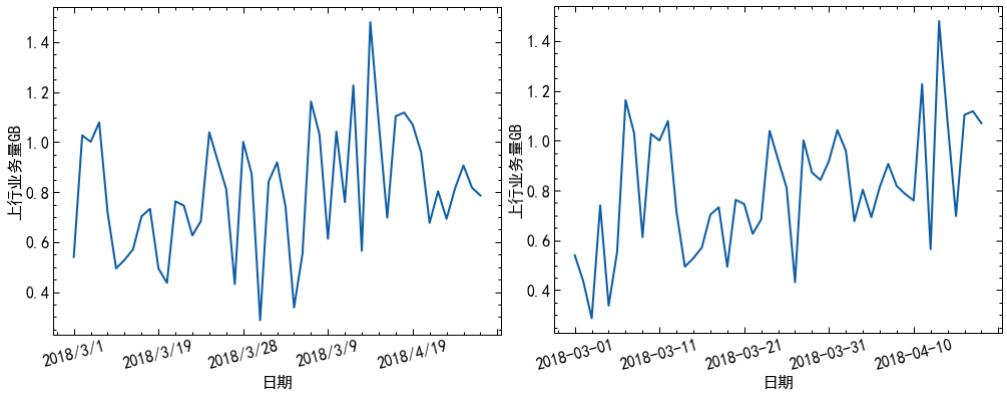

图 4 填补处理后(左)和填补处理前(右)的部分数据分布

2.4 异常值处理

2.4.1 数据文本存在异常

	日期	时间	小区编号	上行业务量GB	下行业务量GB
896	018-04-09	00:00:0	4469	9.398651e-05	7.012272e-05
897	018-04-09	01:00:0	4469	9.398651e-05	2.746582e-07
898	018-04-09	02:00:0	4469	5.702019e-06	1.692772e-05
899	018-04-09	03:00:0	4469	4.577637e-08	2.746582e-07
900	018-04-09	04:00:0	4469	5.802155e-06	1.281166e-05
901	018-04-09	05:00:0	4469	8.458328e-05	2.663307e-04
902	018-04-09	06:00:0	4469	3.983784e-05	2.644539e-05
903	018-04-09	07:00:0	4469	1.177187e-04	2.030296e-04

图 5 异常文本样例

将存在异常的文本数据修改为正常日期格式。

2.4.2 极端值处理

随机选取某一小区的上行业务量数据，绘制箱型图(如下图 6)，可以较为直观的发现存在大于箱线图上须的异常数据[1]。

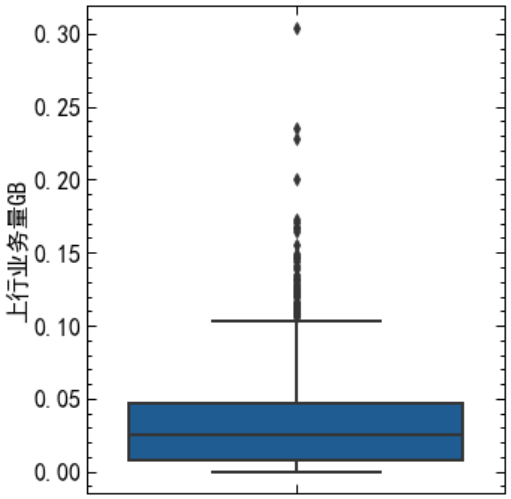

图 6 1014 小区上行业务量数据箱型图分布

从图6我们可以发现1014小区的上行业务量的异常值结点处于0.10附近。对于该小区，上行业务量超过0.10的数据大约占总数据的5%，故而我们选择采取95分位点覆盖异常值(超过95分位点的数据)。可以看到变换后的数据分布(图7)得到显著改善。

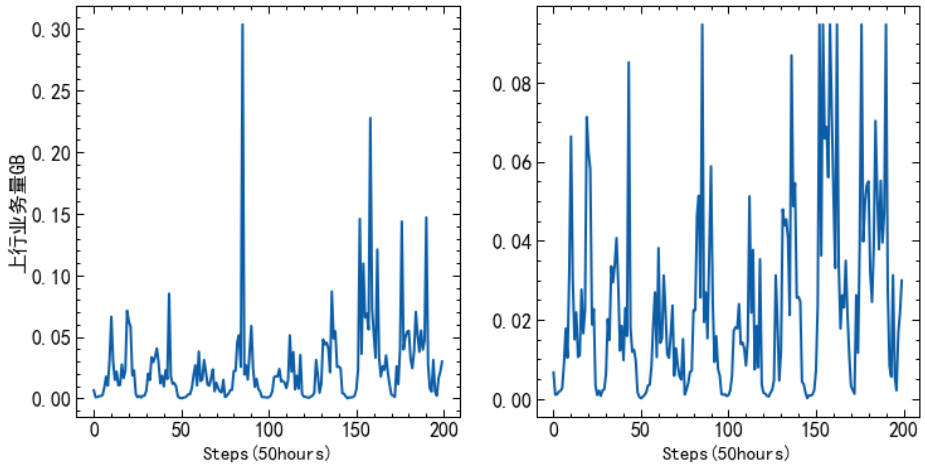

图 7 异常值截断前(左)与异常值截断后(右)的部分数据分布

三、模型的建立与求解

3.1 问题一

3.1.1 问题转化

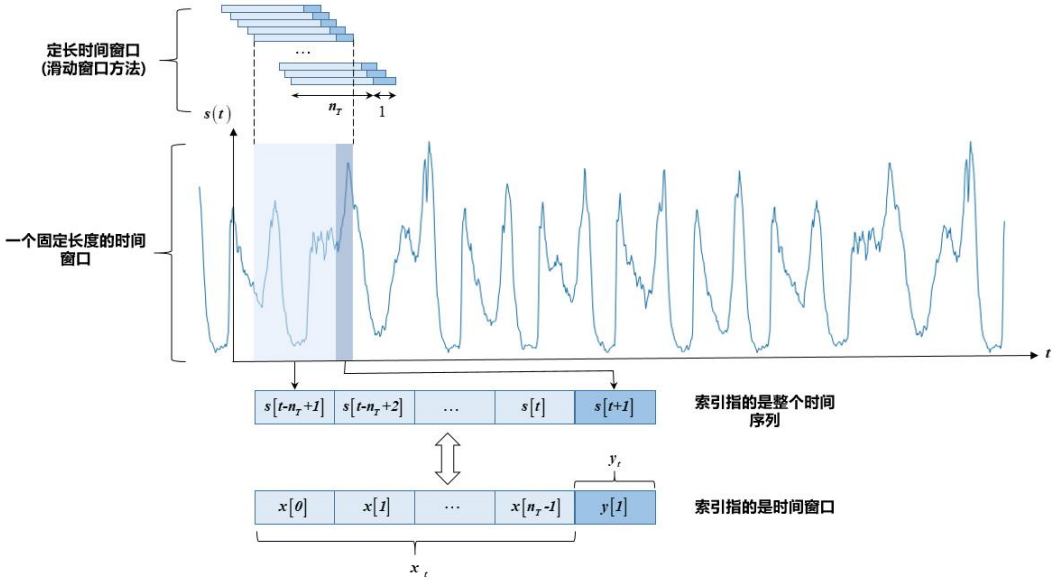

图 8 使用滑动窗口方法将预测问题转换为有监督问题[2]

单变量时间序列 $s=\left[s[0],s[1],\cdots,s[T]\right]$，得到了全部的时间数据。在这项工作中，输入数据作为回归向量呈现给模型的不同预测样本，该回归向量由固定时间滞后数据组成，且与在时间序列上滑动的长度窗口大小 n_T 有关。给定过去值的固定长度视图，预测器 f 旨在预测时间序列的下一个值，被称作单步时间序列预测。在这项工作中，将预测问题作为监督学习问题进行研究。这样，给定在离散时间 t 的输入向量定义为 $x_t=\left[s\left[t-n_T+1\right],\cdots,s\left[t\right]\right]\in IR^{n_T}$，预测问题需要推断下一个测量值 $y_t=\left[s\left[t+1\right]\right]$。

为了简化表示法，我们在时间窗口的参考系统(不是整条时间序列)中表示输入和输出向量[3]。在不失一般性的前提下，在本文的其余部分，我们从 x_t 和 y_t 的内部元素中删除下标 t。引入的符号以及滑动窗口方法如图 8 所示。

3.1.2 数据划分

对于时间序列系列的问题，通常要划分数据来解决才有实际意义，最大的好处就是可以避免模型在训练数据上过度拟合[4]。基于本题，我们通过观察整个和需要预测的数据，选择将 x_t 的最后一周(2018.4.12 - 2018.4.19)即 168 小时作为验证数据，所以我们将 x_t 和 y_t 划分为训练集(Train)、验证集(Valid)和待预测数据集(Test)。本文下面的所有模型都是基于对 Train 进行训练，然后对 Valid 进行验证，最后预测 Test。[5]

3.1.3 ARMA

ARMA 模型(Autoregressive moving average model)全称为自回归滑动平均模型，是研究时间序列的重要方法，由自回归模型(简称 AR 模型)与移动平均模型(简称 MA 模型)为基础共同构成[6]。本文基于 Python 实现 ARMA 传统的统计模型来作为 benchmark 模型。

ARMA 模型原理如下：

自回归 AR(p)模型：

$$x_t = \phi_0 + \sum_{i=1}^{p} \phi_i x_{t-i} + \varepsilon_t$$

自回归模型描述的是当前值与历史值之间的关系。其中 ϕ_0 是常数项；ε_t 被假设为平均数等于 0，标准差等于 σ_ε 的随机误差值；ε_t 被假设为对于任何的 t 都不变。

移动平均 MA(q)模型：

$$x_t = \mu + \varepsilon_t - \sum_{j=1}^{q} \theta_j \varepsilon_{t-j}$$

移动平均模型描述的是自回归部分的误差累计。其中 μ 是序列的均值，$\theta_1, \cdots, \theta_q$ 是参数，$\varepsilon_t, \varepsilon_{t-1}, \cdots, \varepsilon_{t-q}$ 都是白噪声。

ARMA(p,q)模型中包含了 p 个自回归项和 q 个移动平均项，ARMA(p,q)模型可以表示为：

$$x_t = \phi_0 + \sum_{i=1}^{p} \phi_i x_{t-i} + \varepsilon_t - \sum_{j=1}^{q} \theta_j \varepsilon_{t-j}$$

3.1.4 Prophet

Prophet 是 Facebook 2017 年开源的一个时间序列预测的算法[7]，该算法不仅可以处理时间序列存在一些异常值的情况，也可以处理部分缺失值的情形，还能够几乎全自动地预测时间序列未来的走势。从原文论文上的描述来看，Prophet 算法是基于时间序列分解和机器学习的拟合来做的，因此能够在较快的时间内得到需要预测的结果。本文基于 Python 接口实现 Prophet 算法[8]。

Prophet 使用具有三个主要模型组成部分的可分解时间序列模型：趋势，季节性和节假日。它们通过以下等式组合：

$$y(t) = g(t) + s(t) + h(t) + \varepsilon_t$$

其中，$g(t)$是趋势函数，它对时间序列值的非周期性变化进行建模，$s(t)$代表周期性变化(例如，每周和每年的季节性)，$h(t)$代表假期的影响，可能会在一整天或更多天内以不定期的时间表发生。误差项ε_t代表模型不适应的任何特殊变化。

Prophet 所做的事情就是[9]：

(1) 输入已知的时间序列的时间戳和相应的值；
(2) 输入需要预测的时间序列的长度；
(3) 输出未来的时间序列走势；
(4) 输出结果可以提供必要的统计指标，包括拟合曲线，上界和下界等。

3.1.5 LightGBM

GBDT (Gradient Boosting Decision Tree)是机器学习中一个长盛不衰的模型，其主要思想是利用弱分类器(决策树)迭代训练以得到最优模型，该模型具有训练效果好、不易过拟合等优点。GBDT 不仅在工业界应用广泛，通常被用于多分类、点击率预测、回归等任务。在各种数据挖掘竞赛中也是致命武器，据统计 Kaggle 上的比赛有一半以上的冠军方案都是基于 GBDT[10]。而 LightGBM(Light Gradient Boosting Machine)是一个实现 GBDT 算法的框架，支持高效率的并行训练，并且具有更快的训练速度、更低的内存消耗、更好的准确率、支持分布式可以快速处理海量数据等优点[11]。本小节使用的是 LightGBM 做回归监督任务拟合未来流量数据[12]。

(1) 构造时间序列特征

a) 窗口统计特征

统计过去 N 天的属性特征，包括均值、标准差、极值、分位数、偏度、峰度，如图 9 所示。

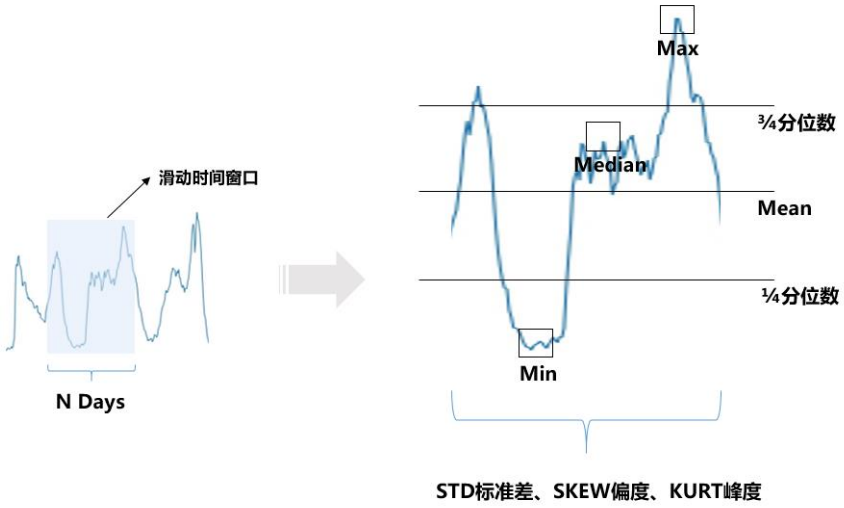

图 9 统计特征构造可视化

结合本题的数据特点，分别选取 24*1、24*2、24*3、24*4、24*5、24*6 的时间窗口数据来提取历史时间信息。

b) 维度特征

小时级特征：一天中哪个时间段(0-24h)

天级特征：一周中的星期几、一年中的第几天

星期级特征：一个月中的第几个星期

c) 统计转换特征

对构造的维度特征进行正弦和余弦变换，得到正余弦周期特征。

d) 高维空间转换特征

对构造的一维的时序窗口统计特征通过 OneHot_Encoding 转化到高维，进而提取时间序列在时间维度上的信息。

(2) 模型的训练和预测

a) 模型选取的参数如下：

```
model = lgb.LGBMRegressor(objective = 'regression',
                          boosting_type='gbdt',
                          num_leaves=32,
                          max_depth=8,
                          learning_rate=0.05,
                          n_estimators=200,
                          subsample=0.8,
                          min_data_in_leaf=350,
                          feature_fraction=0.8,
                          random_state=2020,
                          n_jobs=-1,
                          metric='mse')
```

参数选用亮点：

其中我们使用的损失函数为 'regression' 也就是 L2 正则损失，效果大于 L1 损失。

其次我们使用 'mse' 作为 LGBMRegressor 的评估标准。

最后我们使用了 min_data_in_leaf 参数，该参数设置的值较大可以避免生成一个过深的树，训练时间也优化提升了约 1.5 倍。

b) 五折交叉验证训练模型

本文采用 Kfold 五折交叉验证的方法训练模型[13]，原理如图 10 所示。

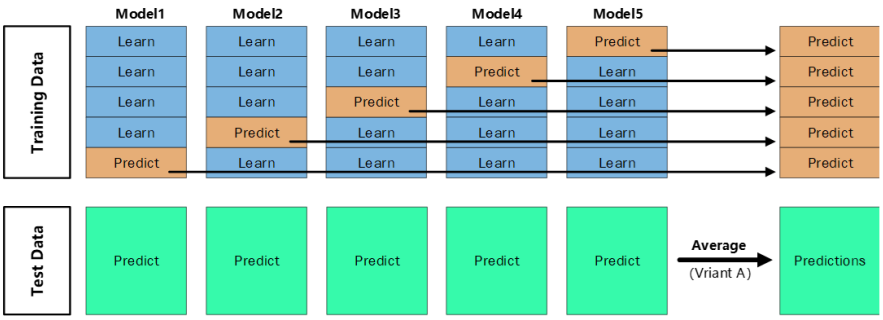

图 10 Kfold 五折交叉验证的原理

由于交叉验证很容易导致数据泄露，所以我们对划分的训练数据进行交叉验证，在验证集上进行测试结果，发现数据并无泄露，然后再使用已有数据对未来数据进行预测。

3.1.6 LSTM

LSTM(Long Short-Term Memory)网络是一种特殊的 RNN 网络，能够解决长期依赖(long-tern dependencies)的问题，由 Hochreiter & Schmidhuber(1997)提出[14]，提出的原因是为了解决普通 RNN 网络梯度消失和爆炸以及更好的预测和分类序列数据等问题，它能挖掘数据中的时序信息以及语义信息，对具有序列特性的数据非常有效[15]。

其基本单元结构如下：

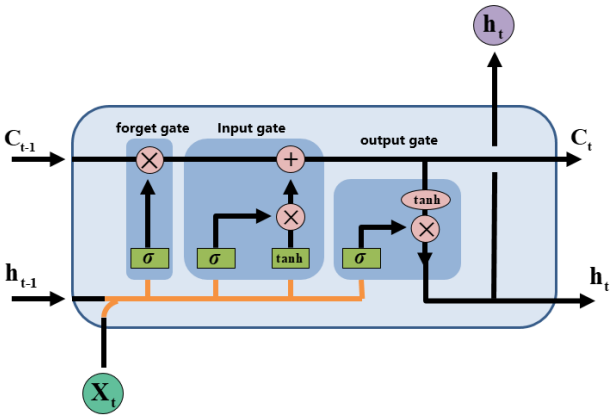

图 11　LSTM 基本单元结构图

其中 C_{t-1} 为上一单元的遗忘元，h_{t-1} 为上一单元的隐藏元，X_t 输入数据，h_t 为输出隐藏元，C_t 为输出遗忘元。

基于本题的数据，我们设计了如下的网络结构：将归一化后的数据输入，经过 LSTM 层，再经过全连接层、Tanh 激活函数、全连接层，最后输出[16]。网络结构图如图 12。

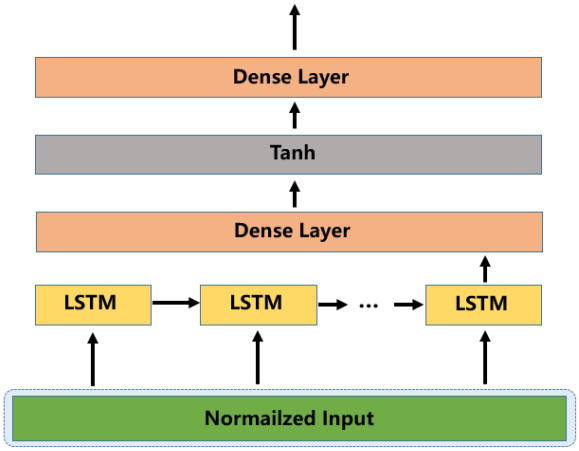

图 12　LSTM 网络结构图

本文的 LSTM 通过 Pytorch 深度学习框架实现完成，因为 Pytorch 具有很强的复现性，损失函数采用平方损失函数 MSE[17]，最后经过参数调整，得出如下 LSTM 网络结构最优参

数如下表：

表 1　LSTM 网络结构最优参数

Hyperparameters	LSTM
Seed	2020
Input_sequence_length	24
Hidden_Layers	64
Activation_Functions	Relu
Optimization_Algorithm	Adam
Learning_Rate	0.001
Batch_Size	1
Epochs	20
Device	GPU

3.1.7 ForecastNet

循环和卷积神经网络是深度学习文献中用于时间序列预测的最常见架构。这些网络通过在时间或空间上重复使用具有固定参数的一组固定架构来使用参数共享[18]。结果是整个体系结构是时不变的(在空间域中是不变的)或固定的，从而导致预测结果较差而且具有滞后性。所以本文借鉴了 ICML2020 新发表的一篇《ForecastNet：A Time-Variant Deep Feed-Foorward Neural Network Architecture for Multi-Step-Ahead Time-SeriesForecasting》[19]来解决这个问题。文章作者认为时间不变性会降低执行多步提前预测的能力，基于这种背景下提出了 ForecastNet，它使用深度前馈体系结构来提供时变模型。 ForecastNet 的另一个新颖之处是交错输出，作者展示了它们有助于缓解消失的梯度。文章中已证明 ForecastNet 在多个数据集上的表现优于统计和深度学习基准模型，而且经过我们的实验，通过调整和改良网络证明该网络结构基于本文的数据，结果表现优异。

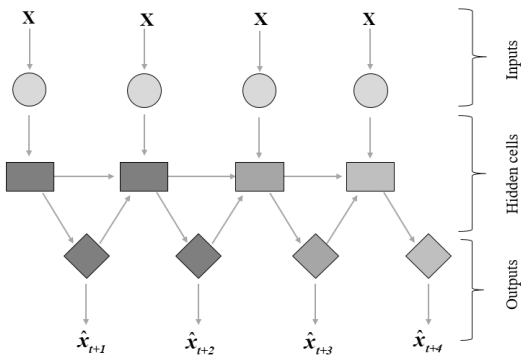

图 13 提供给定的预测 $\hat{x}_{t+1:t+4}$ 的一般 ForecastNet 结构

如图 13 所示，给出 $X = x_{t-nI+1:t}$ 输入(圆圈)。隐藏的单元格(正方形)包含某种形式的前馈神经网络结构。每个隐藏的单元格和输出均以不同的阴影表示，以指示序列上的异质性。

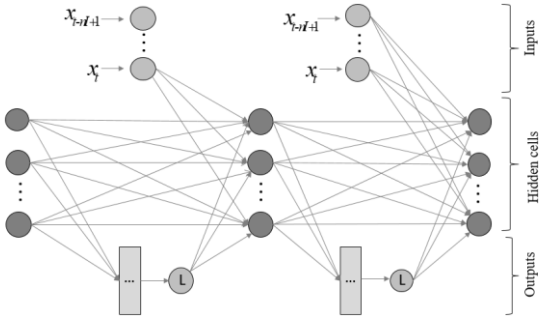

图 14 简单形式的 ForecastNet 的详细图

图 14 是一个简单形式的 ForecastNet 的结构，其中 Hidden 层是用来模拟时间序列动态的，隐藏单元之间的连接来模拟局部动态，单元之间的连接模拟长期动态，在 Hidden 层中具有多个紧密连接的全连接层。然后我们借鉴了注意力机制 **Attention** 的权重分配，将交错输出层里的 Dense 结构分配权重参数，最后合并交错输出和 Hidden 层。尽管每个序列步骤的架构都相同，但是权重会有所不同(即它们不是递归的)。

原文的论文最后输出采用正态分布或高斯分布的形式，但是我们采用了原文的变体结构之一：具有线性输出层而不是混合密度输出层的结构。原因有以下两点：

(1) 输出采用正态分布或高斯分布的形式会导致文章所说的结果不确定性

(2) 采用线性输出层，整个网络运行速度较快

最后我们使用 Keras 深度学习框架实现完成 ForecastNet，损失函数采用平方损失函数 MSE[20]，经过参数调整，得出如下 ForecastNet 网络结构最优参数如下表：

表 2　ForecastNet 网络结构最优参数

Hyperparameters	**ForecastNet**
Seed	20
Input_sequence_length	24
Hidden_Layers	45
Activation_Functions	Relu
Optimization_Algorithm	Adam
Learning_Rate	0.001
Batch_Size	32
Epochs	32
Device	CPU

3.1.8 结果

为了更清晰的查看模型的预测效果，我们选取小区号为 1014 的数据来验证，将所有模型对 Train 进行训练，然后在 Vaild 上预测，每个模型的效果如下表 3 所示。

表 3 所有模型对 1014 小区的预测结果

	上行业务量 GB		下行业务量 GB	
	RMSE	**R2**	**RMSE**	**R2**
ARMA	0.03460	0.06090	0.23112	0.03914
Prophet	**0.02740**	**0.33635**	**0.19652**	**0.25645**
LightGBM	0.03450	0.05461	0.23202	0.04728
LSTM	0.03163	0.11572	0.22723	0.00592
ForecastNet	0.03063	0.16871	0.21795	0.07585

为了直观起见，我们对比了所有模型在 Valid 上一周的预测结果可视化在一张图里，如图 15 图 16 所示。在图 17 中，我们量化了最佳预测值(Prophet)与实际之间的差异。线越细，则预测越接近真实数据。

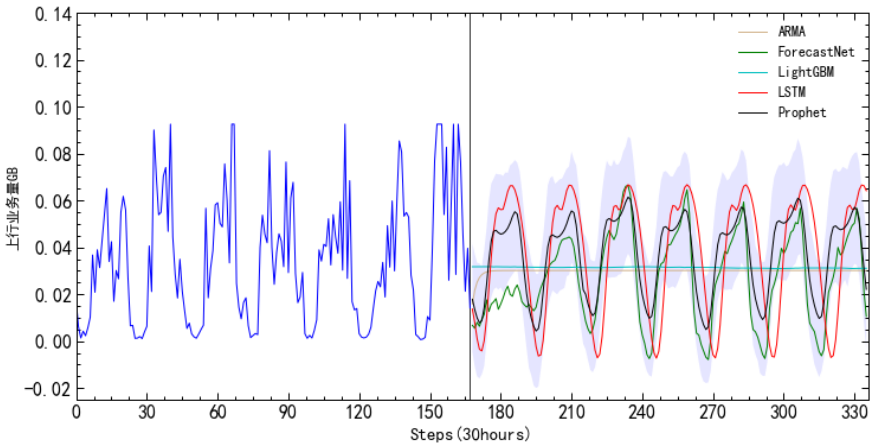

图 15 所有模型对 Valid 上行业务量的预测性能

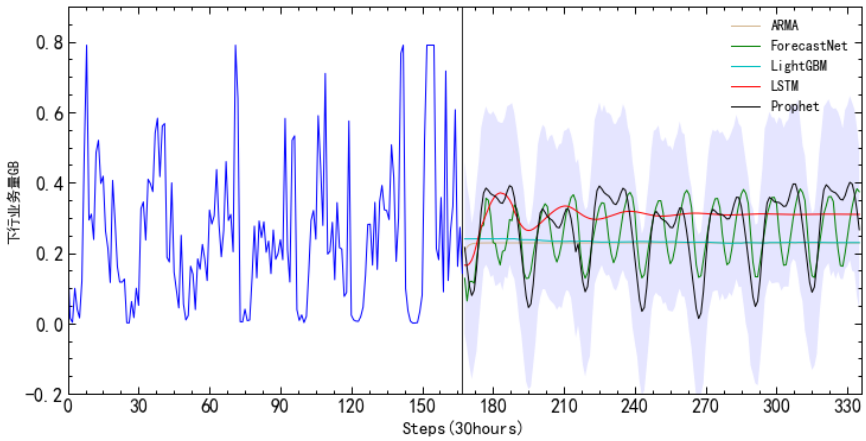

图 16 所有模型对 Valid 下行业务量的预测性能

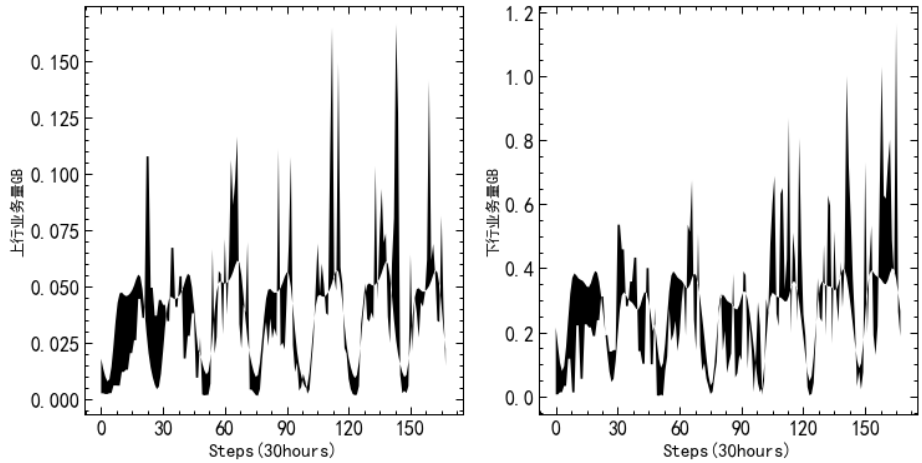

图 17 最佳模型的预测(Prophet)与实际之间的差异

在上述 3 个图中，我们对未来一周的预测连接起来以具有更宽的时间跨度，并评估模型的预测能力。我们观察到模型能够生成一个预测，该预测可以正确地模拟业务量曲线的总体趋势，但是无法预测陡峭的峰值。这可能来自使用 MSE 作为优化指标的设计选择，会阻止深层模型预测高峰，因为对大误差的惩罚很大，因此，根据该指标预测更低和更平滑的函数会带来更好的性能。或者，某些峰值可能仅表示由于特定时间行为而产生的噪声，因此根据模型定义无法预测。

3.1.9 模型融合

最后，我们将 5 个模型分别对所有小区的 Valid 进行验证，然后取所有基站小区的平均误差，结果如表 4，最后取每个小区的最优验证模型(RMSE 误差最小)对未来的 Test 进行预测，保存为提交文件。融合原理如图 18 所示。

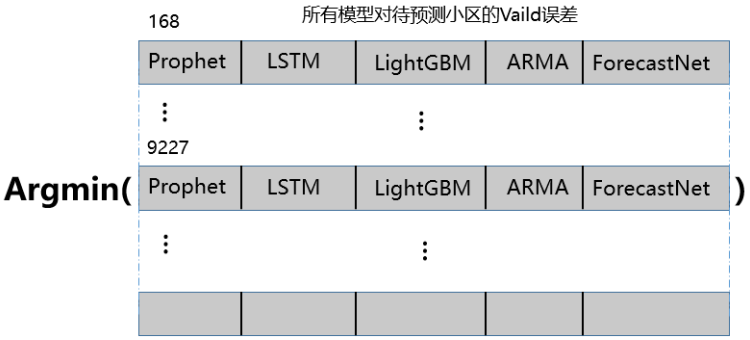

图 18 模型融合原理

表 4 所有模型对所有小区的预测结果

	上行业务量 GB(mean_rmse)	下行业务量 GB(mean_rmse)
ARMA	0.04240	0.29238
Prophet	**0.03234(-0.237%)**	**0.24983(-0.146%)**
LightGBM	0.04232(-0.002%)	0.29193(-0.002%)
LSTM	0.04765(+0.124%)	0.31290(+0.070%)
ForecastNet	0.03315(-0.218%)	0.25891(-0.114%)

从表 4 可以看出 Prophet 对于该题的预测能力最好，原因是因为本题是具有明显趋势性、周期性的单变量时间序列，并且 Prophet 对处理一些样本的异常点和节假日时间也有相应的算法解决。而 ForecastNet 次之，可以发现 ForecastNet 与最好的 Prophet 模型相差很小，也进一步说明了该深度神经网络对周期性时间序列的适应性。预测能力最差的是 LSTM，导致该网络预测效果较差的原因可能是本题的数据存在过多陡峭的峰值，数据的复杂度过高。

由于移动网络话务量存在明显的潮汐效应，但大部分通信基站设备却始终保持持续运行状态，能耗并没有随话务量动态调整，形成浪费。通过题目所给的基站小区的历史话务流量数据(2018 年 3 月 1 日至 4 月 19 日)，对基站小区流量进行分析预测，获得下一阶段(4 月 20 日至 4 月 26 日)基站小区话务流量预测值，从而可以对预测每个小区的低谷进行关断门限和时间进行精准设置，在确保基站覆盖范围不变，对网络 KPI 无影响的条件下，可以使基站能耗降低，大幅降低通信系统运营费用。

3.2 问题二

3.2.1 数据准备

本文中所采用的训练数据来自于部分小区 2018 年 3 月 1 日至 4 月 19 日的小时级流量数据，这些数据中包含日期、时间、小区编号、上行业务量 GB 和下行业务量 GB 信息。需要预测部分小区在 2020 年 11 月 1 日至 11 月 25 日每天的总的上行和下行流量。预测基线与训练数据的小时流量数据不同，故而将数据如下图 19 进行求和统计。

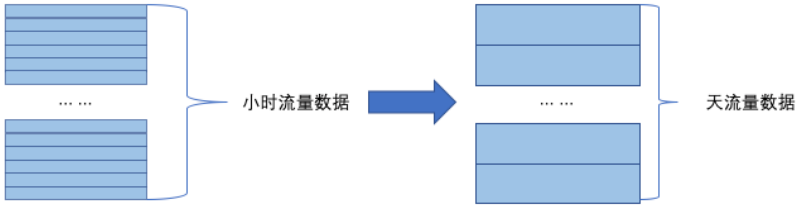

图 19 流量数据求和转化

在对原始数据分析的过程中，发现数据中存在许多异常的数据。这些噪声数据将会不利于模型的训练，需要对原始数据进行预处理。

3.2.2 异常数据的分析处理

1. 编号为 133362、133063、132941、132733、132518、134060、132455、132913、133137、132504、132665、132410，132957 待预测小区无训练数据。编号为 131724、127824 待预测小区只有一天的训练数据。针对上述小区，我们采用均值替代法[21]，即用同一天的其他小区预测的流量数据计算并取均值来填补。
2. 存在一天内流量记录数据不满 24 条，由于本题的预测基线为一天的流量总和，故而会存在数据过低现象。如图 20：未处理，10079 小区上行业务量部分数据，可以比较清楚的看见 2018 年 3 月 27 日、2018 年 3 月 29 日以及 2018 年 3 月 30 日的上行业务量数据接近 0.1GB，相比其他日期的流量低很多。采用同一天其他小时的业务量均值去填充当天的缺失值，结果如下(图 20：已处理)可以发现业务量差距缩小了很多。

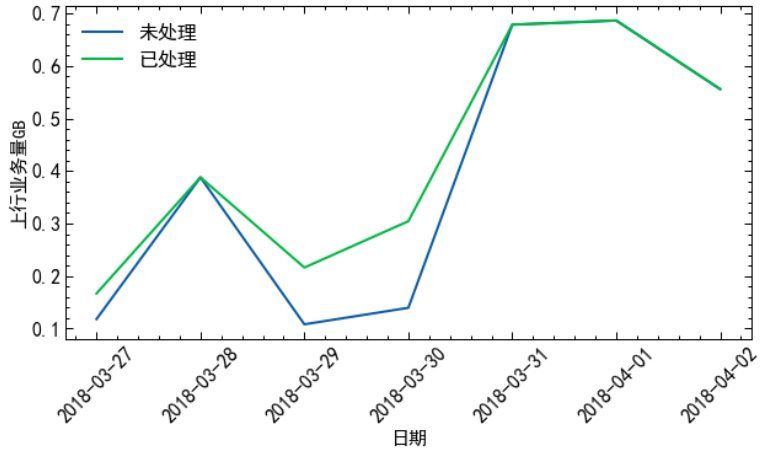

图 20 10079 小区上行业务量部分数据

由于训练和预测数据时间跨度很大，验证数据无法反应预测的情况，故而不选择划分验证集。将所有 Train 作为模型的输入去训练模型，最后预测 Test。

3.2.3 模型预测

由表 4 数据可以发现，Prophet 和 ForecastNet 模型对于回归拟合本题的数据效果较好。

ForecastNet 模型为深度神经网络，其网络结构并不适合对长期时间序列进行预测，原因是滑动时间预测方法会逐步累积误差，预测的时间越久，累积的误差越大，从而导致数据丢失原本信息，向误差方向优化，所以我们没有使用深度神经网络来解决第二问。

Prophet 模型本质上是一种可分解的加法回归模型，即按组件可将模型分解为非周期性变化的趋势项、周或年的季节周期项和节假日效应等[22]。因此 Prophet 模型可以灵活地对各组件的参数进行设置，这些参数值的大小代表着各组件对模型预测结果的贡献度，部分参数说明如表 5 所示。

表 5　Prophet 模型部分参数说明

参数名称	类型	说明
growth	linear/ logistic	模型趋势的增长方式
changepoints	double	指定潜在突变点
yearly prior scale	double	表示模型的年季节性灵活度
weekly prior scale	double	表示模型的周季节性灵活度
daily prior scale	double	表示模型的日季节性灵活度
holidays	Date	可设定节假日

且问题二待预测小区数量高达 2525，Prophet 较于其他模型而言，拟合时间较短。综上考虑采用 Prophet 模型对问题二建模。

预测算法的流程如图 21 所示：

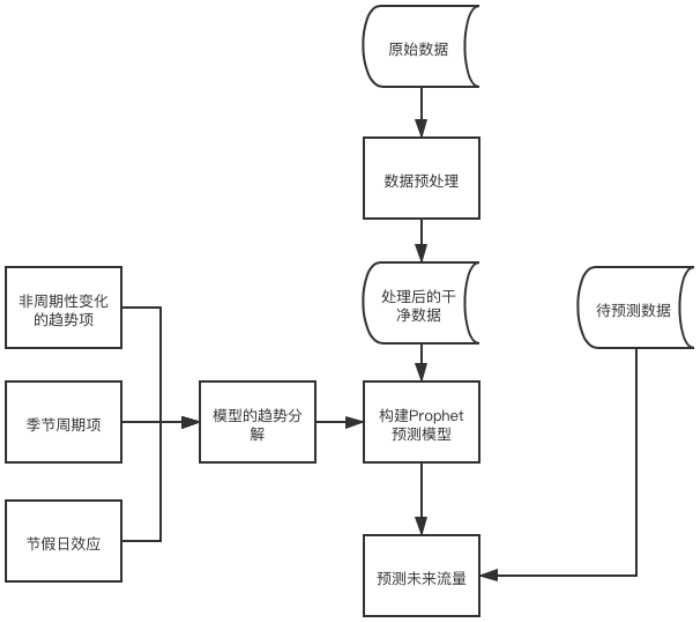

图 21　Prophet 预测算法流程

3.2.4 结果分析

应用 Prophet 模型对给定流量数据进行训练，对预测的总体变化进行组件化分解，最终得到小区在 2020 年 11 月 1 日至 11 月 25 日每天的总的上行和下行流量。

由于训练数据的时间区间范围并不大，部分小区在该区间整体表现为流量下降趋势，导致预测流量的长期变化趋势也为递减趋势，在时间跨度为 2 年的情况下，预测结果为负值。因此需要对预测结果进行一个后处理操作，用同一天其他小区预测结果为正值的流量数据计算并取均值来填补预测结果为负值的流量数据。

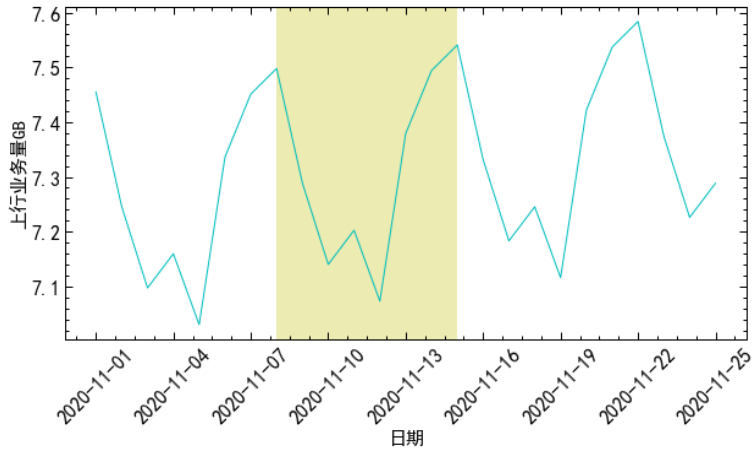

图 22 10245 小区待预测上行业务量

长期时间序列预测可能相较短期时间序列预测存在误差大，时间慢等问题，但从预测结果看：随机选取一小区，对预测的上行业务量进行可视化展示(如图 22)，可以发现基站流量随着时间会有一个波动，但是无论如何波动，从整体来看流量是呈逐渐增加趋势的，且可以预测出业务量的大致范围。而通过长期的业务量预测得到的趋势和范围，可以为运营商的物理扩容规划和设计提供引导和参考。

3.3 问题三

据了解，如今，三大运营商基站越建越多，但在运营商接入侧投资效益管控中，光接入网项目完工一年资源利用率就低于上年末运营商平均水平，而这也意味着运营商在网络建设过程中资源利用的浪费[23]。我们基于前面对于数据的分析，发现数据存在部分小区上行和下行业务量都为 0 或者小区总流量极低，这个现象可能是基站小区建在没有或者很少人用手机的地方，如偏远地区或者山区。这样就导致总体基站的利用率较低。

在本题中，给出了大量基站小区的一段时序数据，我们通过对不同基站小区数据研究，发现可以基于本题数据研究基站小区之间的空间合作关系，即得到基站小区的同簇性和异簇性[24]：距离较近的基站往往存在着紧密的联系，但由于用户群体的移动，使得距离较远的基站可能由于服务相同的用户群体而关系紧密。一方面，用户在不同基站下的访问行为，使得基站需要合作，从而产生紧密的合作关系，另一方面，用户在基站间的移动会造成基站的流量变化，存在因果关系[25]。

3.3.1 思路

基于本题的数据，包含了 132279 个不同的基站小区，探究各个小区之间的因果合作关系，可以分以下三步进行：

(1) 选择不同的目标基站小区，将目标小区的流量序列分别与其他小区的流量序列进行格兰杰因果关系检验[26]，得到因果关系最强的若干小区。

(2) 对于若干因果小区进行 k-mean 聚类，得到细粒度因果合作小区，作为同簇小区。对因果和合作处理单一或很少的小区作为异簇小区。

(3) 对于格兰杰因果关系检验不合格的小区进行相关性检验，对于相关性不强的小区作为异簇小区，其他的小区作为弱关系小区。

通过得到同簇和异簇基站小区，从而可以避免单个异簇或者很少的异簇组成。从而提高总体基站小区的利用率。

四、不足和展望

4.1 不足

1. 在第一问中使用 LSTM 和 ForecastNet 深度神经网络时，我们只是基于单个基站小区的历史数据进行预测未来，但是因本题的数据过大，全部数据将近 1.4 亿，而我们团队没有服务机器，所以没有考虑其他基站的空间合作关系作为网络的 Embedding 层，如果考虑之后结果可能会更好。
2. 在第二问的解决过程中，因为是长期预测，深度神经网络会很乏力，我们采用了 Prophet 单模型预测，可能会导致过度拟合而造成结果误差偏大。

4.2 展望

2022 年将是 5G 规模建设迈入的第二年，我们认为市场对 5G 亦或是 4G 网络价值的认知更加清晰，新基建势头持续上升，所以建立高质量的公网网络是必须要面对的难题。而本文基于题目的要求和启发，成功的预测和分析未来短期和长期的业务量变化，以便开源节流，充分利用资源，助力实现数字化基站建设。

五、参考文献

[1] scxyz_. 一种异常值检测方法、原理、代码实现(基于箱线图). https://blog.csdn.net/sscc_learning/article/details/78771324. 2020.01.07.

[2] Jason Brownlee. How to Convert a Time Series to a Supervised Learning Problem in Python. https://machinelearningmastery.com/convert-time-series-supervised-learning-problem-python. 2020/12/22.

[3] Alberto Gasparin, Slobodan Lukovic, Cesare Alippi. Deep Learning for Time Series Forecasting: The Electric Load Case. arXiv:1907.09207 [cs.LG] 2019.

[4] Xu, Yun, Goodacre Royston(2018). "On Splitting Training and Validation Set: A Comparative Study of Cross-Validation, Bootstrap and Systematic Sampling for Estimating the Generalization Performance of Supervised Learning". Journal of Analysis and Testing. Springer Science and Business Media LLC. 2(3): 249–262. doi:10.1007/s41664-018-0068-2. ISSN 2096-241X.

[5] Pengfei Ni. 数据集拆分.https://feisky.xyz/machine-learning/basic/datasets.html.2020/12/22.

[6] Murat, M., Malinowska, I., Gos, M., Krzyazczak, J.: Forecasting daily meteorological time series using ARIMA and regression models. Int. Agrophysics 32(2), 253–264(2017).

[7] Sean J. Taylor, Benjamin Letham(2018) Forecasting at scale. The American Statistician 72(1):37-45

[8] 张戎. Facebook 时间序列预测算法 Prophet 的研究. https://zhuanlan.zhihu.com/p/52330017. 2020/12/25.

[9] Croston, J. D. (1972). Forecasting and stock control for intermittent demands. Operational Research Quarterly, 23(3), 289–303.

[10] Jerome H Friedman. Greedy function approximation: a gradient boosting machine.Annals of statistics, pages 1189–1232, 2001.

[11] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, Tie-Yan Liu. "LightGBM: A Highly Efficient Gradient Boosting Decision Tree". Advances in Neural Information Processing Systems 30(NIPS 2017), pp. 3149-3157.

[12] Agnis Liukis. Approaching Time-Series with a Tree-based Model. https://towardsdatascience.com/approaching-time-series-with-a-tree-based-model-87c6d1fb6603. 2020/12/26.

[13] 落叶归根的猪. Sklearn—交叉验证(Cross-Validation). https://zhuanlan.zhihu.com/p/26791062 9. 2020/12/27.

[14] H. S. Hippert, C. E. Pedreira, and R. C. Souza. Neural networks for short-term load

forecasting: a review and evaluation. IEEE Transactions on Power Systems, 16(1):44–55, Feb 2001.

[15] Jiann-Fuh Chen, Wei-Ming Wang, and Chao-Ming Huang. Analysis of an adaptive time-series autoregressive moving-average (arma) model for short-term load forecasting. Electric Power Systems Research, 34(3):187–196,1995.

[16] Martin T Hagan and Suzanne M Behr. The time series approach to short term load forecasting. IEEE Transactions on Power Systems, 2(3):785–791, 1987.

[17] Chao-Ming Huang, Chi-Jen Huang, and Ming-Li Wang. A particle swarm optimization to identifying the armax model for short-term load forecasting. IEEE Transactions on Power Systems, 20(2):1126–1133, 2005.

[18] K.Y.Lee, Y. T. Cha, and J. H. Park. Short-term load forecasting using an artificial neural network. IEEE Transactions on Power Systems, 7(1):124–132, Feb 1992.

[19] Dabrowski J.J., Zhang Y., Rahman A. (2020) ForecastNet: A Time-Variant Deep Feed-Forward Neural Network Architecture for Multi-step-Ahead Time-Series Forecasting. In: Yang H., Pasupa K., Leung A.CS., Kwok J.T., Chan J.H., King I. (eds) Neural Information Processing. ICONIP 2020. Lecture Notes in Computer Science, vol 12534. Springer, Cham.

[20] D. C. Park, M. A. El-Sharkawi, R. J. Marks, L. E. Atlas, and M. J. Damborg. Electric load forecasting using an artificial neural network. IEEE Transactions on Power Systems, 6(2):442–449, May 1991.

[21] Zhang S, Zhang J, Zhu X, et al. Missing value imputation based on data clustering[M]//Transactions on computational science I. Springer, Berlin, Heidelberg, 2008: 128-138.

[22] 王晓, 揣锦华, 张立恒. 基于 Prophet 算法的铁路客流量预测研究[J]. 计算机技术与发展, 2020, v.30;No.278(06):136-140+156.

[23] 岑祺. 5G 基站市电建设及改造方案[J]. 信息通信. 2019 (12).

[24] 孙莹, 陈夏明, 王海洋等. 城市尺度下基站人群时空预测模型[J]. 计算机应用研究, 2016, 33(12): 3521-3526.

[25] 彭铎, 周建国, 羿舒文, 江昊. 基于空间合作关系的基站流量预测模型[J]. 计算机应用, 2019, 39(1): 154-159.

[26] PAUL U, SUBRAMANIAN A P, BUDDHIKOT M M, et al. Understanding traffic dynamics in cellular data networks[C]//INFOCOM 2011: Proceedings of the 201130th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies. Piscataway, NJ: IEEE, 2011: 882-890.

附录

程序 1：数据提取	运行环境：Anaconda Python 3

```
import pandas as pd
from tqdm import tqdm
data= pd.read_csv(r"\训练用数据.csv", encoding = 'gbk')
submit1 = pd.read_csv(r"\附件 2：短期验证选择的小区数据集.csv", encoding = '
gbk')
submit2 = pd.read_csv(r"\附件 3：长期验证选择的小区数据集.csv", encoding = '
gbk')
for i in tqdm(submit1['小区编号'].unique()):
    df = data[data['小区编号'] == i].reset_index(drop=True)
    df.to_csv(f'..1/新建文件夹 1/df_{i}.csv')
for i in tqdm(submit2['小区编号'].unique()):
    df = data[data['小区编号'] == i].reset_index(drop=True)
    df.to_csv(f'..2/新建文件夹 2/df_{i}.csv')
```

程序 2：数据预处理	运行环境：Python 3

```
def data_splits(df,n_time):
    train = df.iloc[:-n_time]
    valid = df.iloc[-n_time:]
    return train,valid
def metrics_(y_true, y_pred, metric):
    if metric=='MSE':
        MSE = metrics.mean_squared_error(y_true, y_pred)
        return MSE
    elif metric=='RMSE':
        RMSE = np.sqrt(metrics.mean_squared_error(y_true, y_pred))
        return RMSE
    elif metric=='MAE':
        MAE = metrics.mean_absolute_error(y_true, y_pred)
        return MAE
    else:
        r2 = metrics.r2_score(y_true, y_pred)
        return r2

def data_process(df):
```

```
    df.loc[df['日期']=='018-04-04',"日期"] = '2018/4/4'
    df.loc[df['日期']=='018-04-01',"日期"] = '2018/4/1'
    df.loc[df['日期']=='018-04-09',"日期"] = '2018/4/9'
    df.loc[df['日期']=='018-04-02',"日期"] = '2018/4/2'
    df.loc[df['日期']=='018-04-10',"日期"] = '2018/4/10'
    df.loc[df['日期']=='018-04-08',"日期"] = '2018/4/8'
    df.loc[df['日期']=='018-04-03',"日期"] = '2018/4/3'
    df['date'] = (df['日期']+' '+df['时间']).apply(lambda x:pd.to_datetim
e(x))
    df.sort_values("date",inplace=True)
    df = df.drop_duplicates().reset_index(drop=True)
    return df
def processing(df,code,name,cut,is_test):
    if is_test:
        df,valid = data_splits(df,7*24)
        #对数据进行去重操作
        df = df.drop_duplicates().reset_index(drop=True)
        df = df.fillna(method='ffill')
        #统计数据的日期数量存进字典
        t = dict(df['日期'].value_counts())
        data = pd.DataFrame({'日期': list(t.keys()),'num': list(t.values
())})
        #删除一天中小于12 条数据的天
        data = data.loc[data['num']!=24]
        data0 = data.loc[data['num']<12]
        columns = data0['日期'].values
        df = df[~df.日期.isin(columns)]
        #提取出数据中一天中12-24 条数据的天
        data1 = data.loc[data['num']>=12]
        columns = data1['日期'].values
        for i in range(len(columns)):
            x = pd.date_range(start=columns[i],periods=24,freq='h')
            x = pd.DataFrame(x,columns=['date'])
            df = pd.merge(df,x,how='outer')
            df['日期'] = df["date"].apply(lambda x : x.strftime('%Y-%m-%d
'))
            df['时间'] = df["date"].apply(lambda x : x.strftime('%H:%M:%S
```

```
'))
            df['小区编号'] = code
            df = df.sort_values(by=['时间','日期'])  # 先按时间这一列进行排
序
            df = df.reset_index(drop=True)
            #向前填充
            df = df.fillna(method='ffill')
            #再给他变回来
            df = df.sort_values(by=['date'])  # 先按时间这一列进行排序
            df = df.reset_index(drop=True)
        #对其异常值做一个处理
        data_x = df[name].dropna()
        data_cut = np.percentile(data_x,cut)
        df[name][df[name] >=  data_cut] =  data_cut
        df = pd.concat([df,valid],axis=0).reset_index(drop=True)
        return df
    else:
        #对数据进行去重操作
        df = df.drop_duplicates().reset_index(drop=True)
        df = df.fillna(method='ffill')
        #统计数据的日期数量存进字典
        t = dict(df['日期'].value_counts())
        data = pd.DataFrame({'日期': list(t.keys()),'num': list(t.values
())})
        #删除一天中小于12 条数据的天
        data = data.loc[data['num']!=24]
        data0 = data.loc[data['num']<12]
        columns = data0['日期'].values
        df = df[~df.日期.isin(columns)]
        #提取出数据中一天中12-24 条数据的天
        data1 = data.loc[data['num']>=12]
        columns = data1['日期'].values
        for i in range(len(columns)):
            x = pd.date_range(start=columns[i],periods=24,freq='h')
            x = pd.DataFrame(x,columns=['date'])
            df = pd.merge(df,x,how='outer')
            df['日期'] = df["date"].apply(lambda x : x.strftime('%Y-%m-%d
```

```
'))
            df['时间'] = df["date"].apply(lambda x : x.strftime('%H:%M:%S
'))
            df['小区编号'] = code
            df = df.sort_values(by=['时间','日期'])  # 先按时间这一列进行排
序
            df = df.reset_index(drop=True)
            #向前填充
            df = df.fillna(method='ffill')
            #再给他变回来
            df = df.sort_values(by=['date'])  # 先按时间这一列进行排序
            df = df.reset_index(drop=True)
        #对其异常值做一个处理
        data_x = df[name].dropna()
        data_cut = np.percentile(data_x,cut)
        df[name][df[name] >=  data_cut] =  data_cut
        return df
```

问题一代码	
程序 3：ARMA 模型	运行环境： Anaconda Python 3

```
def Arma_run(df=None ,fore_range=None, target=None, is_test=None):
    df = df.dropna().reset_index(drop=True)
    if is_test:
        #划分数据
        train,valid = data_splits(df,7*24)
        armamodel = ARMA(train[target], order=(1,0))
        fit_result = armamodel.fit(disp = -1, method="css")
        predictions=fit_result.predict(start=len(train), end=len(train)+
len(valid)-1, dynamic=False)
        RMSE1 = metrics_(valid[target], predictions, 'RMSE')
        #预测
        df_fog = df[['date',target]].copy()
        df_fog.columns = ['ds', 'y']
        model = ARMA(df_fog['y'], order=(1,0))
        model_fog = model.fit(disp = -1, method="css")
```

```
        future = pd.DataFrame(list(fore_range))
        future.columns = ['ds']
        forecast_fog = model_fog.predict(start=len(df), end=len(df)+len
(future)-1, dynamic=False)
        return RMSE1,forecast_fog
    else:
        df_fog = df[['date',target]].copy()
        df_fog.columns = ['ds', 'y']
        model = ARMA(df_fog['y'], order=(1,0))
        model_fog = model.fit(disp = -1, method="css")
        future = pd.DataFrame(list(fore_range))
        future.columns = ['ds']
        forecast_fog = model_fog.predict(start=len(df), end=len(df)+len
(future)-1, dynamic=False)
        return forecast_fog
def arma(path = None,str_ = None,code = None,fore_range=None):
    IS_test = False
    df = pd.read_csv(path+'/'+f'df_{code}.csv',index_col=0)
    df = data_process(df)
    fore_range = sub.loc[sub['小区编号']==code,"date"]
    df = processing(df,code,str_,95,IS_test)
    predictions = Arma_run(df ,fore_range, target=str_, is_test=IS_test)
    return predictions
```

程序 4：Prophet 模型	运行环境： Anaconda Python 3

```
def Prophet_run(df=None ,fore_range=None, target=None, is_test=None):
    df = df.dropna().reset_index(drop=True)
    if is_test:
        #划分数据
        train,valid = data_splits(df,7*24)
        df_ = train[['date',target]].copy()
        df_.columns = ['ds', 'y']
        model = Prophet()
        model.fit(df_)
        #预测验证数据
        future_val = pd.DataFrame(list(valid['date']))
        future_val.columns = ['ds']
        forecast = model.predict(future_val)
```

```
        RMSE = metrics_(valid[target], forecast['yhat'], 'RMSE')
#         #预测
#         df_fog = df[['date',target]].copy()
#         df_fog.columns = ['ds', 'y']
#         model_fog = Prophet()
#         model_fog.fit(df_fog)
#         future = pd.DataFrame(list(fore_range))
#         future.columns = ['ds']
#         forecast_fog = model_fog.predict(future)
        return RMSE
    else:
        df_ = df[['date',target]].copy()
        df_.columns = ['ds', 'y']
        model = Prophet()
        model.fit(df_)
        #预测
        df_fog = df[['date',target]].copy()
        df_fog.columns = ['ds', 'y']
        model_fog = Prophet()
        model_fog.fit(df_fog)
        future = pd.DataFrame(list(fore_range))
        future.columns = ['ds']
        forecast_fog = model_fog.predict(future)
        return forecast_fog['yhat']
def prophet(path = None,str_ = None,code = None,fore_range=None):
    IS_test = False
    df = pd.read_csv(path+'/'+f'df_{code}.csv',index_col=0)
    df = data_process(df)
    fore_range = sub.loc[sub['小区编号']==code,"date"]
    df = processing(df,code,str_,95,is_test=IS_test)
    predictions = Prophet_run(df,fore_range,target=str_, is_test=IS_tes
t)
    return predictions
```

程序 5：LightGBM 模型	运行环境： Anaconda Python 3

```
def time_process(data_df):
    date_ = data_df['date'].copy()
    data_df["hour"] = date_.dt.hour
```

```
    data_df["day"] = date_.dt.day
    data_df['dayofweek'] = date_.dt.dayofweek
    data_df['weekofyear'] = date_.dt.week
    return data_df
# 构造时间特征
def get_time_fe(data, col, n, one_hot=False, drop=True):
    data[col + '_sin'] = round(np.sin(2*np.pi / n * data[col]), 6)
    data[col + '_cos'] = round(np.cos(2*np.pi / n * data[col]), 6)
    if one_hot:
        ohe = OneHotEncoder()
        X = OneHotEncoder().fit_transform(data[col].values.reshape(-1,
1)).toarray()
        df = pd.DataFrame(X, columns=[col + '_' + str(int(i)) for i in ra
nge(X.shape[1])])
        data = pd.concat([data, df], axis=1)
    if drop:
        data = data.drop(col, axis=1)
    return data
# 构造过去 n 天的统计数据
def get_statis_n_days_num(data, col, col1,n):
    data['avg_'+ str(n) +'_days_' + col1] = data[col].rolling(window=n).
mean()
    data['median_'+ str(n) +'_days_' + col1] = data[col].rolling(window=
n).median()
    data['max_'+ str(n) +'_days_' + col1] = data[col].rolling(window=n).
max()
    data['min_'+ str(n) +'_days_' + col1] = data[col].rolling(window=n).
min()
    data['std_'+ str(n) +'_days_' + col1] = data[col].rolling(window=n).
std()
    data['skew_'+ str(n) +'_days_' + col1] = data[col].rolling(window=
n).skew()
    data['kurt_'+ str(n) +'_days_' + col1] = data[col].rolling(window=
n).kurt()
    data['q1_'+ str(n) +'_days_' + col1] = data[col].rolling(window=n).q
uantile(0.25)
    data['q3_'+ str(n) +'_days_' + col1] = data[col].rolling(window=n).q
```

```
uantile(0.75)
    return data
def shfit_fe(tmp,tar,col1):
    for i in range(1,24):
        tmp[f'{col1}_shifts_{i}'] = tmp[tar].shift(i)
    return tmp
def get_target(tmp,tar,col1):
    for i in range(1,169):
        tmp[f'{col1}_{i}'] = tmp[tar].shift(-i)
    return tmp
def run_lgb(df_train, df_test,target,cols):
    model = lgb.LGBMRegressor(
                          objective = 'regression', #mse,rmse
                          boosting_type='gbdt',
                          num_leaves=32,
                          max_depth=8,
                          learning_rate=0.05,
                          n_estimators=200,
                          subsample=0.8,
                           min_data_in_leaf=350,
#                            max_bin=128,
                          feature_fraction=0.8,
                          random_state=2020,
                          n_jobs=-1,
                          metric='mse')
    preds = np.zeros(df_test.shape[0])
    kfold = KFold(n_splits=5,shuffle=True, random_state=2020)
    for fold_id, (trn_idx, val_idx) in enumerate(kfold.split(df_train,ta
rget)):
        X_train = df_train.iloc[trn_idx]
        Y_train = target.iloc[trn_idx][cols]
        X_val = df_train.iloc[val_idx]
        Y_val = target.iloc[val_idx][cols]
        lgb_model = model.fit(X_train,
                              Y_train,
                              eval_names=['train', 'valid'],
                              eval_set=[(X_train, Y_train), (X_val, Y_va
```

```
l)],
                              verbose=0,
                              eval_metric='mse',
                              early_stopping_rounds=50)
        preds += lgb_model.predict(df_test, num_iteration=lgb_model.best
_iteration_)/5
    return preds
def lgb_run(df=None, str_=None, title_=None, is_test=None):
    df = get_time_fe(df, 'day', n=31, one_hot=False, drop=True)
    df = get_time_fe(df, 'dayofweek', n=7, one_hot=True, drop=True)
    df = get_time_fe(df, 'hour', n=12, one_hot=True, drop=True)
    df.sort_values("date",inplace=True)
    df = get_statis_n_days_num(df, str_, title_, n=24*1)
    df = get_statis_n_days_num(df, str_, title_, n=24*2)
    df = get_statis_n_days_num(df, str_, title_, n=24*3)
    df = get_statis_n_days_num(df, str_, title_, n=24*4)
    df = get_statis_n_days_num(df, str_, title_, n=24*5)
    df = get_statis_n_days_num(df, str_, title_, n=24*6)
    # df = shfit_fe(df,'上行业务量GB','shangxing')
    df = df.dropna().reset_index(drop=True)
    if is_test:
        #划分数据
        train_data,test_data = data_splits(df,7*24)
        train_data,test_data1 = train_data.iloc[:-1],train_data.iloc[-1]
        train_data = get_target(train_data, str_, title_)
        train_data = train_data.dropna().reset_index(drop=True)
        col = ['日期', '时间', '小区编号', '上行业务量GB', '下行业务量GB','d
ate']
        target_cols = [f'{title_}_'+str(i) for i in range(1,169)]
        feats_cols = [i for i in df.columns if i not in target_cols+col]
        test_df = pd.DataFrame()
#         test_x = test_data.iloc[0]
        test_x = pd.DataFrame().append(test_data1, ignore_index=True)

        for i in target_cols:
            tests = run_lgb(train_data[feats_cols], test_x[feats_cols], t
rain_data[target_cols], i)
```

```
            test_df[i] = tests
        RMSE = metrics_(test_data[str_], test_df.values.flatten(), 'RMSE
')
        return RMSE,test_df
    else:
        train_data,test_data = df.iloc[:-1],df.iloc[-1]
        train_data = get_target(train_data, str_, title_)
        train_data = train_data.dropna().reset_index(drop=True)
        col = ['日期', '时间', '小区编号', '上行业务量GB', '下行业务量GB','d
ate']
        target_cols = [f'{title_}_'+str(i) for i in range(1,169)]
        feats_cols = [i for i in df.columns if i not in target_cols+col]
        test_df = pd.DataFrame()
        test_x = pd.DataFrame().append(test_data, ignore_index=True)
        for i in target_cols:
            tests = run_lgb(train_data[feats_cols], test_x[feats_cols], t
rain_data[target_cols], i)
            test_df[i] = tests
        return test_df
def lightgbm(path = None,str_ = None,code = None,title_ = None):
    IS_test = False
    df = pd.read_csv(path+'/'+f'df_{code}.csv',index_col=0)
    df = data_process(df)
    df = processing(df,code,str_,95,IS_test)
    df = time_process(df)
    test_df = lgb_run(df=df, str_=str_, title_=title_, is_test=IS_test)
    predictions = test_df.values.flatten()
    return predictions
```

程序 6：LSTM 模型	运行环境：Anaconda Python 3、Pytorch 1.0.1

```
def get_lstm_data(df=None, slidingWindowLength=None, str_=None, is_test=
None, device=None):
    """
    该函数用于得到lstm训练和测试数据
    :param df: 输入数据
    :param slidingWindowLength: 训练数据时间步
    :param str_: 输入需要预测单变量字符串
    :param is_test: 是否测试误差
```

```
    """
    if is_test:
        data = df[str_].values
        train_data, test_data = data[:-7*24],data[-7*24:]  #划分最后一周数
据测试
        train_data = train_data.reshape(-1,1)
        #标准化
        sc = MinMaxScaler(feature_range=(0,1))
        train = sc.fit_transform(train_data).reshape(-1)
        train = torch.tensor(train, dtype=torch.float32, device=device).v
iew(-1)
        #构建训练数据和标签
        inout_seq = []
        for i in range(len(train)-slidingWindowLength):
            x = train[i : i+slidingWindowLength]
            y = train[i+slidingWindowLength : i+slidingWindowLength+1]
            inout_seq.append((x ,y))
        return inout_seq, train, test_data, sc
    else:
        data = df[str_].values
        data = data.reshape(-1,1)
        #标准化
        sc = MinMaxScaler(feature_range=(0,1))
        train = sc.fit_transform(data).reshape(-1)
        train = torch.tensor(train, dtype=torch.float32, device=device).v
iew(-1)
        #构建训练数据和标签
        inout_seq = []
        for i in range(len(train)-slidingWindowLength):
            x = train[i : i+slidingWindowLength]
            y = train[i+slidingWindowLength : i+slidingWindowLength+1]
            inout_seq.append((x ,y))
        return inout_seq, train, sc

def train_lstm_model(model=None, optimizer=None, loss_function=None, tra
in_inout_seq=None, epochs=None, device=None):
    for i in range(epochs):
```

```
        for seq, labels in train_inout_seq:
            optimizer.zero_grad()
            model.hidden_cell = (torch.zeros(1, 1, model.hidden_layer_siz
e, device=device),
                                 torch.zeros(1, 1, model.hidden_layer_size, dev
ice=device))
            y_pred = model(seq)
            single_loss = loss_function(y_pred, labels)
            single_loss.backward()
            optimizer.step()
#         if i%5 == 0:
#             print(f'epoch: {i:3} loss: {single_loss.item():10.8f}')

#     print(f'epoch: {i:3} loss: {single_loss.item():10.10f}')
    return model

def evaluate_lstm_test_set(model=None, train_data_normalized=None, train
_window=None, pred_size=None, device=None, sc=None, is_test=None):
    if is_test:
        test_inputs = train_data_normalized[-train_window:].tolist()
        model.eval()
        for i in range(pred_size):
            seq = torch.tensor(test_inputs[-train_window:], dtype=torch.f
loat32, device=device)
            with torch.no_grad():
                model.hidden = (torch.zeros(1, 1, model.hidden_layer_siz
e, device=device),
                                torch.zeros(1, 1, model.hidden_layer_size,
device=device))
                test_inputs.append(model(seq).item())
        actual_predictions = sc.inverse_transform(np.array(test_inputs[t
rain_window:]).reshape(-1, 1))
        RMSE1 = metrics_(test_data,actual_predictions, 'RMSE')
        return RMSE1
    else:
        test_inputs = train_data_normalized[-train_window:].tolist()
        model.eval()
```

```
        for i in range(pred_size):
            seq = torch.tensor(test_inputs[-train_window:], dtype=torch.f
loat32, device=device)
            with torch.no_grad():
                model.hidden = (torch.zeros(1, 1, model.hidden_layer_siz
e, device=device),
                                torch.zeros(1, 1, model.hidden_layer_size,
device=device))
                test_inputs.append(model(seq).item())
        actual_predictions = sc.inverse_transform(np.array(test_inputs[t
rain_window:]).reshape(-1, 1))
        return actual_predictions

class LSTM(nn.Module):
    def __init__(self, input_size=1, hidden_layer_size=64, output_size=
1, num_layers=None):
        super().__init__()
        self.num_layers = num_layers
        self.hidden_layer_size = hidden_layer_size
        self.lstm = nn.LSTM(input_size, hidden_layer_size, num_layers=sel
f.num_layers)
        self.hidden_cell = (torch.zeros(self.num_layers,1,self.hidden_la
yer_size),torch.zeros(self.num_layers,1,self.hidden_layer_size))
        self.reg = nn.Sequential(
            nn.Linear(hidden_layer_size, 32),
#             nn.Dropout(p=0.5),
            nn.Tanh(),
            nn.Linear(32, output_size),
        )
    def forward(self, input_seq):
        lstm_out, self.hidden_cell = self.lstm(input_seq.view(len(input_
seq) ,1, -1), self.hidden_cell)
        predictions = self.reg(lstm_out.view(len(input_seq), -1))
        return predictions[-1]

def lstm(path = None,str_ = None,code = None):
    pred_size = 168
```

```
    train_window = 24
    num_layers = 1
    IS_test = False
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu
")
    setup_seed(2020)
    df = pd.read_csv(path+'/'+f'df_{code}.csv',index_col=0)
    model = LSTM(num_layers=num_layers).to(device)
    loss_function = nn.MSELoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
    df = data_process(df = df)
    df = processing(df,code,str_,95,IS_test)
    inout_seq, train, sc = get_lstm_data(df = df, slidingWindowLength = t
rain_window, str_ = str_, is_test = IS_test, device=device)
    model = train_lstm_model(model=model, optimizer=optimizer, loss_func
tion=loss_function, train_inout_seq=inout_seq, epochs=20, device=device)
    predictions = evaluate_lstm_test_set(model=model, train_data_normali
zed = train, train_window=train_window, pred_size=pred_size, device=devi
ce, is_test = IS_test, sc=sc)
    return predictions
```

程序 7：ForecastNet 模型	运行环境：Python 3、tensorflow 2.0.0

```
class forecastNet():
    """
    Class for the densely connected hidden cells version of the model
    """
    def __init__(self, seed_value=None, path=None, str_=None, IS_test=No
ne, WindowLength=None,
                 hidden_dim=None, epochs=None, batch_size=None,code=Non
e):
        """
        Constructor
        :param seed_value: tf 的种子值
        :param path: 文件夹路径
        :param str_: 预测的目标字符串
        :param IS_test: 是否验证(True 表示只验证，False 表示只预测)
        :param WindowLength: 训练的时间步长
        :param hidden_dim: dense 的神经元个数
```

```
        :param epochs: 训练的迭代次数
        :param batch_size: 训练的批次大小
        """
        self.is_test = IS_test
        self.epochs = epochs
        self.batch_size = batch_size
        self.code = codes

        random.seed(seed_value)
        np.random.seed(seed_value)
        tf.random.set_seed(seed_value)
        df = pd.read_csv(path+'/'+f'df_{code}.csv',index_col=0)
        df = self.data_process(df = df)
        df = self.processing(df=df,code=code,name=str_,cut=95,is_test=IS
_test)
        trainX, trainY, data, sc = self.get_data(df = df, slidingWindowLe
ngth = WindowLength, str_ = str_, is_test = IS_test)
        self.predicted = self.forecastNetDenseDenseGraph(Xtrain = trainX,
 Ytrain = trainY, train_data = data, sc = sc, hiddenDim = hidden_dim, is_
test = IS_test)
    def data_process(self, df=None):
        df.loc[df['日期']=='018-04-04',"日期"] = '2018/4/4'
        df.loc[df['日期']=='018-04-01',"日期"] = '2018/4/1'
        df.loc[df['日期']=='018-04-09',"日期"] = '2018/4/9'
        df.loc[df['日期']=='018-04-02',"日期"] = '2018/4/2'
        df.loc[df['日期']=='018-04-10',"日期"] = '2018/4/10'
        df.loc[df['日期']=='018-04-08',"日期"] = '2018/4/8'
        df.loc[df['日期']=='018-04-03',"日期"] = '2018/4/3'
        df['date'] = (df['日期']+' '+df['时间']).apply(lambda x:pd.to_dat
etime(x))
        df.sort_values("date",inplace=True)
        return df

    def processing(self, df=None,code=None,name=None,cut=None,is_test=No
ne):
        if is_test:
            df,valid = self.data_splits(df=df,n_time=7*24)
```

```
            #对数据进行去重操作
            df = df.drop_duplicates().reset_index(drop=True)
            df = df.fillna(method='ffill')
            #统计数据的日期数量存进字典
            t = dict(df['日期'].value_counts())
            data = pd.DataFrame({'日期': list(t.keys()),'num': list(t.val
ues())})
            #删除一天中小于12 条数据的天
            data = data.loc[data['num']!=24]
            data0 = data.loc[data['num']<12]
            columns = data0['日期'].values
            df = df[~df.日期.isin(columns)]
            #提取出数据中一天中12-24 条数据的天
            data1 = data.loc[data['num']>=12]
            columns = data1['日期'].values
            for i in range(len(columns)):
                x = pd.date_range(start=columns[i],periods=24,freq='h')
                x = pd.DataFrame(x,columns=['date'])
                df = pd.merge(df,x,how='outer')
                df['日期'] = df["date"].apply(lambda x : x.strftime('%Y-%m
-%d'))
                df['时间'] = df["date"].apply(lambda x : x.strftime('%H:%
M:%S'))
                df['小区编号'] = code
                df = df.sort_values(by=['时间','日期'])  # 先按时间这一列进
行排序
                df = df.reset_index(drop=True)
                #向前填充
                df = df.fillna(method='ffill')
                #再给他变回来
                df = df.sort_values(by=['date'])  # 先按时间这一列进行排序
                df = df.reset_index(drop=True)
            #对其异常值做一个处理
            data_x = df[name].dropna()
            data_cut = np.percentile(data_x,cut)
            df[name][df[name] >=  data_cut] =  data_cut
            df = pd.concat([df,valid],axis=0).reset_index(drop=True)
```

```
            return df
        else:
            #对数据进行去重操作
            df = df.drop_duplicates().reset_index(drop=True)
            df = df.fillna(method='ffill')
            #统计数据的日期数量存进字典
            t = dict(df['日期'].value_counts())
            data = pd.DataFrame({'日期': list(t.keys()),'num': list(t.val
ues())})
            #删除一天中小于12 条数据的天
            data = data.loc[data['num']!=24]
            data0 = data.loc[data['num']<12]
            columns = data0['日期'].values
            df = df[~df.日期.isin(columns)]
            #提取出数据中一天中12-24 条数据的天
            data1 = data.loc[data['num']>=12]
            columns = data1['日期'].values
            for i in range(len(columns)):
                x = pd.date_range(start=columns[i],periods=24,freq='h')
                x = pd.DataFrame(x,columns=['date'])
                df = pd.merge(df,x,how='outer')
                df['日期'] = df["date"].apply(lambda x : x.strftime('%Y-%m
-%d'))
                df['时间'] = df["date"].apply(lambda x : x.strftime('%H:%
M:%S'))
                df['小区编号'] = code
                df = df.sort_values(by=['时间','日期'])  # 先按时间这一列进
行排序
                df = df.reset_index(drop=True)
                #向前填充
                df = df.fillna(method='ffill')
                #再给他变回来
                df = df.sort_values(by=['date'])  # 先按时间这一列进行排序
                df = df.reset_index(drop=True)
            #对其异常值做一个处理
            data_x = df[name].dropna()
            data_cut = np.percentile(data_x,cut)
```

```
            df[name][df[name] >=  data_cut] =  data_cut
            return df

    def data_splits(self, df=None,n_time=None):
        train = df.iloc[:-n_time]
        valid = df.iloc[-n_time:]
        return train,valid

    def metrics_(self, y_true=None, y_pred=None, metric=None):
        if metric=='MSE':
            MSE = metrics.mean_squared_error(y_true, y_pred)
            return MSE
        elif metric=='RMSE':
            RMSE = np.sqrt(metrics.mean_squared_error(y_true, y_pred))
            return RMSE
        elif metric=='MAE':
            MAE = metrics.mean_absolute_error(y_true, y_pred)
            return MAE
        else:
            r2 = metrics.r2_score(y_true, y_pred)
            return r2

    def get_data(self, df=None, slidingWindowLength=None, str_=None, is_
test=None):
        """
        该函数用于得到forecastNet 训练和测试数据
        :param df: 输入数据
        :param slidingWindowLength: 训练数据时间步
        :param str_: 输入需要预测单变量字符串
        :param is_test: 是否测试误差
        """
        if is_test:
            data = df[str_].values
            train_data, test_data = data[:-7*24],data[-7*24:]  #划分最后一
周数据测试
            train_data = train_data.reshape(-1,1)
            #标准化
```

```
            sc = MinMaxScaler(feature_range=(0,1))
            train = sc.fit_transform(train_data).reshape(-1)
            #构建训练数据和标签
            trainX, trainY = list(), list()
            for i in range(len(train)-slidingWindowLength):
                x = train[i : i+slidingWindowLength]
                y = train[i+slidingWindowLength : i+slidingWindowLength+
1]
                trainX.append(x.tolist())
                trainY.append(y.tolist())
            trainX, trainY = np.asarray(trainX), np.asarray(trainY)
            return trainX, trainY, train_data, test_data, sc
        else:
            data = df[str_].values
            data = data.reshape(-1,1)
            #标准化
            sc = MinMaxScaler(feature_range=(0,1))
            train = sc.fit_transform(data).reshape(-1)
            #构建训练数据和标签
            trainX, trainY = list(), list()
            for i in range(len(train)-slidingWindowLength):
                x = train[i : i+slidingWindowLength]
                y = train[i+slidingWindowLength : i+slidingWindowLength+
1]
                trainX.append(x.tolist())
                trainY.append(y.tolist())
            trainX, trainY = np.asarray(trainX), np.asarray(trainY)
            return trainX, trainY, data, sc

    def forecastNetDenseDenseGraph(self, Xtrain=None, Ytrain=None, train
_data=None, test_data=None, sc=None, hiddenDim=None, is_test=None):
        '''
        该函数用于单变量全连接时变网络模型的构建
        :param X: 多步长单变量特征输入
        :param Y: 目标变量值
        :param hiddenDim: 每一个隐含层节点数
        :param outSeqLength: 输出多步长目标变量的长度
```

```
        :return: 时变深度前馈预测模型
        '''
        # 获取输入参数
        _, featureDims = Xtrain.shape
        # 创建输入
        inputs = tf.keras.Input(shape=(featureDims, ))
        # 第一个隐藏层
        hidden = tf.keras.layers.Dense(units=hiddenDim,
                                       activation="relu")(inputs)
        # 第二个隐藏层
        hidden = tf.keras.layers.Dense(units=hiddenDim,
                                       activation="relu")(hidden)
        # 第一交错输出
        outputs = tf.keras.layers.Dense(units=1, activation=None)(hidde
n)
        # 合并以上输出
        concat = tf.keras.layers.Concatenate(axis=-1)([inputs, hidden, ou
tputs[:, 0:1]])
        # 注意力机制
        concatWeights = tf.keras.layers.Dense(units=concat.get_shape().a
s_list()[-1], activation="sigmoid", name="attention1")(concat)
        concatWighted = tf.keras.layers.Multiply()([concat, concatWeight
s])
        # 接下来第一个隐藏层
        hidden = tf.keras.layers.Dense(units=hiddenDim, activation="relu
")(concatWighted)
        # 接下来第二个隐藏层
        hidden = tf.keras.layers.Dense(units=hiddenDim, activation="relu
")(hidden)
        # 最后一层为全连接
        output = tf.keras.layers.Dense(units=1, activation=None)(hidden)
        # 合并输出
        outputs = tf.keras.layers.Concatenate(axis=-1)([outputs, outpu
t])
        model = tf.keras.Model(inputs=inputs, outputs=outputs)
        model.compile(optimizer="adam", loss="mse", metrics=["mse"])
        history = model.fit(Xtrain, Ytrain, batch_size=self.batch_size, v
```

```
erbose=0, epochs=self.epochs, shuffle=False, use_multiprocessing=False)
        if is_test:
            X_test = np.array(train_data[-24:])
            for i in range(168):
                test = X_test[i:i+24]
                X_ = np.reshape(test, (test.shape[1],test.shape[0],1))
                predicts = model.predict(X_,batch_size=32)
                X_test = np.vstack((X_test, predicts[:,0]))
            predicted = sc.inverse_transform(X_test[-168:])
            absoluteError = self.metrics_(y_true = test_data, y_pred = pr
edicted, metric = 'RMSE')
            return absoluteError
        else:
            X_test = np.array(train_data[-24:])
            for i in range(168):
                test = X_test[i:i+24]
                X_ = np.reshape(test, (test.shape[1],test.shape[0],1))
                predicts = model.predict(X_,batch_size=32)
                X_test = np.vstack((X_test, predicts[:,0]))
            predicted = sc.inverse_transform(X_test[-168:])
            return predicted
```

程序 8：模型融合	运行环境： Anaconda Python 3

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.preprocessing import MinMaxScaler,OneHotEncoder
import os
from tqdm import tqdm
from sklearn import metrics
import torch
from torch import nn
import random
from fbprophet import Prophet
from statsmodels.tsa.arima_model import ARMA
import lightgbm as lgb
from sklearn.model_selection import KFold
import tensorflow as tf
```

```
import warnings
warnings.filterwarnings('ignore')

df1= pd.read_csv(r"\model_loss\shangxing.csv")
df2= pd.read_csv(r"\model_loss\xiaxing.csv")
code = pd.read_csv(r"\model_loss\code.csv")

df_1 = np.argmin(np.array(df1),axis=1)
df_2 = np.argmin(np.array(df2),axis=1)

str_ = '上行业务量 GB'
# date_range = pd.date_range(start='20180420',end='20180427', freq='H',
closed='left')
df1 = {}
for i,j in tqdm(enumerate(df_1)):
    codes = code['code'].iloc[i]
    if codes in [9579,8954]:
        j = 0
    if j == 0:
        predictions = prophet(path = path,str_ = str_,code = codes)
        df1 = dict(df1,**{str(codes):predictions.values})
    elif j == 1:
        predictions = lstm(path = path,str_ = str_,code = codes)
        df1 = dict(df1,**{str(codes):predictions})
    elif j == 2:
        predictions = lightgbm(path = path,str_ = str_,code = codes,title
_ = 'shangxing')
        df1 = dict(df1,**{str(codes):predictions})
    elif j == 3:
        predictions = forecastNet(seed_value=20, path=path, str_=str_,
                      IS_test=False, WindowLength=24, hidden_dim=45, epoc
                       hs=32, batch_size=32,code = codes)
        df1 = dict(df1,**{str(codes):predictions.predicted})
    else:
        predictions = arma(path = path,str_ = str_,code = codes)
        df1 = dict(df1,**{str(codes):predictions.values})
```

```
strs_ = '下行业务量 GB'
df2 = {}
for i,j in tqdm(enumerate(df_2)):
    codes = code['code'].iloc[i]
    if j == 0:
        predictions = prophet(path = path,str_ = strs_,code = codes)
        df2 = dict(df2,**{str(codes):predictions.values})
    elif j == 1:
        predictions = arma(path = path,str_ = strs_,code = codes)
        df2 = dict(df2,**{str(codes):predictions.values})
    elif j == 2:
        predictions = forecastNet(seed_value=20, path=path, str_=strs_,
                                  IS_test=False, WindowLength=24, hidden_di
m=45, epochs=32, batch_size=32,code = codes)
        df2 = dict(df2,**{str(codes):predictions.predicted})
    elif j == 3:
        predictions = lightgbm(path = path,str_ = strs_,code = codes,titl
e_ = 'xiaxing')
        df2 = dict(df2,**{str(codes):predictions})
    else:
        predictions = lstm(path = path,str_ = strs_,code = codes)
        df2 = dict(df2,**{str(codes):predictions})
```

程序 9：可视化结果	运行环境： Anaconda Python 3

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
plt.rcParams['font.sans-serif'] = ['SimHei'] # 指定默认字体为黑体
plt.rcParams['axes.unicode_minus'] = False # 解决保存是负号'-'显示为方块的
问题

shangxing = pd.read_csv(r"\实验结果\shangxing.csv")
xiaxing = pd.read_csv(r"C:\实验结果\xiaxing.csv")
df = pd.read_csv(r"\实验结果\df.csv")

data_ = df[['上行业务量 GB','下行业务量 GB','date']].iloc[-168*2:-168]
fig = plt.figure(figsize=(12, 6))
graph = fig.add_subplot(1, 1, 1)
```

```
graph.set_xlim(0, 168*2)
graph.set_ylim(-0.025, 0.14)
graph.plot(data_['date'],data_['上行业务量 GB'], color="blue", linewidth=
1)
graph.vlines(167, -0.04, 0.14, colors = "black", linestyles = "-", linew
idth=0.8)
graph.plot(shangxing['ds'],shangxing['arma'], linestyle='-', color="tan
", label="ARMA", linewidth=1.)
graph.plot(shangxing['ds'],shangxing['forecastnet'], linestyle='-', colo
r="g", label="ForecastNet", linewidth=1.)
graph.plot(shangxing['ds'],shangxing['lgb'], linestyle='-', color="c", l
abel="LightGBM", linewidth=1.)
graph.plot(shangxing['ds'],shangxing['lstm'], linestyle='-', color="r",
label="LSTM", linewidth=1.)
graph.plot(shangxing['ds'],shangxing['prophet'], linestyle='-', color="k
", label="Prophet", linewidth=1.)
graph.fill_between(shangxing['ds'], shangxing['yhat_upper'], shangxing['
yhat_lower'],facecolor="blue", alpha=0.1)
graph.set_xticks(range(0, 168*2, 30))
graph.set_xticklabels([index for index in graph.get_xticks()])
graph.set_ylabel("上行业务量 GB",fontsize=12)
graph.set_xlabel("Steps(30hours)",fontsize=15)
plt.legend(fontsize=12)
plt.show()

fig = plt.figure(figsize=(12, 6))
graph = fig.add_subplot(1, 1, 1)
graph.set_xlim(0, 168*2)
graph.set_ylim(-0.2, 0.9)
graph.plot(data_['date'],data_['下行业务量 GB'], color="blue", linewidth=
1)
graph.vlines(167, -0.2, 0.9, colors = "black", linestyles = "-", linewid
th=0.8)
graph.plot(xiaxing['ds'],xiaxing['arma'], linestyle='-', color="tan", la
bel="ARMA", linewidth=1.)
graph.plot(xiaxing['ds'],xiaxing['forecastnet'], linestyle='-', color="g
", label="ForecastNet", linewidth=1.)
```

```
graph.plot(xiaxing['ds'],xiaxing['lgb'], linestyle='-', color="c", label
="LightGBM", linewidth=1.)
graph.plot(xiaxing['ds'],xiaxing['lstm'], linestyle='-', color="r", labe
l="LSTM", linewidth=1.)
graph.plot(xiaxing['ds'],xiaxing['prophet'], linestyle='-', color="k", l
abel="Prophet", linewidth=1.)
graph.fill_between(xiaxing['ds'], xiaxing['yhat_upper'], xiaxing['yhat_l
ower'],facecolor="blue", alpha=0.1)
graph.set_xticks(range(0, 168*2, 30))
graph.set_xticklabels([index for index in graph.get_xticks()])
graph.set_ylabel("下行业务量 GB",fontsize=12)
graph.set_xlabel("Steps(30hours)",fontsize=15)
plt.legend(fontsize=12)
plt.show()

fig = plt.figure(figsize=(12, 6))
graph1 = fig.add_subplot(1, 2, 1)
graph1.fill_between(shangxing['ds'],shangxing['上行业务量 GB'], shangxing
['prophet'],facecolor="black", alpha=1,interpolate=True)
graph1.set_xticks(range(0, 168, 30))
graph1.set_xticklabels([index for index in graph1.get_xticks()])
graph1.set_ylabel("上行业务量 GB",fontsize=12)
graph1.set_xlabel("Steps(30hours)",fontsize=15)
graph2 = fig.add_subplot(1, 2, 2)
graph2.fill_between(xiaxing['ds'],xiaxing['下行业务量 GB'], xiaxing['prop
het'],facecolor="black", alpha=1,interpolate=True)
graph2.set_xticks(range(0, 168, 30))
graph2.set_xticklabels([index for index in graph1.get_xticks()])
graph2.set_ylabel("下行业务量 GB",fontsize=12)
graph2.set_xlabel("Steps(30hours)",fontsize=15)
```

程序 10：问题二代码	运行环境：Anaconda Python 3

```
def data_pro(df):
    df.loc[df['日期']=='018-04-04',"日期"] = '2018/4/4'
    df.loc[df['日期']=='018-04-01',"日期"] = '2018/4/1'
    df.loc[df['日期']=='018-04-09',"日期"] = '2018/4/9'
    df.loc[df['日期']=='018-04-02',"日期"] = '2018/4/2'
    df.loc[df['日期']=='018-04-10',"日期"] = '2018/4/10'
    df.loc[df['日期']=='018-04-08',"日期"] = '2018/4/8'
    df.loc[df['日期']=='018-04-03',"日期"] = '2018/4/3'
    df['date'] = (df['日期']+' '+df['时间']).apply(lambda x:to_datetime
(x))
    df.sort_values("date",inplace=True)
    return df

def processing(df,code,name):
    #对数据进行去重操作
    df = df.drop_duplicates().reset_index(drop=True)
    df = df.fillna(method='ffill')
    #统计数据的日期数量存进字典
    t = dict(df['日期'].value_counts())
    data = pd.DataFrame({'日期': list(t.keys()),'num': list(t.values())})
    data = data.loc[data['num']!=24]
    columns = data['日期'].values
    v = pd.DataFrame(columns)
    v.columns = ['columns']
    v['columns'] = pd.to_datetime(v['columns'])
    v['columns'] = v['columns'].apply(lambda x : x.strftime('%Y-%m-%d'))
    if len(df)!=0:
        for i in range(len(columns)):
            x = pd.date_range(start=v['columns'].iloc[i],periods=24,freq=
'h')
            x = pd.DataFrame(x,columns=['date'])
            df = pd.merge(df,x,how='outer')
            df['日期'] = df["date"].apply(lambda x : x.strftime('%Y-%m-%d
'))
            df['时间'] = df["date"].apply(lambda x : x.strftime('%H:%M:%S
'))
```

```
            df['小区编号'] = code
            #均值填充
            df_q = df[df['日期']==v['columns'].iloc[i]]
            df = df.fillna(df_q.mean())
            #df = df.fillna(df.mean())
            df = df.reset_index(drop=True)
        return df
    else:
        return df

def combine(df,x,y):
    combine = df[x].groupby(df[y]).agg('sum')
    combine = pd.DataFrame({
        'ds': combine.index,
        'y': combine.values,
  })
    return combine

def model_prophet(df,time1,time2):
    model = Prophet()
    model.fit(df)
    future = pd.date_range(time1,time2)
    future = pd.DataFrame(future)
    future.columns = ['ds']
    future['ds']= to_datetime(future['ds'])
    forecast = model.predict(future)
    return forecast['yhat']

path = ' /长期验证'
sub = pd.read_csv('/长期验证.csv',encoding='gbk')
future = pd.date_range('2020-11-01','2020-11-25')
future = pd.DataFrame(future)
future.columns = ['ds']
future['ds']= to_datetime(future['ds'])
files = os.listdir(path)
Code = []
#创建一个空的dataframae
```

```
data = pd.DataFrame(columns = ["code","日期",
                               '上行业务量 GB',"下行业务量 GB"])
#上行
for i in tqdm(files):
    code = int(os.path.splitext(i)[0][3:])
    print(code)
    df = pd.read_csv(path+'/'+i)
    if len(df)!=0:
        df = data_pro(df)
        df1 = processing(df,code,'上行业务量 GB')
        df2 = processing(df,code,'下行业务量 GB')
        df_up = combine(df1,'上行业务量 GB','日期')
        df_down = combine(df2,'下行业务量 GB','日期')
        if len(df_up)!=0 and len(df_up)!=1:
            forecast1 = model_prophet(df_up,'2020-11-01','2020-11-25')
            forecast2 = model_prophet(df_down,'2020-11-01','2020-11-25')
            df_x = pd.DataFrame({
                  'code': code,
                  '日期':future['ds'],
                  '上行业务量 GB': list(forecast1),
                  '下行业务量 GB': list(forecast2),
                  })
            data = data.append(df_x)
          #print(data.tail())
        else:
          #将需要最后单独处理的小区号提出
          Code.append(code)
    else:
        Code.append(code)
time = data['日期'].head(25)
df_none = pd.DataFrame(columns = ["code", "日期","上行业务量 GB","下行业务
量 GB"])
for i in range(len(time)):
    data1 = data[data['日期']==time.iloc[i]]
    #将单独异常的小区号赋予一个新的dataframe，上行业务量GB 和下行业务量GB 初始
均为0
    df_x = pd.DataFrame({
```

```
    'code': Code,
    '日期': time.iloc[i],
    '上行业务量 GB': 0,
    '下行业务量 GB': 0
    })
    data1 = data1.append(df_x)
    #取出上行业务量 GB 大于 0 的均值并赋予上行业务量 GB 小于等于 0 的值
    data2 = data1[data1['上行业务量 GB']>0]
    q = data2['上行业务量 GB'].mean()
    data1.loc[data1.上行业务量 GB<=0,'上行业务量 GB']=q
    #取出下行业务量 GB 大于 0 的均值并赋予下行业务量 GB 小于等于 0 的值
    data3 = data1[data1['下行业务量 GB']>0]
    p = data3['上行业务量 GB'].mean()
    data1.loc[data1.下行业务量 GB<=0,'下行业务量 GB']=p
    df_none = df_none.append(data1)

df_none.columns = ['小区编号', '日期', '上行业务量 GB', '下行业务量 GB']
#将日期格式修改为提交格式集
df_none['日期'] = pd.to_datetime(data['日期'])
df_none['日期'] = df_none['日期'].apply(lambda x : x.strftime('%Y/%-m/%-d
'))
#待预测文件并不是每一个小区都为 25 天，取预测结果与提交结果的交集
sub=pd.read_csv('./长期验证.csv',encoding='gbk')
submit = sub.drop(['上行业务量 GB', '下行业务量 GB'],axis=1)
intersection_result = pd.merge(submit, data)
intersection_result.to_csv('./submit2.csv',index=0)
```