

数据挖掘第三次作业报告

姓名：孙浩然

学号：1652714

答辩时间：6月21日，周五

目录

数据挖掘第三次作业报告	
目录	
a题	
坐标转换	
数据归一化	
测试集分割	
数据筛选	
模型建立	
结果分析	
误差计算	
误差概率分布图	
中位误差，平均误差与90%误差	
结果可视化	
matplotlib	
百度地图	
b题	
栅格划分	
模型建立	
结果分析	
误差概率分布图	
中位误差，平均误差与90%误差	
结果可视化	
matplotlib	
百度地图	
c题	
序列分组	
模型构建	
batch_size 的选择	
结果分析	
误差概率分布图	
中位误差，平均误差与90%误差	
结果可视化	
matplotlib	
百度地图	

d题

结果分析

误差概率分布图

中位误差，平均误差与90%误差

结果可视化

matplotlib

百度地图

e题

数据问题

模型构建

结果分析

误差概率分布图

中位误差，平均误差与90%误差

结果可视化

matplotlib

百度地图

f题

模型构建

结果分析

误差概率分布图

中位误差，平均误差与90%误差

结果可视化

matplotlib

百度地图

a题

a题是我最先开始尝试的题目，编码过程中遇到了很多很多问题，在报告里详细的记录一下。其中很多方法也在后面的题目中应用到了，将在这里直接解释清楚，后面就不在赘述。

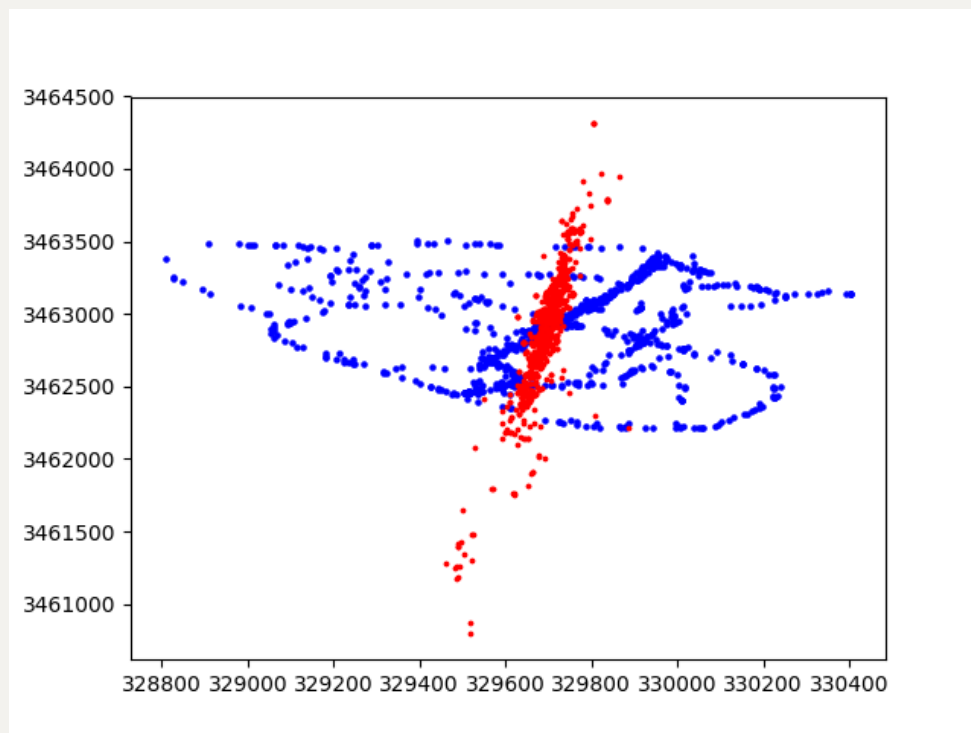
坐标转换

在模型的训练过程中，我一开始直接使用了原始数据中的经纬度坐标，即 (Latitude, Longitude) 坐标。但是紧接着，我发现在训练模型时，每层 epoch 所计算的损失函数的输出值非常非常小，一般为 $10e-4$ 的数量级，这样的数据既不利于模型的收敛，更不利于直观的观察数据的输出结果，也不适合最后计算距离准确坐标的距离，所以我最后将所有的坐标转化为了 utm 坐标，这类坐标的绝对值更大，计算出的损失函数更加直观。

另一个需要注意的是，由于数据小数点位数过多，所以为了更加准确的存储数据值，需要在将字符串类型的数据转化为 float64 类型，`x = np.array(X).astype('float64')`，如使用 float32 类型则会丢失掉很多数据。

数据归一化

在a题模型的最初版本，我并没有对数据进行归一化或标准化操作，得到的模型效果非常差，中位误差大概在300左右，预测值和准确值的关系大致如下(红色的点为预测值，蓝色的点为真实值)：



后来我意识到，由于每个 MR 的样本中不同的特征值得绝对值差别非常大，比如 `Dbm_1` 和 `SignalLevel_1` 大小差距就非常大，也就造成了在模型的训练过程中，每个特征对于模型训练效果的影响也可能是天差地别的，所以需要对其进行归一化的操作，让所有的数据的绝对值相近，才可以让模型得出更好的效果。

测试集分割

在训练模型的过程中，我意识到了需要将模型的训练数据分成 9:1 两部分来区分训练集和测试集，以鉴别模型的训练的好坏，但是我最初直接将读取的全部数据按顺序分成了 9:1 两部分，导致模型在训练集的效果不是很好，当我将所有的训练集和测试集 plot 出来时，我才发现了问题：



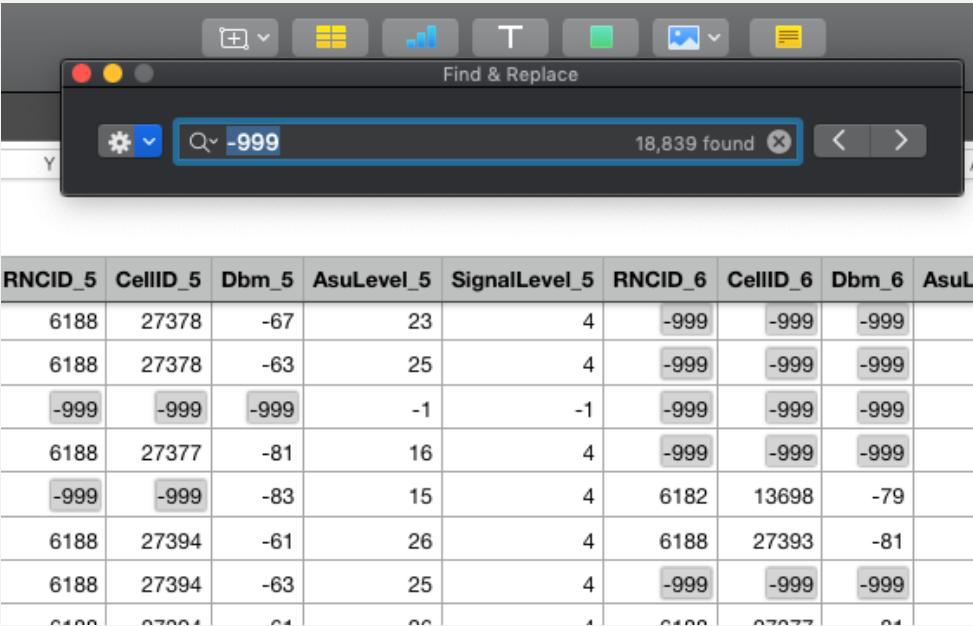
如果直接将数据按顺序分成训练集和测试集，那么数据的分布就可能像上图一样(蓝色的点为测试集，红色的点为训练集)，可以看出，所有的测试集的点都集中在了一起，而训练集的点的覆盖范围则大得多。

下图是我对数据进行了 shuffle 之后再切分训练集和测试集之后的效果，这时训练集和测试集就完全分开了，模型对测试集的预测结果更加能反应模型的好坏了。



数据筛选

虽然我们拥有 `train_2g.csv` 一整个文件作为训练集，但是并不是其中所有的数据都是有效的 MR 样本数据。比如有值为 -999 或 -1 的无效数据，如下图。



RNCID_5	CellID_5	Dbm_5	AsuLevel_5	SignalLevel_5	RNCID_6	CellID_6	Dbm_6	AsuL
6188	27378	-67	23	4	-999	-999	-999	
6188	27378	-63	25	4	-999	-999	-999	
-999	-999	-999	-1	-1	-999	-999	-999	
6188	27377	-81	16	4	-999	-999	-999	
-999	-999	-83	15	4	6182	13698	-79	
6188	27394	-61	26	4	6188	27393	-81	
6188	27394	-63	25	4	-999	-999	-999	
6188	27394	-61	25	4	6188	27393	-81	

所以在读入数据的过程中，我使用 `check_validation` 函数来检测一个 MR 样本数据是不是有效的。当然，我也不是一旦出现 -999 等无效数据就把一个 MR 样本丢掉，我尝试过，如果把全部的无效的数据全部丢掉，那么只会剩下大约 5500 条 MR 数据，相当于整体的数据量减少了整整一般，这显然是不可以接受的。所以在判断一条 MR 样本是否是有效数据时，我判断这个 MR 样本是否具有至少三个有效的基站数据，如果有效的基站数据少于三个，那么我就认为仅凭借这条 MR 样本是不足以确定一个准确的手机坐标的，就丢掉这个 MR 样本数据。如果有效基站数大于三且不到六，那么就随机从这条 MR 样本的有效基站数据中选取，补全到六个有效基站。

同时，由于我们还检查每条 MR 样本数据的 GPS 坐标的精度，即 `Accuracy` 项的内容，如果精确度超过了30米，那么我就认为这条 MR 样本数据是不够精确的，也就丢掉这一条 MR 样本数据。

经过上面两种筛选过程后，我们的数据最后可以省下大约 9400 条，大致在全体训练集的数据上减少了10%，我认为这是一个可以接受的范围。

模型建立

a题采用 CNN 的回归模型，我采用了两层卷积层，之后经过 `Flatten` 层就是全连接层，模型结构比较简单。

```
model.add(Conv2D(64, kernel_size=3, activation='relu',  
input_shape=(6, 5, 1)))  
model.add(Conv2D(128, kernel_size=3, activation='relu'))  
model.add(Flatten())  
model.add(Dense(32, activation='relu'))  
model.add(Dense(2))  
  
model.compile(optimizer=adam, loss='mean_squared_error')  
model.fit(X_Train, Y_Train, epochs=1500, batch_size=64)
```

当使用默认的 Adam 作为优化器时，我发现损失函数在后面的波动过大，我认为这是 learning rate 过大的一个表现，所以我不断地调小 learning rate 以减小波动，但是同时需要不断增大 epochs 才能让模型最后收敛。

经过不断的调整 learning rate 和 epochs，我最后得出了一个相对较好的结果。但是由于 epochs 过大，所以我将 batch_size 调整为 64，来加快模型的训练速度。

结果分析

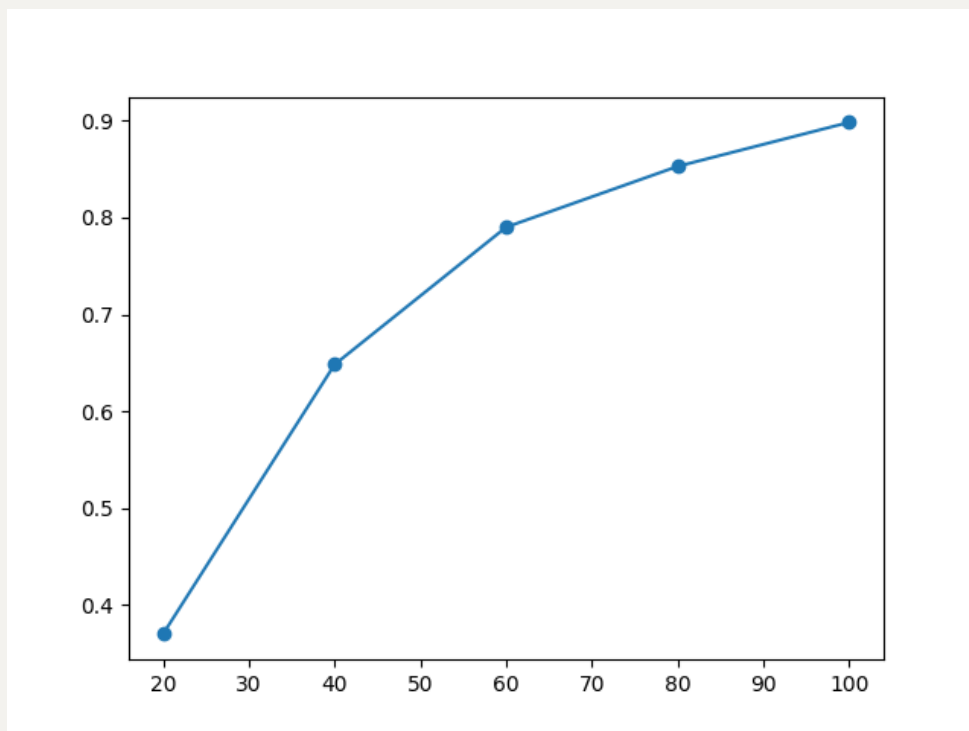
误差计算

在 utm 坐标下，两个点近似的处于一个平面内，所以可以直接勾股定理出两个点之间的距离，我使用这个距离作为预测的误差值，即真实值距离预测值相差的距离(单位：米)。

当然，在计算误差之前，需要将之前归一化的数据原模原样的还原回来。

误差概率分布图

可以从图中看出，一半的误差值都在30米之内，效果还是相对较好的。



中位误差，平均误差与**90%**误差

中位误差：26.67米

平均误差：45.64米

90%误差：101.48米

结果可视化

为了可以让结果更好的展示，我进行了两种可视化的操作，前一种一般作为测试时可以直接直观看到测试集数据的输出结果。

后一种则是直接在百度地图上展示了测试数据的输出结果。

matplotlib

b题

栅格划分

b题在数据处理方面与a题大同小异，一个主要的区别是，b的输出结果不再是经纬度坐标，而是一个 one hot 的概率矩阵。为了将数据变为概率矩阵，我们需要对地图进行划分，首先，我们需要找出地图的精确地最大最小坐标(文档中给出的精确度太低)，经过对数据的遍历，我得出：

```
max_latitude = 330414.05273900216
max_longitude = 3463503.3311055717
min_latitude = 328787.97245886514
min_longitude = 3462203.7096695383
```

根据最大最小值的差值，我大致估算 `latitude` 方向上划分 $16*2$ 个栅格，而 `longitude` 方向则划分 $13*2$ 个栅格。这样，每个栅格的大小就大致为 $50*50$ 米。(上文中 $*2$ 是因为我一开始设置为 $16*13$ ，但是经过尝试，发现如果两个方向同时扩大二倍的效果更好)。

所以我将地图一共分为了 832 个栅格，于是每个坐标都需要转化为一个 $1*832$ 维的概率矩阵，矩阵中所有数值的和为 1。其中值最大的位置即为这个点的分类的 label。

模型建立

这个模型结构与a题几乎相同，但与a题的回归模型不同是，b题采用的是多分类模型（832类），所以每一层的 units 数量相对增加了。并且最后一层的 `activation` 选用了 `softmax`。

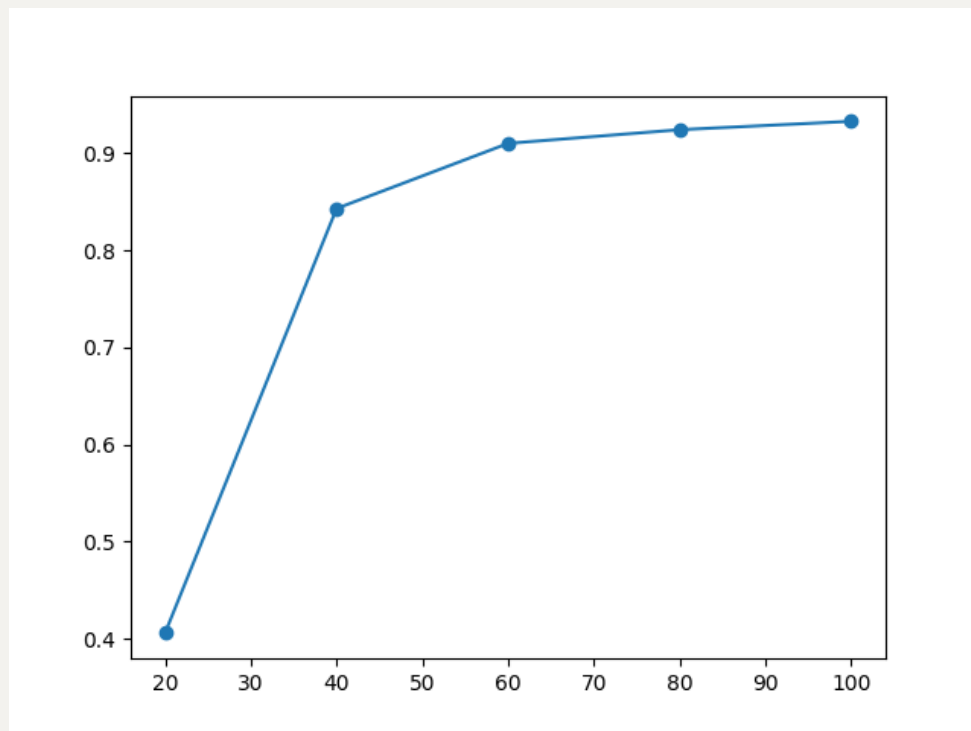
```
model.add(Conv2D(128, kernel_size=3, activation='relu',
input_shape=(6, 5, 1)))
model.add(Conv2D(256, kernel_size=3, activation='relu'))
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dense(256, activation='relu'))
model.add(Dense(la_size*lo_size, activation='softmax'))
```

结果分析

出乎意料的是，我没有想到b题的多分类模型的效果会比a题的回归模型的要好。但是结果出来之后，我仔细想了想，在b题中，由于使用了栅格，每个栅格的大小我设置的是 50×50 ，只要最后预测对了栅格，那么误差就不会差的太多，最多最多才能到70米。所以当栅格数量越多，大小越小的时候，误差应该就会越小。但是我发现，栅格的增多意味着分类的增多，也就意味着训练时间的不断增大。

误差概率分布图

对比a题的误差概率分布图可以看出，b题的误差值明显更加集中在小于40米的范围内，模型的效果也是更好的。



中位误差，平均误差与90%误差

中位误差：22.90米

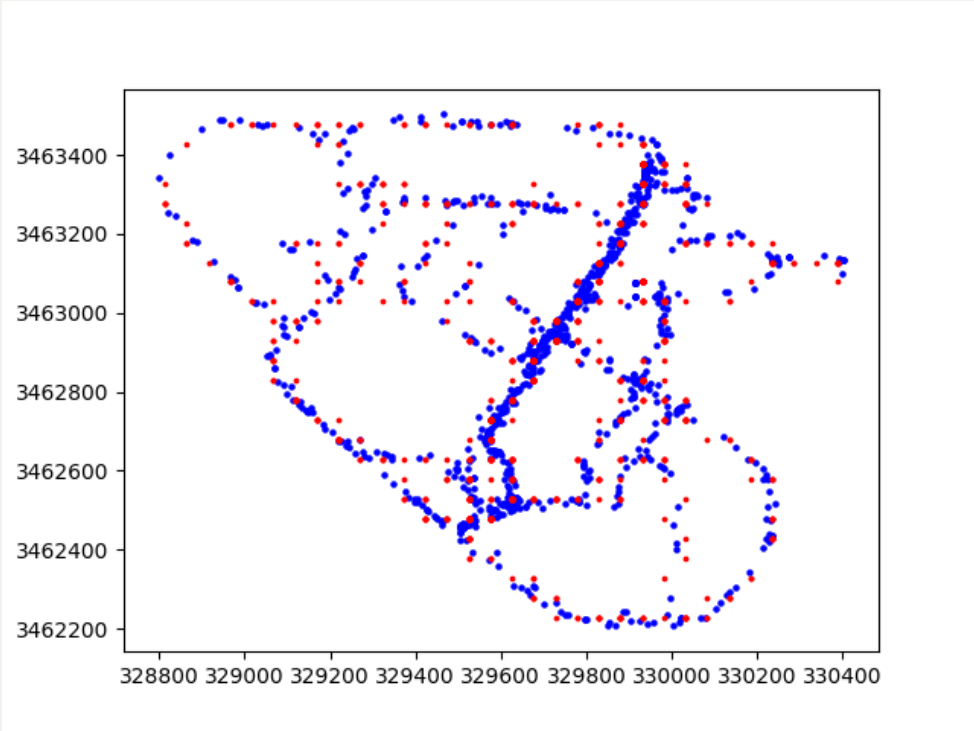
平均误差：43.87米

90%误差：55.68米

结果可视化

matplotlib

与a题不同的是，由于b题是对栅格的分类，所以当预测的点较为密集时，很有可能所有的点全部集中在一个栅格内了，导致下图中的红色的点的数量看起来相对较少，但是我认为那是因为有些点有很多是重复的。因为毕竟一个栅格内的点全部都算做了栅格的中心点。



百度地图



c题

序列分组

c题开始，不再使用 CNN 模型，改为使用 LSTM (Long Short Term Memory) 模型。在这个模型中，数据的单位不再是一条 MR 样本数据了，而需要是一个按照时间戳排序的 MR 样本序列。

我对测试集数据和训练集数据进行了简单的处理，发现，如果按照轨迹 ID 进行分组，那么最小的一个组会出现在测试集中，它的长度是 6，所以我选择 6 作为我的模型中样本序列的长度。

在分组的过程中，我还针对每个轨迹进行了分类，如果一个轨迹中所有的 MR 样本数刚好被 6 整除，那么正好分开；如果不能被 6 整除，那么就把剩下的点与前面的点连接起来，再次形成一个长度为六的 MR 样本序列。这种方法虽然会造成一些样本点的重复，但是也让每一个有效的训练数据进行了训练。

模型构建

LSTM 模型层的第一个参数代表了隐藏层数，其实我也不是很理解这个参数的含义，但是在不断地尝试过程中，我发现这个值越大，准确度越高，但是训练花费的时间也越长，最后经过取舍，选择了 240。

为了能够直接坐到 seq2seq 的训练，我还引入了 TimeDistributed，这个修饰器可以让 Dense 层上保持 LSTM 序列，从而可以每次训练一个序列。

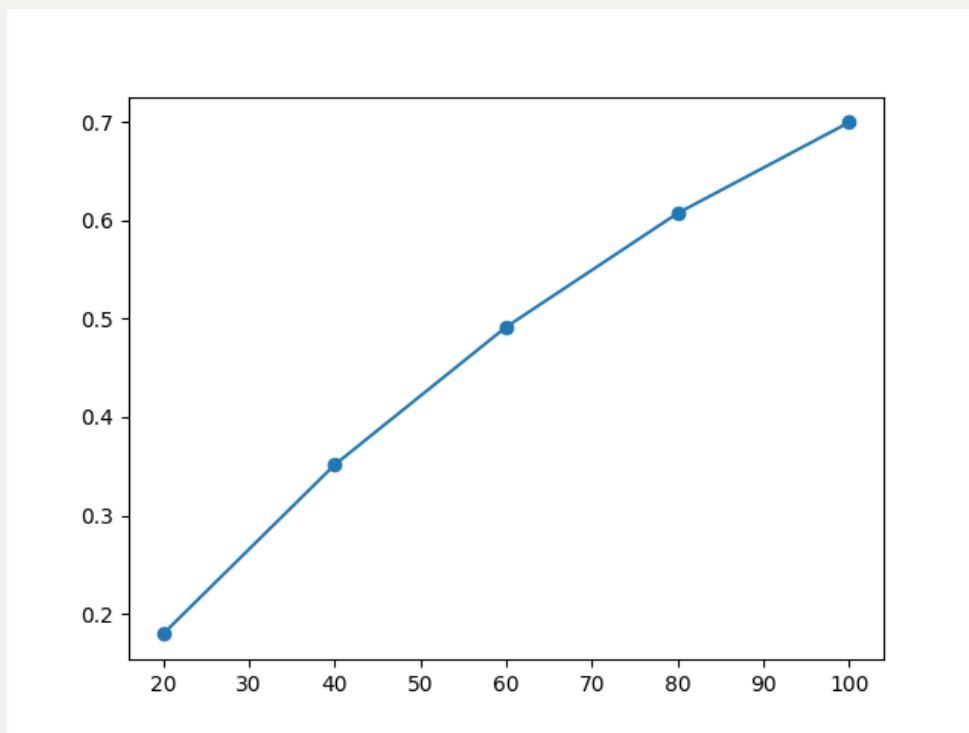
```
model.add(LSTM(slice_length*40, input_shape=(X_Train.shape[1], X_Train.shape[2]),
return_sequences=True))
model.add(TimeDistributed(Dense(72, activation='relu'))
model.add(TimeDistributed(Dense(2,
activation='linear')))
```

batch_size 的选择

从这个题往后，我的 batch_size 参数几乎都是选择的 6，因为我每个 MR 样本序列中就是包含 6 个 MR 样本，而且我在测试的过程中发现，当 batch_size 为 6 时，模型的训练效果也的确不错。

结果分析

误差概率分布图



中位误差，平均误差与**90%**误差

中位误差：61.43

平均误差：89.59

90%误差：203.04

结果可视化

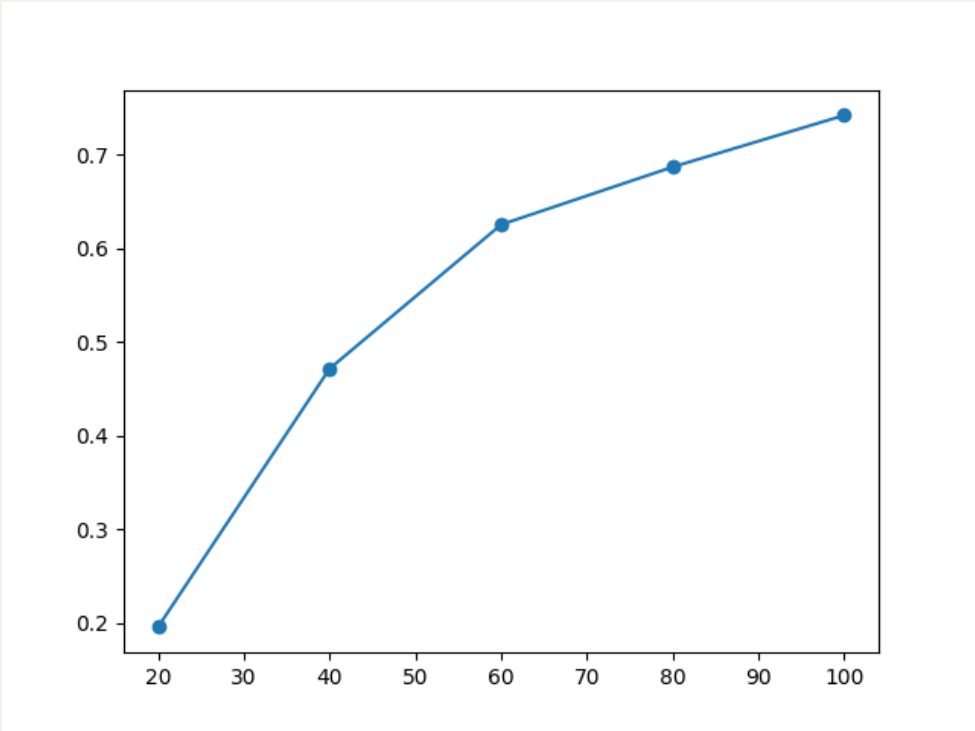
matplotlib

d题

栅格划分方法同b题，建模方法同c题。

结果分析

误差概率分布图



中位误差，平均误差与90%误差

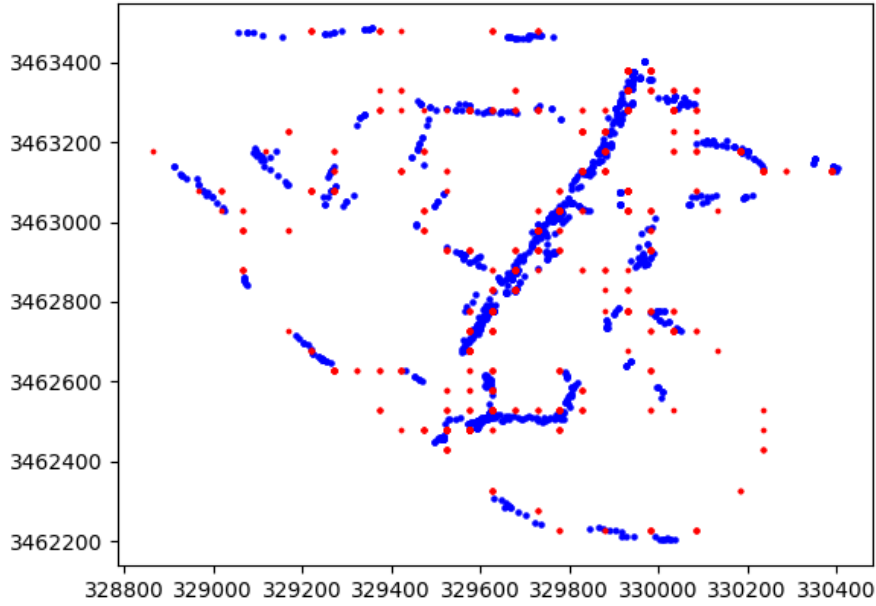
中位误差：43.10

平均误差：112.06

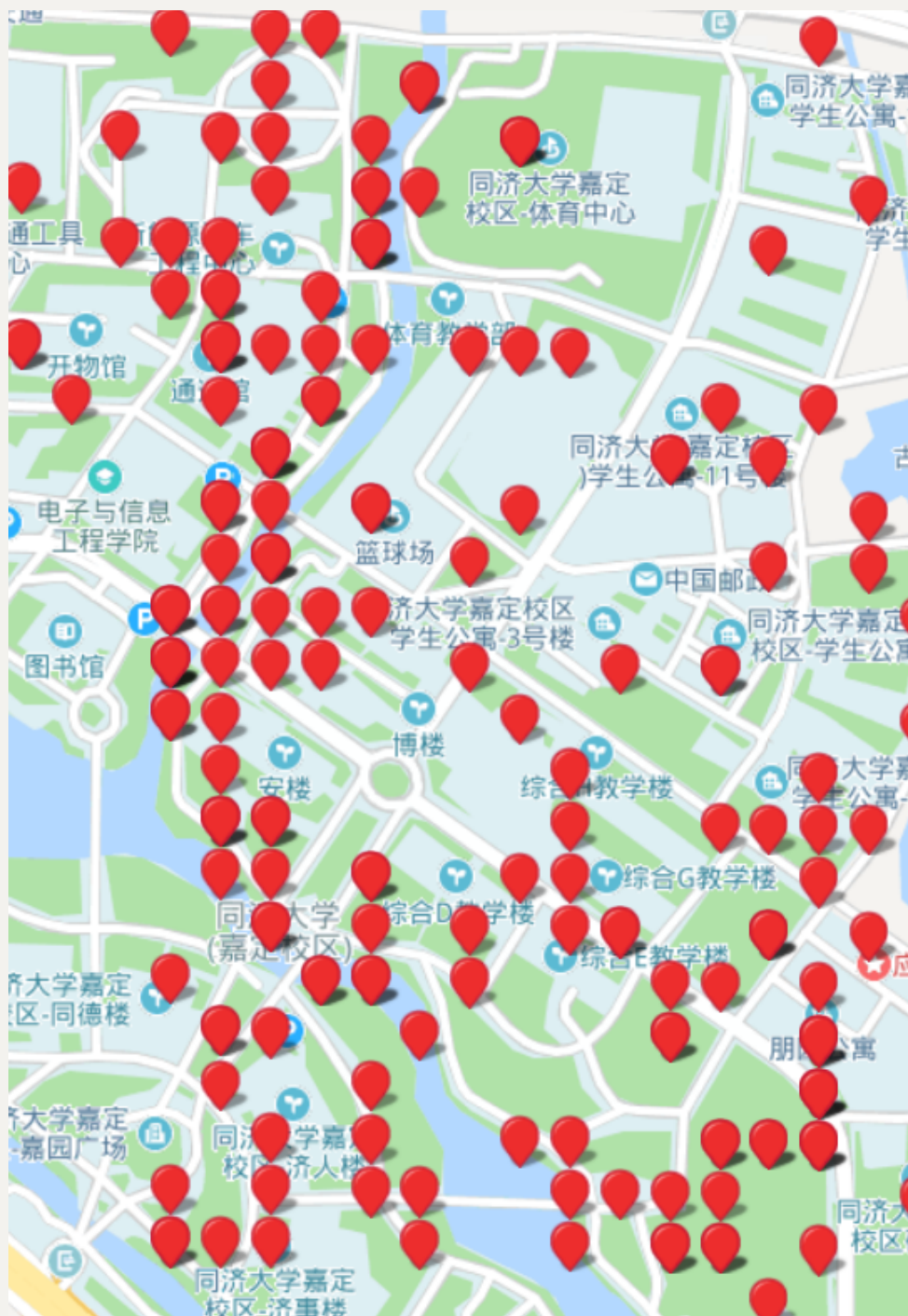
90%误差：346.19

结果可视化

matplotlib



百度地图



e题

数据问题

在建立e题的模型的过程中，我遇到的最大的问题是数据不足。

由于e题分为两大层（CNN 和 LSTM），所以我首先需要分配 50% 的数据给 CNN 层进行训练，之后使用训练好的第一层对剩下的 50% 进行预测，然后将预测完的 50% 再次按9：1分配作为第二层 LSTM 的训练集和测试集。

这样一来，第一层 CNN 只有 50% 的训练数据，而第二层 LSTM 则只有 45% 的训练数据。我认为数据不足是这个模型预测结果不好的一大原因。

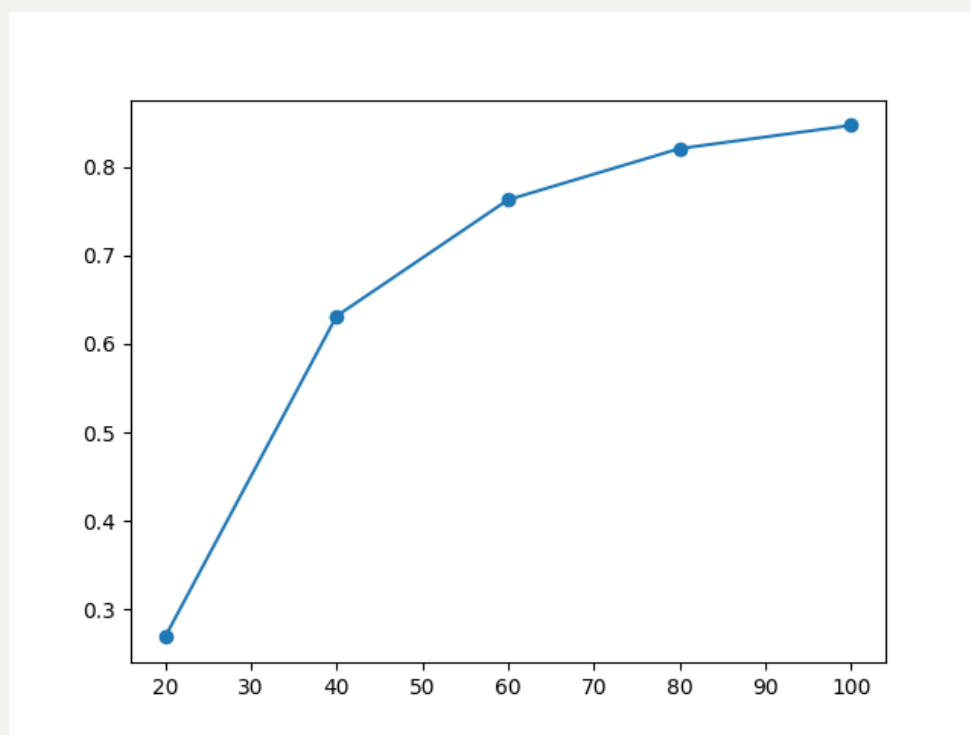
模型构建

e题的 CNN 模型与d题相同，LSTM 模型与d题相同。唯一的不同时，LSTM 模型需要转而使用概率矩阵序列作为输入，而不是 MR 样本序列，所以输入的 shape 受到了影响。

结果分析

e题的结果不如d题或是b题，我认为这充分说明直接尝试将两个模型连接起来不一定会让效果变得更好。第一个模型的误差可能会在第二个模型的误差上扩大。如果想让两个模型的优点更好的结合在一起，我觉得需要花费更多的精力。

误差概率分布图



中位误差，平均误差与90%误差

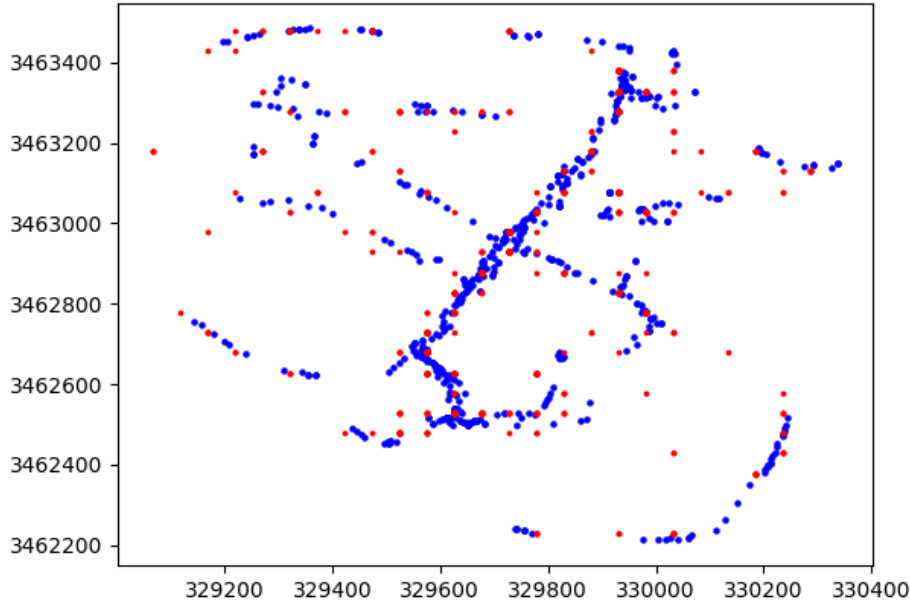
中位误差：31.48

平均误差：81.89

90%误差：241.41

结果可视化

matplotlib



百度地图



f题

模型构建

f题相对于d题来说，就是多了一个 autoencoder 层，经过搜索，这一个层主要用于特征提取和降维等作用。我们把 autoencoder 层放在 LSTM 层前，而在数据归一化后。在测试过程中也尝试过将 autoencoder 放置在数据归一化前，但是效果却是非常差。

```

inputs = Input(shape=(slice_length, features-1))
encoded = LSTM(slice_length*20)(inputs)

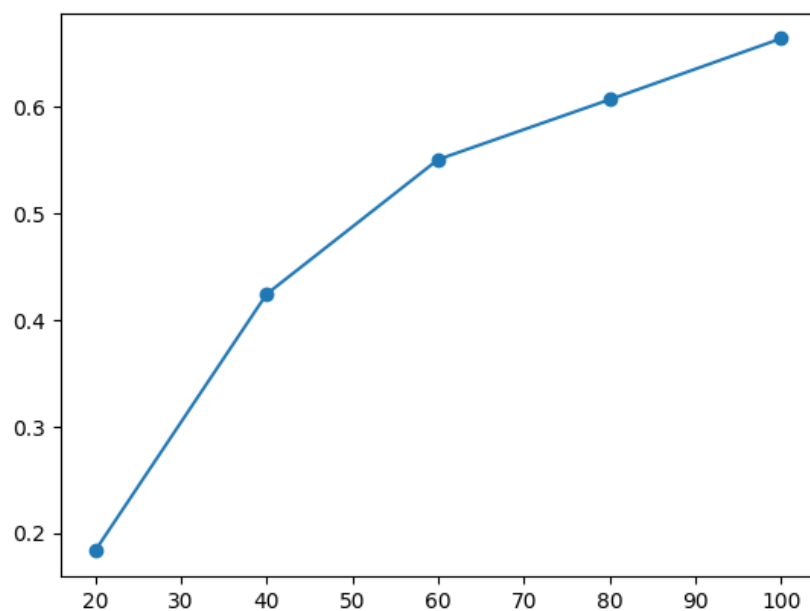
decoded = RepeatVector(slice_length)(encoded)
decoded = LSTM(features-1, return_sequences=True)
(decoded)

sequence_autoencoder = Model(inputs, decoded)
encoder = Model(inputs, encoded)
sequence_autoencoder.compile(loss='mean_squared_error',
optimizer='adam')
sequence_autoencoder.fit(X, X, epochs=100, batch_size=6,
shuffle=True)
X = sequence_autoencoder.predict(X)

```

结果分析

误差概率分布图



中位误差，平均误差与90%误差

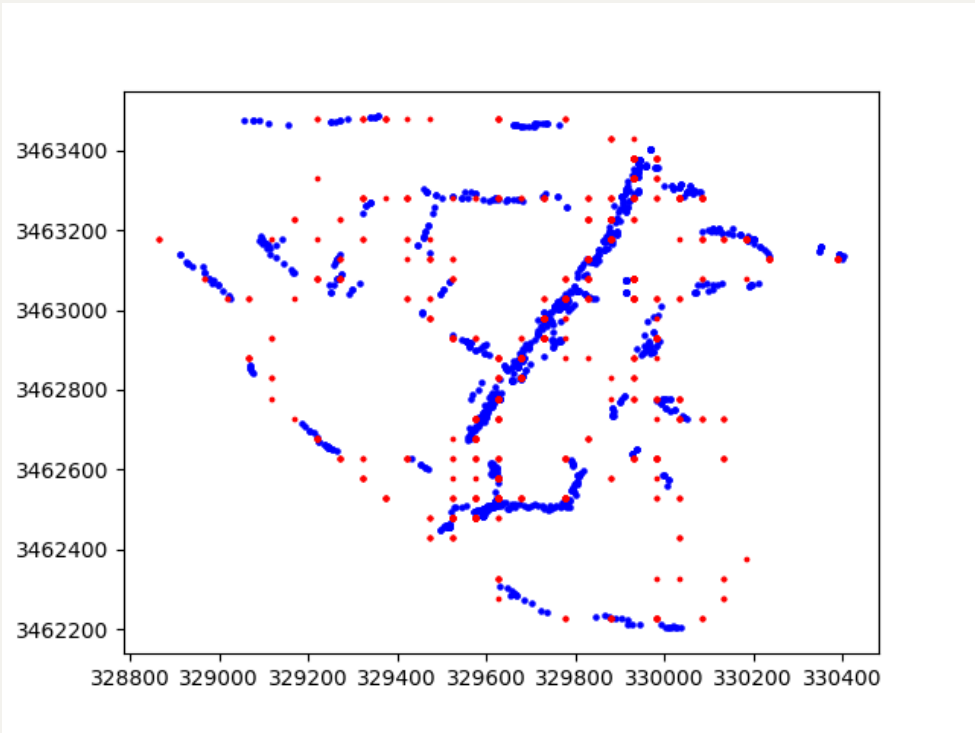
中位误差：50.98

平均误差：138.86

90%误差：430.03

结果可视化

matplotlib



百度地图

