中国研究生创新实践系列大赛
“华为杯”第十六届中国研究生数学建模竞赛

学　　校	中国科学技术大学
参赛队号	19103580027
队员姓名	1. 吴生生
	2. 张翰韬
	3. 石少猛

中国研究生创新实践系列大赛
“华为杯”第十六届中国研究生数学建模竞赛

题　目　　无线智能传播模型

摘　　要：

由于电磁信号在传播过程中容易受到多方干扰，极易发生透射、绕射、散射、反射、折射等情况，信号在传播过程中将会大量损耗，急需优质模型来辅助无线网络建设。本文通过对工程数据的深度挖掘与分析，针对 5G 技术下的无线信道建模问题，设计并提出了高效的无线智能传播预测模型。

针对问题一，通过分析现有数据，调研并理解 Cost 231-HATA 模型的原理及应用场景，并在算法调研中受到传统传播模型的启发，针对基站的空间位置信息设计出更为合理的特征。随后，通过对原有数据集进行统计分析、异常点检测，决定保留或构造如下特征：基站栅格与用户栅格的相对距离，小区发射机相对地面高度，小区发射机水平方向角，小区发射机垂直电下倾角，小区发射机垂直机械下倾角，小区发射机中心频率，小区发射机发射功率，基站真实海拔高度，基站与用户的真实海拔高度差，小区用户所在栅格地物类型索引以及基站所在栅格地物类型索引。

针对问题二，有效的数据和特征能给模型提供更高上限，所以对数据进行特征筛选就尤为重要。首先使用 LightGBM 模型量化各个特征对于平均信号接收功率 RSRP 的重要性；随后，在统计意义上选择使用 Pearson 相关系数计算特征与 RSRP 的相关性；同时，还考虑到特征内的方差。通过实验可以观察到，在这三种维度下，各特征相关性程度排序基本一致，最终得出结论，用户与基站距离、用户与基站的高度差是最为突出的重要特征，而小区发射机中心频率几乎可以忽略不计，其他特征重要性排序详见正文。

针对问题三，在设计和筛选了有效特征之后，需要利用更深层的 AI 模型对无限传播模型建模。我们分别选择了三种建模方法——全连接神经网络（DNN）、卷积神经网络（CNN）和 LightGBM。DNN 网络基于感知机理论，经过多层全连接后可以回归出 RSRP 值；CNN 网络经过卷积操作对数据进行回归，实验探索了针对无线传播场景，基于 Network In Network，ResNet， GoogleNet 三种经典结构设计的多种模型；最后使用 LightGBM 模型进行数据回归。经过三个版本的模型结果对比可以发现，LightGBM 取得的结果不仅在 RMSE、PCRR 等指标上最优，预测出的 RSRP 分布还十分接近真实值的分布；而基于 ResNet 的 CNN 效果优良，易于部署上线，取得了不错的效果。

关键字：特征提取　特征选择　平均信号接收功率 RSRP　LightGBM　CNN PCRR RMSE

目 录

一、问题重述

1.1 问题背景

随着通信技术的不断发展，网络用户对信号的要求也越来越高，5G 技术应运而生。目前，我国的华为公司 5G 技术在全球范围内已处于领先地位，且应用良好。5G 信号的传输是应用超高频电磁波进行的，频率越高速度越快，因此 5G 系统全面覆盖开展商业化的相关技术理论成为现实。但是生活中的通信并不是畅通无阻的，而是会受各种因素的影响。如何挑选模型并合理有效地选择基站地址则成为基站安装最大的问题。在整个无线网络规划流程中，高效的网络估算对于解决基站选址问题有着重要的意义。无线传播模型正是通过对目标通信覆盖区域内的无线电波传播特性进行预测，使得小区覆盖范围、小区间网络干扰以及通信速率等指标的估算成为可能。

在现实情况中，无线网络已经遍及全球，与人们生活息息相关。我们每天都在运用网络并产生大量的数据，所以如何利用已有信息搭建 5G 无线网络是我们亟待解决的难题。实际运用数据建模过程中，我们需要针对实际环境，处理数据，寻找特征，例如工程参数等，对传播模型进行校正与改进。

1.2 问题重述

运营商在部署 5G 网络的过程中，需要合理地选择覆盖区域（测试集数据中的覆盖区域为用户小区）内的基站站址，进而通过科学地部署基站来降低成本的同时，能够最大限度满足用户的通信需求。围绕基站选址问题的任务规划，本文依次解决以下问题：

问题 1：测试集数据中提供了 4000 个小区的各类数据，我们需要分析数据，并根据已知的 Cost 231-Hata 模型，分析并设计合适的特征，在后续建立预测模型是会以这些特征作为输入变量，说明这样设计的原因，以便于后续建立预测模型。

问题 2：根据测试集中的小区数据，我们分析并设计出多个合适的特征，题目要求我们计算这些特征与目标（平均信号接受功率）的相关性，并将结果按相关性排序，同时要说明原因和特征的计算方法。此外，可以自行根据需要，同时使用其他方方法或者模型处理特征数据。

问题 3：我们在解决了前两个问题之后，确定了建模所需要的特征。我们通过建立不同的预测模型来计算平均信号接收功率和均方根误差的值，并进行比较，分析模型的预测水平以及模型优缺点。

二、模型假设

根据特征选择和设计、模型建立和模型预测，我们需要进行如下假设：

假设 1　假设 RSRP 是题目所给的工程参数和地图数据中的所有变量的函数，且不受这以外的变量的影响。同时，本题所提供的数据是真实可靠的。

假设 2　假设相邻小区在 RSRP 测量时不会产生相互干扰，即同一个位置对两个小区进行测量时，两个小区的 RSRP 值不会叠加。

假设 3　假设对地图进行栅格化处理是合理的，一方面，每个栅格的范围只有 5 米，在这么小的范围内，地理特征几乎不会变，无论是海拔高度还是地物类型，都很难发生变化，因此可以假定栅格左上角的位置就代表了处于该栅格的用户或者基站的实际位置。

三、参数与符号说明

3.1 参数说明

根据 4000 个小区各类数据的分析我们保留了以下参数，如表所示。

表 1：保留参数数据的说明

字段名称	含义	单位
Distance	基站栅格与用户栅格的相对距离	m
Height	小区发射机相对地面高度	m
Azimuth	小区发射机水平方向角	Deg
Electrical Downtilt	小区发射机垂直电下倾角	Deg
Frequency Band	小区发射机中心频率	MHz
RS Power	小区发射机发射功率	dBm
Cell Attitude + Height	基站高度 + 基站位置海拔高度	m
Height Difference	基站与用户的真实海拔高度差	m
New Clutter Index	小区用户所在栅格地物类型索引	-
New Cell Clutter Index	基站所在栅格地物类型索引	-

3.2 符号说明

表 2：模型中符号的含义说明

符号	意义
x	神经网络的任意层输入
n	神经网络输入的样本个数
c	神经网络中的特征通道数
h	神经网络中任意层的高
w	神经网络中任意层的宽
k	卷积核的高
m	卷积核的宽
c_1	卷积层输入通道数
c_2	卷积层输出通道数
b_1	自定义残差块的大卷积尺寸
b_2	自定义残差块的通道数

b_3	自定义 inception 模块的输入通道数
TP	真阳性率
FP	假阳性率
FN	假阴性率

四、问题一的分析与求解

4.1 问题一的分析

高效的机器学习模型建立依赖于输入变量与问题目标的强相关性，因此输入变量也称为 “特征”。特征工程的本质是从原始数据中转换得到能够最好表征目标问题的参数，并使得各个参数的动态范围在一个相对稳定的范围内，从而提高机器学习模型训练的效率。

因此，选择与问题目标具有强相关的输入变量对于模型的建立和实际问题问题的解决非常重要。

4.2 问题一的求解

4.2.1 特征的来源

通过对 Cost 231-Hata 模型以及数据集信息的分析，我们以下列方式作为模型备选特征的数据源：

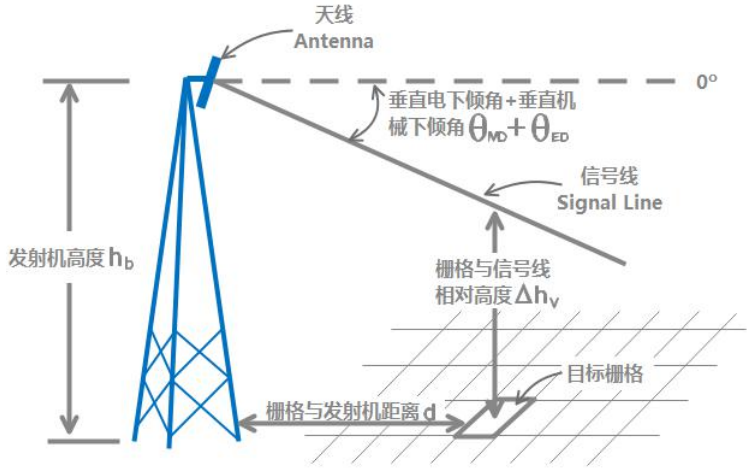

图 1：根据目标栅格与发射机的地理位置关系提取特征

1. 坐标距离

根据图 4，我们取 Cell X 与 X 的差值，Cell Y 与 Y 的差值为新坐标，并根据新坐标[（Cell X）-X，（Cell Y）-Y]，求出栅格与发射机的距离 d 的数值，作为一个备选特征。理由：小区所属站点的栅格位置（Cell X，Cell Y）并不是以基站所在位置为坐标原点，其数值相对于小区面积较大，进而栅格的位置坐标（X，Y）的数值也较大，两个坐标值的绝对值远超其他数据，直接输入模型会导致模型误差变大。此外一点，在其他条件不变的情况下，

绝对坐标没有任何意义，**相对距离**才能决定信号传播的质量。

我们随机选取一个小区（小区标识：112001），以坐标[X-（Cell X）,Y-（Cell Y）]表示每个栅格相对于基站栅格的位置。这里差值之后，就可以把基站位置作为坐标原点，差值坐标则表示栅格相对于基站原点的位置，每个栅格有与其对应的 RSRP 的值，我们将坐标与其对应的 RSRP 的值进行可视化处理，得到直观的坐标距离 d 与 RSRP 的关系，如图 5 所示：

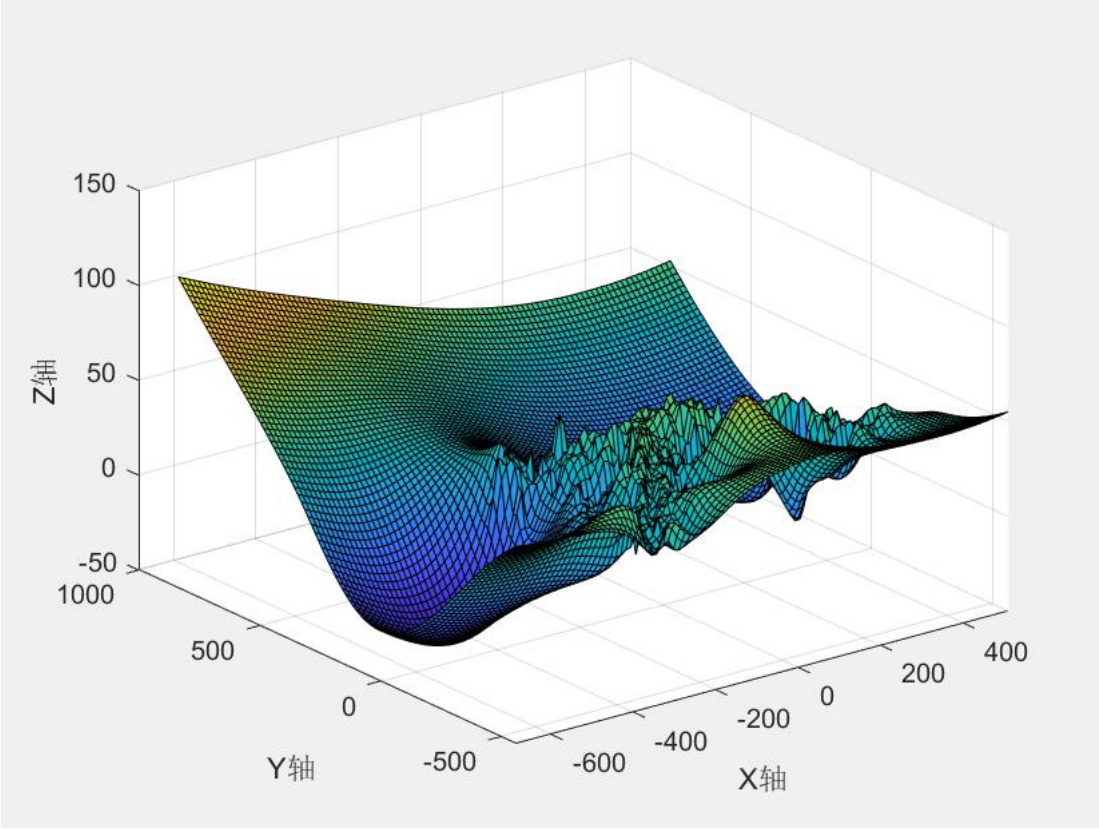

图 2：小区中不同栅格位置的 RSRP 强度三维图

图 5 中，x 轴和 y 轴对应的就是原始数据中的 X 和 Y 所在的轴，Z 轴表示栅格所在位置经过处理的 RSRP 值。它表示用每一个 RSRP 值分别减去它们中的最小值得到。我们把所有点的每一个 RSRP 值，转换成一个固定的颜色。每个位置上的颜色代表了该点对应的小区用户栅格处的 RSRP 值，蓝色的区域代表 RSRP 值较大，绿色的区域代表 RSRP 值一般，黄色的区域代表 RSRP 值较小。我们从图中可以看出如下几点：

（1）距离基站很远的位置，RSRP 值很小，而大多数距离基站较近的位置，RSRP 值都较大；

（2）真实海拔高度相对于基站所在栅格的真是海拔高度差距较大的栅格，RSRP 值往往较低；

（3）有的区域内，所有栅格的真实海拔高度与小区基站的真实海拔高度相当或者略微偏低，这样的情况下，在很大的范围内，RSRP 值的变化并不会很大；在一些离基站比较近的区域，因为地形以及建筑高度的变化比较频繁，RSRP 值也产生了较大的波动。

（4）在一些离基站比较近的区域，因为地形以及建筑高度的变化比较频繁，RSRP 值也产生了较大的波动。

2. 基站与用户所处位置的真实高度

我们考虑基站的所处位置的真实高度时（这里基站本身的高度为 Height 表示基站发射机到地面的高度），将海拔数据考虑进去，以两者之和，即 Cell Altitude + Height（记为 CA + H）作为一个备选特征。

在第二项数据源中，我们注意到所给测试集数据中存在两个海拔数据，小区站点所在栅格(Cell X, Cell Y)的海拔高度和其他位置栅格(X,Y)上的海拔高度。高度不同的地方，空气密度不同，电磁传播过程中的折射受到影响。位置越高，空气密度越稀薄，空气折射率越小，电磁信号的折射就越小。这里我们对不同位置**栅格的海拔高度和其上的建筑物高度**取和，作为一个备选特征。此外，不同栅格上的建筑物高度也不同。在相距较近的地方，海拔相差不多，从数据中可以直观地感受到，如图 6 所示。

此外，选择基站所处位置的真实高度作为特征的另一个原因：基站的海拔，基站发射机到地面的高度，对于小区里的所有用户从平均意义上来说，是同一种影响因素。而如果考察基站与小区内单个用户的关系，那么基站真实高度和用户真实高度的差值是关键影响因素。这就是我们选择基站所处位置的真实高度、基站与用户高度差这两个特征的原因。

这里我们以第一个特征选取中随机抽取的小区（小区标识：112001）为例，以基站所在位置为原点，也即以上述第一项数据源中的坐标[X-（Cell X）,Y-（Cell Y）]为栅格位置，画出不同位置栅格的等高线，如图 6 所示。据图可以直观地看出，在小区里距离较近的地方，海拔差距并不明显。

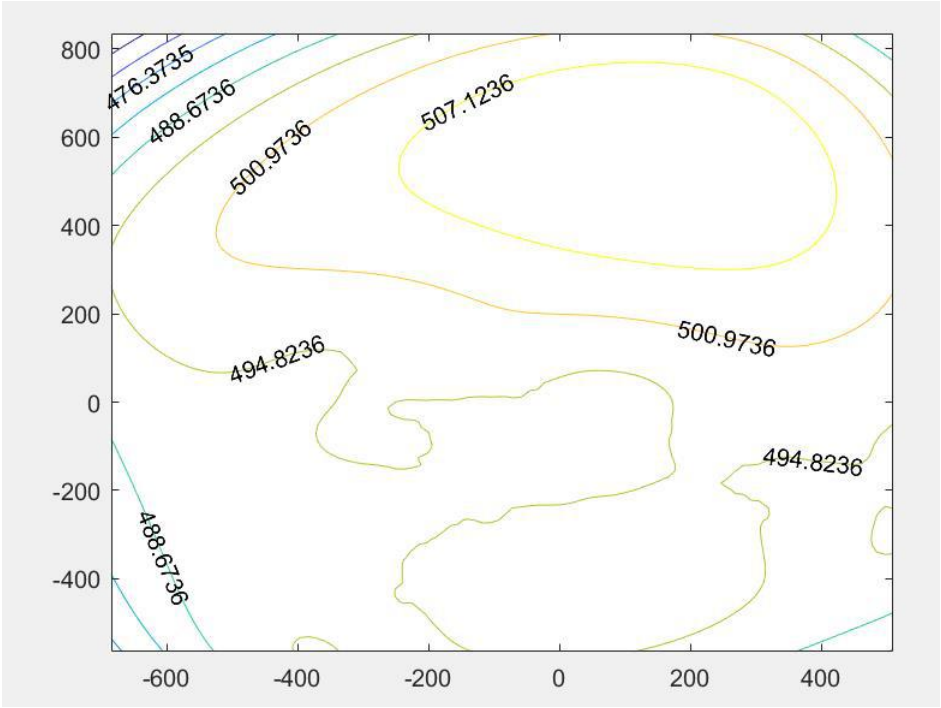

图 3：小区中不同栅格位置的海拔等高线图

这里，我们保留基站的实际高度，即 CA + H，再以栅格(X,Y)上建筑物的实际高度（即建筑物高度与所处海拔高度之和）作差取值，得到一组数据。由几何方法可以得出：

delta_h = (CA + H) - (BH + A) - d * tan(ED + MD)

我们以此差值数据作为模型的一个备选特征。

3.地貌特征

对于地物类型索引，我们计算出 20 类地貌类型（Clutter Index）每一类的出现次数，编制出表 5：

表 3：不同地物类型出现频率

Clutter Index	含义	出现次数	Clutter Index	含义	出现次数
1	海洋	0	11	城区高层建筑（40m~60m）	151335
2	内陆湖泊	157233	12	城区中高层建筑（20m~40m）	718328
3	湿地	0	13	城区<20m 高密度建筑群	776184
4	城郊开阔区域	0	14	城区<20m 多层建筑	368303
5	市区开阔区域	6159532	15	低密度工业建筑区域	135082
6	道路开阔区域	2397951	16	高密度工业建筑区域	12374
7	植被区	739713	17	城郊	13714
8	灌木植被	108659	18	发达城郊区域	4492
9	森林植被	0	19	农村	0
10	城区超高层建筑（>60m）	268933	20	CBD 商务圈	0

由表 5 可以看出 1, 3, 4, 9, 19, 20 这几个类出现次数为零，所以我们将其删除，所以 1-20 改成 1-14，2 -> 1, 5 -> 2，6 -> 3，18 -> 14，整理后的 Clutter Index 重新排序如下述表 6 所示：

表 4：新排列的地物类型表

Clutter Index	含义	Clutter Index	含义
1	内陆湖泊	8	城区中高层建筑（20m~40m）
2	市区开阔区域	9	城区<20m 高密度建筑群
3	道路开阔区域	10	城区<20m 多层建筑
4	植被区	11	低密度工业建筑区域
5	灌木植被	12	高密度工业建筑区域
6	城区超高层建筑（>60m）	13	城郊
7	城区高层建筑（40m~60m）	14	发达城郊区域

此外，基站所处位置栅格的地貌类型（Cell Clutter Index），其处理方法同上。

4. 天气因素

天气因素会影响到信号的传输效率，例如空气 PM2.5、湿度、温度等的不同，会影响到信号传播。PM2.5 浓度越大，即空气中可吸入颗粒物体越多，光线从其他地方传播到这里时，该空间由于 PM2.5 导致空气密度变大，光线传播在从低密度介质传播到高密度介质时，会偏移原来的直线传播方向，加大其光线的折射率，影响测试点的信号强度，进而影响传输效率（海市蜃楼的形成原因亦是光线在从低密度介质进入高密度介质时，会改变原来的直线传播方向）；空气湿度也是同理，湿度越高，空气中含水量越高，会加大该空间介质的密度，影响光线的传播。此外，一天之中，天气多数情况下会随着时间的推移产生变化，在从早晨到中午时，温度逐渐升高，空气湿度会降低，进而导致空气介质密度相对

变大。由于天气数据在测试集数据中没有体现出来，我们这里忽略其影响。

4.2.2 异常值检测

1. 检测原理：主成分分析（Principle Component Analysis），简称 PCA。PCA 的原理是通过构造一个新的特征空间，把原数据映射到这个新的低维空间里。PCA 可以提高数据的计算性能，并且缓解“高维灾难”，保留原数据最有效和最大限度的信息。

用 PCA 进行异常检测的原理是：PCA 在做特征值分解之后得到的特征向量反应了原始数据方差变化程度的不同方向，特征值为数据在对应方向上的方差大小。所以，最大特征值对应的特征向量为数据方差最大的方向，最小特征值对应的特征向量为数据方差最小的方向。原始数据在不同方向上的方差变化反应了其内在特点。如果单个数据样本跟整体数据样本表现出的特点不太一致，比如在某些方向上跟其它数据样本偏离较大，可能就表示该数据样本是一个异常点。

2. 检测步骤：

① 对数据归一化处理，使用数据和均值的差，与方差做商；

② 初始化一个 PCA 对象，用默认参数对所有成分进行保留（注意：这里由于数据本身的特征列就很少，也没有方差特别大或者特别小的主成分，方差中最大的只有 12%，最小的 2%）。对标准化后的数据进行训练，得到基本的 PCA 模型；

③ 用 PCA()对步骤①中取样的数据进行降维，计算数据样本在该方向上的偏离程度；

④ 计算主成分和次成分的阈值。这里我们取的是成分是解释数据集 90%的方差成分；

⑤ 计算出训练数据的阈值，这里异常点的阈值计算出来是 0.017。这里的阈值跟归一化过后的输入相比，差距也不大。

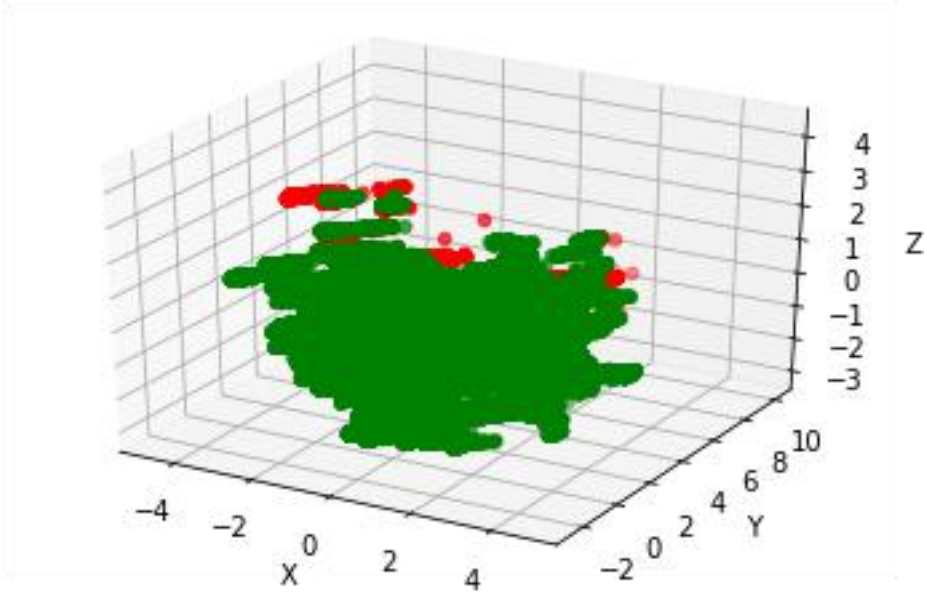

图 4：异常值分析图

在图 7 的数据中，我们取了前三个主成分，在训练数据中随机抽取了一百万个样本，异常值标为红点，其他样本标为绿点。从图 7 中可以看出两点：（1）前三个主成分所占方差的比例都不算太大；（2）从计算公式上来看，异常值确实是离主成分最远的值，但另一方面，我们找出的异常值距离大部分数据都不是很远，仅仅只是在数据的边缘。因此，是否去除这一部分样本，对模型的影响应该不大，我们最终选择不处理。

4.2.3 特征的选取

这里我们罗列出所有的特征如下表 5 所示：

表 5：整合的特征名称

特征名称	含义	单位
Distance	基站栅格与用户栅格的相对距离	m
Height	小区发射机相对地面高度	m
Azimuth	小区发射机水平方向角	Deg
Electrical Downtilt	小区发射机垂直电下倾角	Deg
Frequency Band	小区发射机中心频率	MHz
RS Power	小区发射机发射功率	dBm
Cell Attitude + Height	基站实际高度= 基站位置海拔高度 + 基站高度	m
Height Difference	基站与用户实际高度差=基站高度与基站海拔和 — 用户位置的建筑物高度与海拔高度	m
New Clutter Index	重新排列的小区用户所在栅格地物类型索引	-
New Cell Clutter Index	重新排列的基站所在栅格地物类型索引	-
Electrical Downtilt	小区发射机垂直电下倾角	Deg
Clutter Index	小区用户所在栅格地物类型索引	-
Cell Clutter Index	基站所在栅格地物类型索引	-
Cell Building Height	基站位置上的建筑物高度	m
Building Height	用户栅格位置上的建筑物高度	m
Altitude	用户栅格位置的海拔 A	m
X	用户栅格位置，X 坐标	m
Y	用户栅格位置，Y 坐标	m
Cell X	基站栅格位置，X 坐标	m
Cell Y	基站栅格位置，Y 坐标	m
RSRP	平均信号接收功率	dBm

经过以上分析，我们删掉的特征为：基站位置上的建筑物高度 Cell Building Height；基站位置海拔高度 Cell Attitude；基站自身到地面的高度 Height；用户栅格位置上的建筑物高度 Building Height；用户栅格位置的海拔高度 Altitude；用户栅格位置和基站栅格位置的四个坐标点 Cell X 、Cell Y、 X、 Y。

五、问题二的分析与求解

5.1 问题二的分析

基于提供的各小区数据集，我们根据如下因素来分析设计特征。

设计因素：电磁信号在传播过程中导致损耗的影响可以来源于多个方面，例如发射机的实际安装情况，所处地理位置情况等等，我们在工程参数和地图参数中可以找到并计算出所需要的变量。如用户栅格与基站栅格的相对距离 Distance，基站栅格的实际高度与用户栅格实际高度的差 Height Difference，以及各个栅格位置的地貌特征。另外，发射机的安装情况，如下倾角度、实际高度等，会影响标签数据 RSRP。我们根据相关性的绝对值大小倒序排列后可以大致的对各个特征对平均信号接收功率 RSRP 的影响有初步的了解，以便于我们后续根据建立模型并进行相应调整。

首先我们通过 LightGBM 模型选出特征，直观分析各个特征的重要性；然后我们计算各个特征与平均信号接受功率 RSRP 的相关性与方差，通过量化指标分析特征对 RSRP 的影响。

5.2 问题二的求解

5.2.1 LightGBM 特征选择

1. LightGBM 模型原理

提升树是一类利用加模型与前向分布算法实现学习的优化过程，最具代表性的方法包括 GBDT，XGBoost 等等。他们继承了树模型的优点，具有很强的可解释性，是处理结构化数据的利器；更重要的是，他们作为集成学习的代表方法，通过集成的方式，把树模型的性能大大提高。GBDT 采用负梯度作为节点划分的指标，XGBoost 则利用到二阶导数。强大的可解释性和数据驱动的特征分裂方式，给我们的特征选择带来了更加可靠的依据。然而，他们共同的不足是，确定分裂节点需要扫描所有样本，在面对大量数据或者特征维度很高时，效率很难使人满意，尤其是针对本题，训练样本数量在一千万左右，我们更希望使用一个高效的特征选择算法。

LightGBM 提出了很好的解决方案：从减少样本的角度，LightGBM 排除大部分小梯度的样本，仅用剩下的样本计算信息增益。从减少特征的角度，通过捆绑互斥特征提高效率。在训练过程中，AdaBoost 采用权重很好诠释了样本的重要性，GBDT 没有这种权重，但是每个数据样本的梯度可以被用来做采样的信息。如果一个样本的梯度小，那么表明这个样本已经训练好了，它的训练误差很小了，我们可以丢弃这些数据。LightGBM 保存大梯度样本，随机选取小梯度样本，并为其弥补上一个常数权重，从而更关注训练不足的样本，同时也不会改变原始数据太多。

大部分决策树的学习算法通过 level(depth)-wise 策略生长树，如图 5：

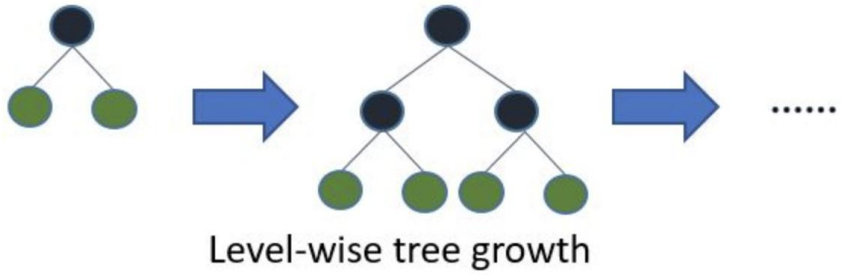

图 5：策略生长数

LightGBM 通过 leaf-wise 策略来生长树，它将选取损失函数值增长最大的叶节点来

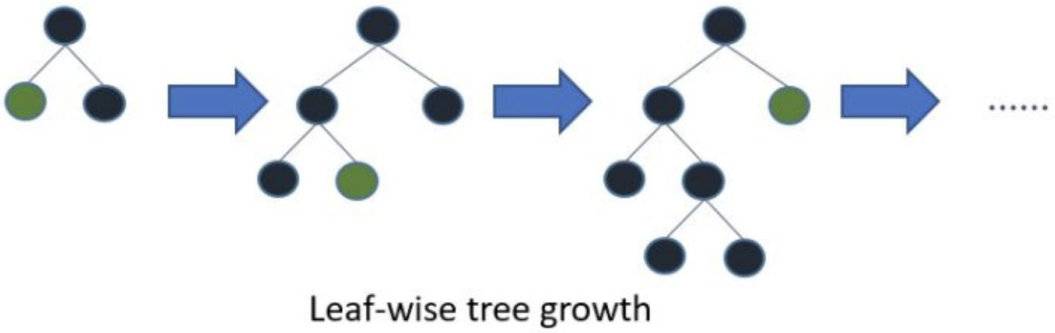

图 6：策略生长树-最大节点生长

生长（如图 6）。当样本数量较小的时候，leaf-wise 可能会造成过拟合，所以 LightGBM 可以利用额外的参数 max_depth 来限制树的深度。level-wise 过一次数据可以同时分裂同一层的叶子，容易进行多线程优化，不容易过拟合。但实际上 level-wise 是一种低效的算法，因为它不加区分的对待同一层的叶子，带来了很多没必要的开销。因为实际上很多叶子的分裂增益较低，没必要进行搜索和分裂。

高维数据一般是稀疏的，并且在稀疏特征空间中，许多特征都是互斥的，也就是它们几乎不同时取非 0 值，LightGBM 可以安全的把这些互斥特征绑到一起形成一个特征，然后基于这些特征束构建直方图，从而达到加速效果。本题数据的特征数量较少，仅有 12 列，而且经过了一些清洗，因此在这个问题上，并不会给我们带来太多的困扰。

2.LightGBM 特征选择原理

使用 python sklearn 工具包，lightgbm 类自带 feature_importance()成员函数。选择的依据包含两种方法：特征在所有决策树中被用来分割的总次数；特征在所有决策树中被用来分割后带来的增益总和。我们使用默认的第一种方法。

LightGBM 做特征选择有如下优势：① 可解释性强； ②对比其他传统机器学习方法（svm，logistics 等），更容易处理离散数据，且不需要归一化，不需要使用 one-hot 编码拆分特征；③对比其他最流行的集成学习方法 xgboost，速度更快，效果相当，比 tensorflow 封装的集成模型效果好很多。

3. 特征选择结果

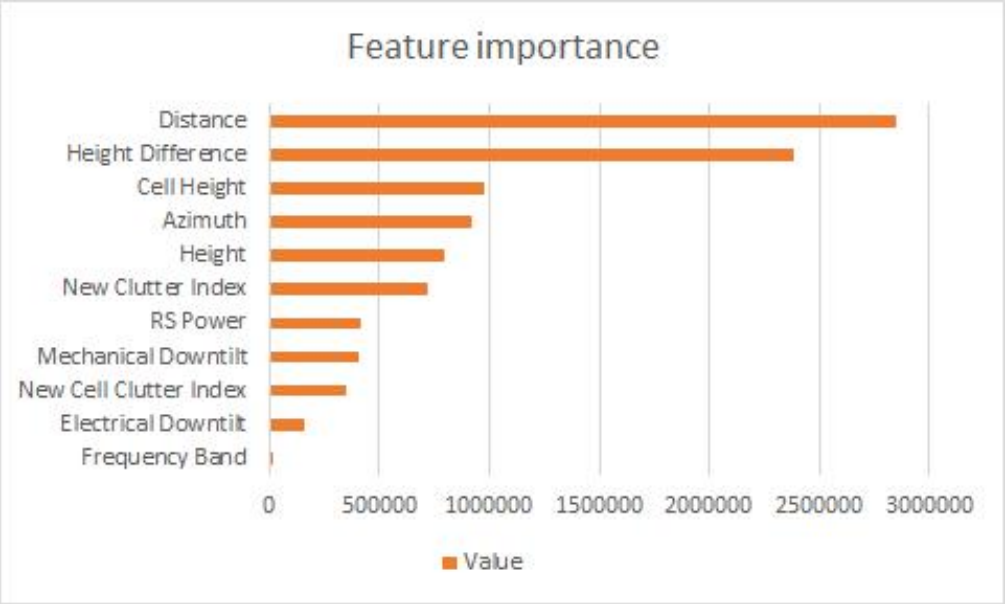

图 7：LightGBM 特征选择结果

根据图 7，我们可以得出，LightGBM 特征选择的结果最重要的两项特征为 Distance 和 Height Difference，两者的重要程度远大于其他特征，其他特征的重要性相对要小一些。

同时我们还发现，Frequency Band，即小区发射机中心频率，这个特征的重要程度极低。在数据分析时，我们也发现该特征在所有样本中，总共只有三种取值，2585 出现的次数超过了一千万次。因此，在后续的模型训练和推理之前，我们把这一特征删除。

5.2.2 相关性分析

1. 相关性原理

相关性，是指两个变量的关联程度。一般地，从散点图上可以观察到两个变量有以下三种关系之一：两变量正相关、负相关、不相关。如果一个变量高的值对应于另一个变量高的值，相似地，低的值对应低的值，那么这两个变量正相关；反之，如果一个变量高的值对应于另一个变量低的值，那么这两个变量负相关。如果两个变量间没有关系，即一个变量的变化对另一变量没有明显影响，那么这两个变量不相关。日常中使用的相关性有以下几种：

表 6：相关系数的分类比较

类型	含义

Pearson 相关系数	反映两个变量线性相关程度的统计量
Kendall 秩相关系数	N 个同类的统计对象按特定属性排序，其他属性通常是乱序
Spearman 相关系数	等级相关，描述两个变量之间的关联程度与方向的统计量

2.　相关性分析

这里我们通过对数据的分析，选择 Pearson 相关系数计算 RSRP 与其他特征的相关性，其计算方法为：

$$r = \frac{1}{n-1}\sum_{i=1}^{n}\left(\frac{X_i - \bar{X}}{s_X}\right)\left(\frac{Y_i - \bar{Y}}{s_Y}\right)$$

其中，$\overline{X}$、$\overline{Y}$ 分别为变量均值，S_X、S_Y 分别为变量的的标准差，n-1 为数据的自由度。

基于提供的约 4000 条各小区数据集，数据整理和清洗后，我们共提取出 12 列特征数据。其中，最后一列为标签数据 RSRP，即栅格(X, Y)的平均信号接收功率。

我们分别计算出 RSRP 与其他 11 列特征数据的相关性，得出相关系数的大小，并按相关系数的绝对值降序排列，如表 7 所示：

表 7：特征名称及其与目标的相关性

排序	特征名称	含义	Pearson 相关系数
1	Distance	栅格与基站所处栅格的距离	-0.186717732
2	Height Difference	基站所在栅格实际高度与用户所在栅格实际高度的差值	0.165491934
3	New Clutter Index	新排序的用户栅格的地貌特征	-0.027255504
4	New Cell Clutter Index	新排序的基站栅格的地貌特征	-0.016689726
5	RS Power	小区发射机发射功率	-0.01445243
6	Electrical Downtilt	小区发射机垂直电下倾角	-0.006652122
7	Frequency Band	小区发射机中心频率	-0.006302005
8	Mechanical Downtilt	小区发射机垂直机械下倾角	0.00492064
9	Cell Height	小区站点所在栅格的海拔高度	0.004886513
10	Height	小区发射机相对地面的高度	-0.001388338
11	Azimuth	小区发射机水平方向角	0.00109509

由表 8 可以看出：11 个变量相关系数绝对值最大为 Distance 变量，绝对值为 0.19，其次为 Height Difference 变量，绝对值为 0.17，其他变量相关系数绝对值均在 0.03 以下。结果与上述 LightGBM 特征选择的结果基本吻合。

5.2.3 方差分析

对于已经处理好特征的 4000 个小区样本，我们对 12 列特征数据分别计算各自的方差，

入已按照相关性绝对值降序对特征排列的表 9 中。

表 8：特征数据的方差

排序	特征名称	含义	方差
1	Distance	栅格与基站所处栅格的距离	1186355.84
2	Height Difference	基站所在栅格实际高度 - 用户所在栅格实际高度	34838.45
3	New Clutter Index	新排序的用户栅格的地貌特征	6.87
4	New Cell Clutter Index	新排序的基站栅格的地貌特征	7.91
5	RS Power	小区发射机发射功率	6.32
6	Electrical Downtilt	小区发射机垂直电下倾角	5.73
7	Frequency Band	小区发射机中心频率	29.57
8	Mechanical Downtilt	小区发射机垂直机械下倾角	5.99
9	Cell Height	小区站点所在栅格的海拔高度	361.57
10	Height	小区发射机相对地面的高度	92.04
11	Azimuth	小区发射机水平方向角	10667.97
12	RSRP	平均信号接受功率	114.68

由表 8 可以看出，方差最大的两个特征变量依然是栅格与基站所处栅格的距离 Distance 以及基站所在栅格实际高度与用户所在栅格实际高度的差值 Height Difference，其所对应的相关系数绝对值也是最大的，其余特征变量的方差相对较小。由图 7-9 可得出，LightGBM 特征选择、相关性分析、方差分析，三者的结果是一致的。

经过以上分析可知，我们设计并选择的特征变量与问题目标有较强的相关性，适合作为表征目标问题的参数，并以此为基础，进行问题三的分析与建模。

六、问题三的分析与建模

6.1 问题三的分析

前面两个问题，我们分别解决了特征提取和特征选择的问题，完成了整个特征工程任务。现在，我们需要建立机器学习模型，来对无线传播模型中不同地理位置的 RSRP 值进行预测。这是一个监督学习回归问题，训练集里 RSRP 值是已知的，可以作为监督信息。同时，RSRP 是一个连续的变量，因此这是一个回归问题。

6.2 问题三的建模

6.2.1 模型介绍

1. 多层感知机

神经网络是基于感知机的扩展，而 DNN 可以理解为有很多隐藏层的神经网络。多层神经网络和深度神经网络 DNN 其实也是指的一个东西，DNN 有时也叫做多层感知机（Multi-Layer perceptron,MLP）。从 DNN 按不同层的位置划分 DNN 内部的神经网络层可以分为三类，输入层，隐藏层和输出层,如下图示例，一般来说第一层是输入层，最后一层是输出层，而中间的层数都是隐藏层。

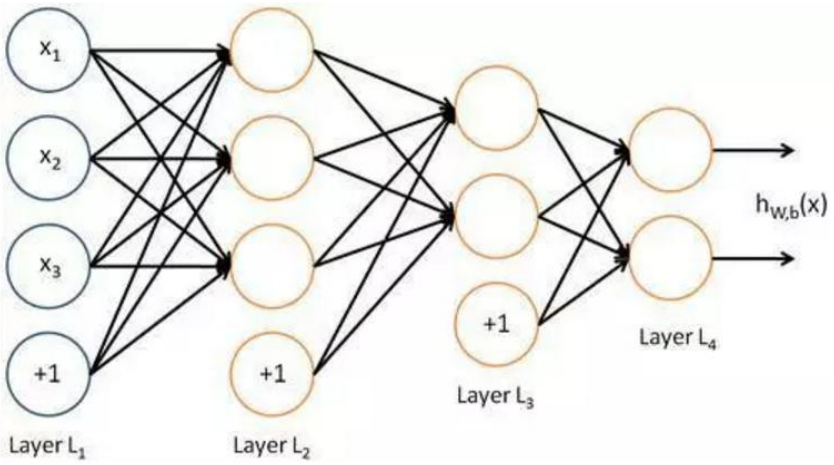

图 8：多层感应机示意图

MLP 的每一层通常都是全连接层，全连接层的本质就是输入数据 X 和权值矩阵 W 做乘法，得到输出数据 Y。DNN 中除了全连接层，还会有池化层，激活层，等等。池化层通常用于压缩数据和参数的量，减小过拟合。除此之外，池化操作具有尺度不变性。数据压缩时去掉的信息只是一些无关紧要的信息，而留下的信息则是具有尺度不变性的特征， 是最能表达图像的特征。常见的池化操作包括均值池化（average-pooling）和最大值池化（max-pooling），均值池化就是把一个窗口中的数据用它们的均值来表示，最大值池化就是把一个窗口中的数据用它们的最大值来表示。下面左图就是以图像数据为例的一个 2x2 池化操作的示意图，右图是一个最大值池化的例子。

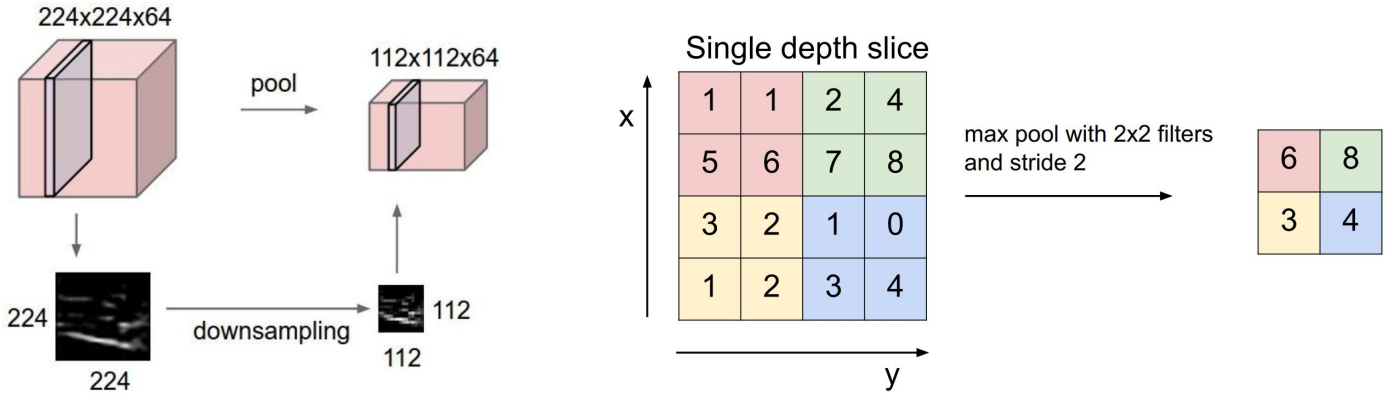

图 9：池化操作示意图

激活层的作用是为神经网络提供非线性变换。常见的激活函数包括 sigmoid 函数，relu 函数，tanh 函数，等等。

图 10：sigmoid 和 relu 集合函数

sigmoid 函数会把特征映射到 0～1 之间，可以把输出解释为概率值，在非中心区域，容易梯度消失。tanh 函数把特征映射到-1～1 之间，能够使特征中心化，但仍然无法避免类似 sigmoid 的梯度消失问题。relu 函数在 x>0 范围内梯度总是 1，而且相比其他的激活函数，计算很高效，也最为符合生物的神经学特质。relu 还有一些变种，如 leaky relu，Exponetial Linear Unit 等等，但是在深度学习中，有很多理论，包括常用的参数初始化方法，都是依据 relu 作为激活函数来推导的。

2. 卷积神经网络

多层前馈神经网络扩展性不好，原因主要有两点，一是神经元是全连接的，二是同层的神经元是相互独立的。这样导致随着隐含层层数或者隐含层的神经元个数的增加，需要建立的链接是指数级的增长，其计算量是极大的挑战。卷积神经网络(Convolutional Neural Network,CNN)是一种以卷积层为核心的前馈神经网络， 卷积层的人工神经元可以响应一部分覆盖范围内的周围单元， 对于大型图像处理有出色表现，在图像/视频的分类，检测，分割，以及去噪、内容生成等诸多领域中，都占据了最主流的地位。近年来，自然语言处理、语音处理、知识图谱等领域也开始广泛使用 CNN。

卷积的基本操作为：卷积核按照固定大小的窗口，在输入数据上滑动，卷积核与每个窗口中的数据做矩阵乘法。相比之下，DNN 中的全连接层，权值矩阵是和全数据进行矩阵乘法。卷积神经网络中的卷积层有如下两个特点：（1）局部连通性，即每一次卷积操作都是针对数据的局部特征来进行的；（2）参数共享性，即同一组卷积核以滑动窗口的方式，遍历输入特征的各个位置，不同位置的卷积操作，是共享同一组卷积核的。 在卷积神经网络中，卷积层通常也配合着池化层、激活层一起使用。下图就是一个 32x32x3 的特征经过卷积操作，变为 28x28 的特征的示意图。

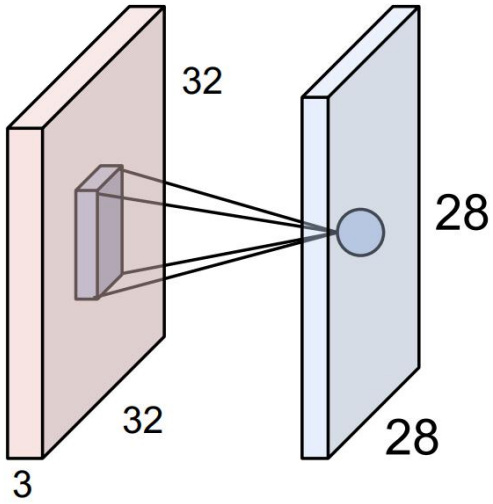

图 11：卷积操作特征的示意图

在卷积神经网络 CNN 中，决定某一层输出结果中一个元素所对应的输入层的区域大小， 被称作感受野。举个例子，在 maxpooling 层中，如果它的 kenerl size 是 2x2， 输出结果中的每一个元素都是其对应输入的 2x2 的区域中的最大值， 所以这一层的感受野大小就是 2。在 3x2 卷积层中，如果上一层的感受野是 m，那么这一层的感受野就是 m + 2。

DNN 和 CNN 神经网络的数据输入层是类似的。我们在运用神经网络时，不需要像使用传统机器学习模型或者集成学习模型一样做太多的人工特征选择，但是需要遵循一些特定的数据预处理原则。常用的预处理方法有：（1）去均值： 把输入数据各个维度都中心化为 0，如下图所示，其目的就是把样本的中心拉回到坐标系原点上；（2）归一化： 幅度归一化到同样的范围， 如下所示 即减少各维度数据取值范围的差异而带来的干扰， 比如，我们有两个维度的特征 A 和 B， A 范围是 0 到 10， 而 B 范围是 0 到 10000， 如果直接使用这两个特征是有问题的， 好的做法就是归一化，即 A 和 B 的数据都变为 0 到 1 的范围。去均值与归一化效果如下图：

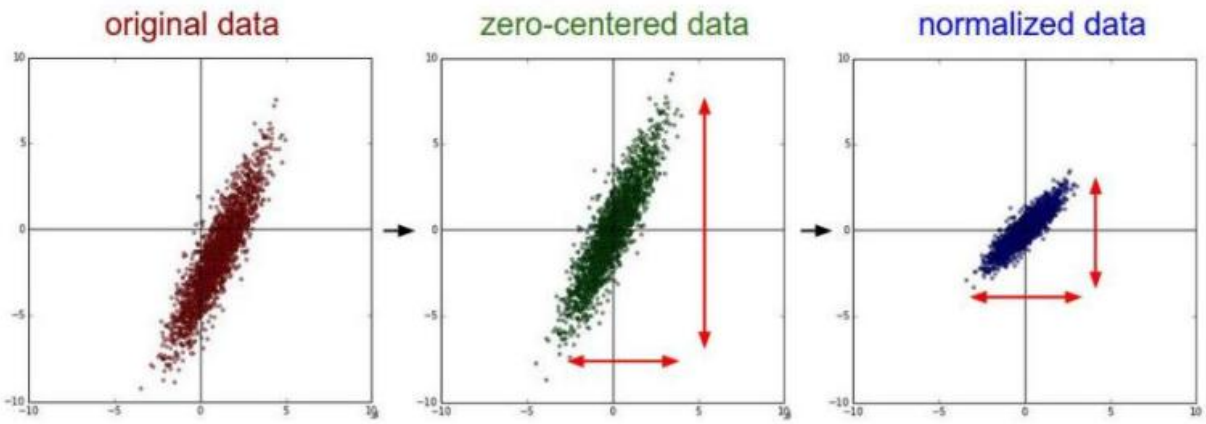

图 12：去均值与归一化效果图

3.LightGBM

在问题二的求解中，我们已经介绍了 LightGBM 的基本原理。本题作为一个有监督的回归问题，同样很适合 LightGBM 建模求解。LightGBM 一般集成的就是分类回归树，首先，它具有树模型可解释性强的优点。问题二中，我们使用 LightGBM 做特征选择，充分解释了哪些特征的是对回归目标最具有影响力的。问题二和问题三是相通的：在问题二中，我们实际上也是训练了一个 LightGBM 模型，通过计算特征在所有决策树中被用来分割的总次数，很直接地就可以对模型进行解释；而在问题三中，我们使用的是问题二和问题一的特征工程提取出来的特征。其次，LightGBM 特别适合结构化数据，且可以完美兼容离散数据、类别特征、缺失数据的处理，而这一点是 CNN 等深度神经网络做不到的。实际上，在这类结构化数据的回归问题中，最常用的方法就是基于树模型的集成学习方法，包括随机森林（random forest），XGBoost 等等，而这其中，LightGBM 又是最高效的方法。根据我们的经验，LightGBM 往往能给这类问题提供一个保证下限的 baseline。因此，我们希望用 LightGBM 模型跟神经网络模型做一个对比。

6.2.2 模型结构设计

1. 多层感知机

我们首先尝试了 2~4 层的 DNN 模型，所有的层均为全连接层。每个全连接层后面加上 relu 激活函数。全连接层的优点在于，每一层都融合了所有的特征，但它的参数量非常大，会把训练速度拖慢，并且很容易过拟合。我们训练的全连接层采用均方误差（mse）作为损失函数，在输入网络之前，把数据归一化成均值为 0 方差为 1。这一模型的 rmse 为 10.0。

2. 卷积神经网络（CNN）

考虑到卷积神经网络可以有效地提取局部相关性，同时它可以通过不断地加深网络深度，扩大感受野，使得输出的特征包含足够大范围的全局信息。此外，CNN 的参数利用率非常高，因为数据或特征的各个位置共享卷积参数。我们设计了三种 CNN 结构，如下表所示。需要说明的是，conv 表示卷积层（若不作特别说明，每个卷积层后面都跟随一个批标准化层 batchnorm，和一个激活层 relu），卷积参数为(k, m, c_1, c_2)，分别表示卷积核

尺寸(k, m)，输入通道数 c_1，输出通道数 c_2，每一层的特征为一个具有尺寸(n, h, w, c)的四位数组，其中 n 为样本个数，原始数据的尺寸被变换到(h, w, c) = (特征数量, 1, 1)。fc 表示全连接层，括号内的数表示该层参数矩阵的维度，feat_dim 表示模型输入的特征维度。

表 9：模型结构表

Model1	Model2	Model3
Layer1: conv (3, 1, 1, 32)	Layer1: conv (5, 1, 1, 64)	Layer1: conv (5, 1, 1, 64)
Layer2: conv (1, 1, 32, 32)	Layer2: block (3, 64)	Layer2: conv (1, 1, 64, 128)
Layer3: conv (1, 1, 32, 32)	Layer3: conv (1, 1, 64, 128)	Layer3: conv (3, 1, 128, 128)
Layer4: conv (3, 1, 32, 64)	Layer4: block (3, 128)	Layer4: inception block (128)
Layer5: fc (feat_dim x 64, 1)	Layer5: conv (1, 1, 128, 256)	Layer5: inception block (256)
	Layer6: block (3, 256)	Layer6: inception block (256)
	Layer7: fc (feat_dim x 256, 1)	Layer7: fc (feat_dim x 256, 1)

第一种结构我们称之为 naive-network-in-network，基于 Network In Network （NIN）设计。我们发现仅用两层 3x1 的卷积层，就可以超越 4 层全连接层的效果，而 NIN 在不同的卷积层之间加入小的子网络（比如 1x1 卷积），它是一种加深网络深度同时压缩参数数量的有效方式，是深度学习流行早期的一个经典方法。我们设计的网络可以看作是极简的 NIN 形式的网络，所以称之为 naive-network-in-network。

第二种结构我们称之为 progressive-residual-network，基于 ResNet 设计。ResNet 的基本单元如下图所示。如果我们把一般的 CNN 看作是学习 x -> H(x) 这样的映射，那么 ResNet 所使用的残差学习可以表示为 x -> (H(x) - x) + x = H(x) = F(x) + x。这样的网络结构可以有效的避免反向传播过程中的梯度消失，增加收敛速度，因为通过如图所示的跳跃连接，上一个残差块的信息可以没有阻碍的流入到下一个残差块。ResNet 的出现，在当时大大增加了 CNN 的深度，通过堆叠多个 residual

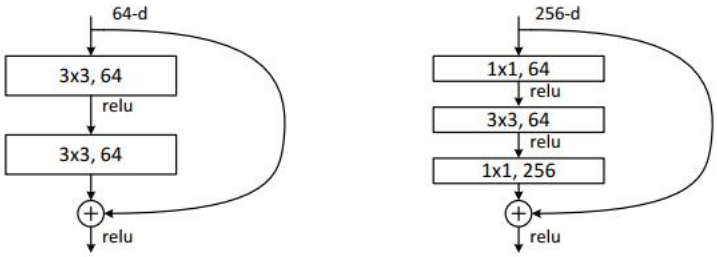

Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.

图 13：ResNet 残差结构

block 和 bottleneck，使得 50 层（ResNet-50），101 层（ResNet-101），甚至 200 层以上（ResNet-200）的深度网络的训练成为可能。除此之外，我们可以把 ResNet 看作是浅网络和深网络的融合，一定程度上起到了模型融合的效果。

表格中 block 的参数(b_1， b_2)表示两个卷积层的参数：conv（1，1， b_2， b_2），conv（b_1，1， b_2，b_2）。我们设计了 3 个 block 结构，加上最开始的 5x1 卷积，这样网络的感受野刚好是 11，刚刚超过我们使用的特征数量，这样网络的感受野能完全覆盖数据的所有维度。我们的网络宽度（每一层的通道数）随着深度的增加而增加，使用 1x1 卷积来进行升维，减小计算量，高效地提取出逐层抽象的高维特征，因此这个模型被称为 progressive-residual-network。

第三种结构称之为 multi-view-wider-network。CNN 的几个经典结构中，除了 ResNet 之外，最有代表性的应当属于 Inception（GoogleNet）和 VGG。在图像处理问题中，VGG 是直接堆叠 1x1 和 3x3 卷积，如果说 ResNet 的思想是如何把网络加深，那么 GoogleNet 的思想可以看成是如何把网络加宽。如果我们希望扩大感受野的同时，还要尽量减少参数数量，那么应该怎么做？GoogleNet 使用多尺度卷积核为同一个输入层提供了不同大小的感受野，然后把多个不同感受野的支路融合起来。下图为 GoogleNet 的其中一层的结构。

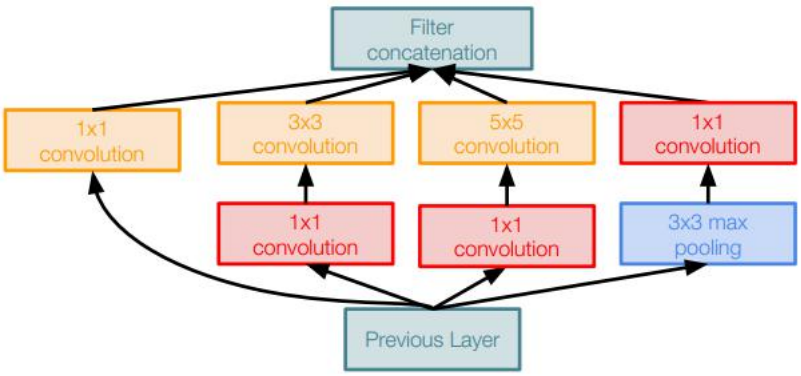

图 14：GoogleNet 中的 inception 结构

我们仿照 GoogleNet 的 inception 结构，自定义了一种 4 个支路的 inception block，带有一个参数b_3的 inception block（b_3）结构如下表所示

表 10：自定义 inception 结构

Branch1	Branch2	Branch3	Branch4
conv (1, 1, b_3, b_3 / 2)	conv (1, 1, b_3, b_3 / 4)	conv (1, 1, b_3, b_3 / 4)	conv (1, 1, b_3, b_3 / 4)
	conv (3, 1, b_3 / 4, b_3 / 2)	conv (5, 1, b_3 / 4, b_3 / 2)	dilate conv (3, 1, b_3 / 4, b_3 / 2)

其中卷积层的表示方法与本节开头的定义相同，dilate conv 表示膨胀系数为 2 的孔洞卷积，它只需要 3x1 卷积层的参数量，就可以达到 5x1 卷积层的感受野大小，如下图所示。

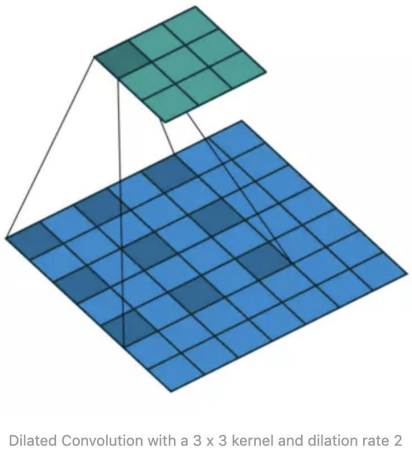

图 15：孔洞卷积示意图

这种结构通过构造 4 个感受野不同的支路，模拟了多视角观测的行为。我们的数据有 10 列，有的列之间相关性很强，有的相邻列之间也许没有那么强的局部特征，那么这种模型就可以帮助我们更好地融合局部特征。

3.LightGBM

我们还尝试了使用 LightGBM 来进行回归。LightGBM 相比于 CNN，优势在于以下几点：（1）可解释性强，通过模型内部的分裂算法，我们可以很自然地得到每个特征被作为分裂点的次数，以及分裂之后信息增益的总和，这样就很容易解释结果对特征的依赖。（2）不需要特征归一化，且更适合处理离散数据，通过对数据的分析，我们知道本题的训练数据，有很多高度离散的特征（它们有可能是具有连续物理意义，但是取值种类很少的特征），比如小区发射机中心频率、小区小区发射机发射功率、小区站点和小区用户所在位置的地物类型，也有一些有一定离散性的特征，比如跟角度相关的特征。这样的数据，未必很适合 CNN 来处理。但是相比于 CNN，LightGBM 也有明显的缺点，比如推理速度太慢，模型太大（可达到 400M 或更大），难以在本题的生产环境部署，等等。因此，这一部分得到的只是在本地运行的结果。

6.2.3 训练结果

我们使用的评价方法有两个：

- **弱覆盖识别率 (PCRR : Poor coverage recognition rate)**

PCRR 综合考虑 Precision（准确率）和 Recall（召回率）的目标，其计算公式如下：

$$\mathrm{PCRR} = 2 * \frac{\mathrm{Precision} * \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}}$$

其中 Precision 可以理解为预测结果为弱覆盖的栅格实际也是弱覆盖的概率，其定义如下：

$$\mathrm{Precision} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

Recall 可以理解为真实结果为弱覆盖的栅格有多少被预测成了弱覆盖的概率，其定义如下：

$$\mathrm{Recall} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

- **均方根误差 (RMSE : Root mean squared error)**

RMSE 是评估预测值和实测值整体偏差的指标，其大小直观表现了仿真准确性。直接计算待评估数据的 RMSE，计算公式如下：

$$\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(P^{(i)} - \widehat{P}^{(i)})^2}$$

其中$P^{(i)}$为参赛队机器学习模型对于第 i 组评审数据集的 RSRP 预测值，$\widehat{P}^{(i)}$为第 i 组评审数据集的 RSRP 实际测量值。

在进行预测的过程中如果可以有效识别**弱覆盖区域**，能够更好地帮助运营商进行精准规划和优化网络从而提升客户体验。因此，弱覆盖识别准确率是作为一项非常有价值的评价指标。在弱覆盖识别率 PCRR 达标的情况下，我们的主要评价依据**均方根误差 RMSE**，其是评估预测值和实测值整体偏差的指标，其大小直观表现了仿真准确性。

1 神经网络训练结果

我们这首先使用上节提到的多层感知机模型，RMSE = 10.0。

现在我们来比较一下上述三种方法，从 naive-network-in-network 到 progressive-residual-network，再到 multi-view-wider-network，复杂程度是递增的。在本地的 python3.6+tensorflow1.8+cuda10.1 的环境下，使用 1 个 1080ti gpu。我们所有模型都使用如下参数：学习率(learning rate)设置为 0.0001，每 10 轮下降到之前的 10%，训练 30 轮；正则化方法采用权值衰减策略，参数为 0.0005，以防止过拟合；批大小(batch size)设置为 2048；优化器使用 Adam，自适应的优化器通常能够加速收敛，并且一定程度上减少参数初始化和调整学习率的麻烦，由此带来的参数量成倍增加也是可以接受的；批标准化(batch normalization)层在每一个卷积层后面使用；激活函数全部使用 relu。需要说明的是，受到平台部署的限制，我们暂时没有把带有批标准化层的模型部署到云端，造成了实际提交结果比本地运行的结果略低。与前面的方法一样，我们把训练数据集的 20%划分成验证集，在验证集上，本地运行结果如下表所示 （Model1 到 Model3 的顺序与前文描述顺序一致）：

表 11：三种 cnn 模型在验证集上的结果

模型	Model1	Model2	Model3
RMSE	9.110	7.805	7.799
PCRR	0.139	0.398	0.392

我们同时给出了 RMSE 和 PCRR 两个结果，但我们的损失函数就是 RMSE，因此主要参考 RMSE 的结果。从上表我们可以分析出如下结论：（1）相比于全连接网络，Model1 基于 NIN 仅用了 4 层卷积，RMSE 就提升了接近 1 个点。同时，Model1 模型的参数量仅有 8928，而 4 层全连接层，每层输出维度跟 Model1 相同的情况下，模型大小达到了 413440；（2）通过加深模型深度和宽度，回归效果提升非常明显，这说明我们的数据内部含有很多隐藏的深层次的特征，另一方面，模型的规模并不能无限制地增加下去，以 Model2 为例，在实验中发现，如果我们把每一层的通道数减半，训练迭代次数增加一倍，则 RMSE=7.984，如果继续增加通道数或者深度，RMSE 减小的幅度会非常小，类似地，Model3 和 Model2 相比，参数量增加了接近一倍，但是效果相差不大；（4）CNN 的结果与 LightGBM 相比还是有一定的差距，这也说明了在更大的参数空间下，搜索最优解的难度也大大增加。

2. LightGBM 模型训练结果

LightGBM 需要调整的参数不多。为了增加模型的推理能力，我们把叶子结点数量上限设为 15000， 学习率设置为 0.15，并且不进行列采样；为了防止过拟合，我们使用了 0.8 倍的行采样。RMSE 值为 6.697，PCRR 值为 0.531。

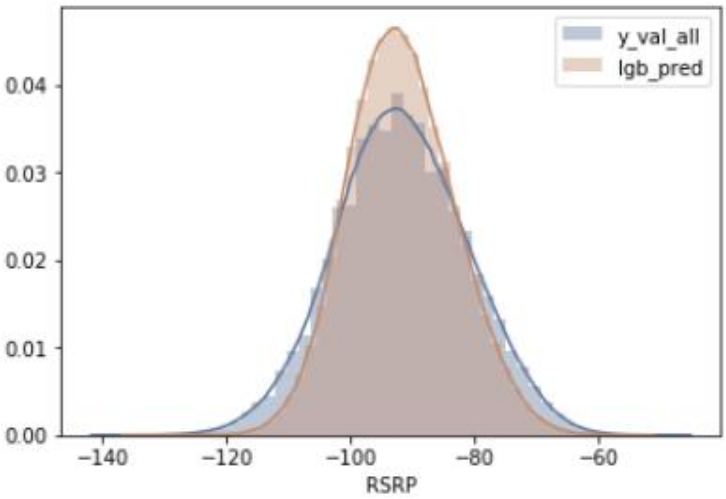

图 13：RSRP 真实值和预测值的概率分布比较

为了进一步考察 LightGBM 模型的性能，我们对模型预测的 RSRP 分布和标签分布进行了对比。记验证集的 RSRP 为 y，输出的 RSRP 预测值为 y_pred。经过计算，y 的均值为-91.807，y_pred 的均值为-91.783；y 的标准差为 10.706，y_pred 的标准差为 8.481；y 的峰度为 2.949，y_pred 的峰度为 2.924。可以看出，从一阶、二阶，甚至更高阶的统计量来看，二者的分布确实非常接近。更进一步，我们计算了 y 与 y_pred 的 KL 散度。KL 散度衡量的是，以 y_pred 近似 y 的分布时，损失的信息量，该值仅为 0.0027。这说明，用 y_pred 的分布来作为 y 的分布的近似，是非常合适的，几乎不会有信息损失。

七、模型评价

神经网络是当前机器学习最受关注的方向之一，也是解决大部分实际问题的利器。多层感知机 MLP，或者说全连接层堆叠的 DNN，参数量大，参数利用率低下，表征能力有限。基于 CNN 的模型是我们线上部署的模型，除了易于部署之外，它推理速度快，不太依赖特征提取和特征选择，而且有时候具有很大的潜力。CNN 的模型表征能力相比于 MLP 要强大得多，被广泛应用于视觉计算、自然语言处理、语音处理、知识图谱、广告推荐等领域。但是 CNN，特别是深度 CNN，具有庞大的参数搜索空间，加之神经网络本来就不是一个完美的凸优化问题，这就给模型收敛到全局最优解造成了非常大的困难。CNN 或者 DNN 要想顺利工作，离不开合适的权值初始化方法、适当的参数量、合理网络结构、精心调节的大量参数（学习率、正则化等），而本题中的无线传播模型很少有基于 CNN 的研究应用，算法开发者缺乏先验知识。上述种种问题，给我们短时间内训练一个千万数据级别的无线传播场景模型带来了很大的挑战。

在本题中，我们在线上部署的 CNN 模型表现优良，在超过 2000 人注册参赛的情况下，在本文撰写完成的时间节点，能够排名在第 35 位。我们提出的的基于 ResNet 和 GoogleNet 的 CNN 模型，在线下的验证场景中，均方误差 RMSE 能达到 7.8 左右，弱覆盖率 PCRR 接近 0.4。这充分表明，我们的模型具有较强的学习表征能力。

另一个解决此类数据分析和回归问题的常用方法就是基于树的集成学习模型。这类方法以随机森林、GBDT、XGBoost、LightGBM 等为代表，具有和树模型一样的可解释性，又把树模型的威力通过集成的方式扩大到了极致。XGBoost 一度是近几年来最受欢迎的集成学习模型，LightGBM 模型是对 XGBoost 模型的进一步优化，保持 XGBoost 精度的同时，提高了运算效率。除了可解释性强，LightGBM 还有诸多优点。对于结构化数据，尤其是包含离散特征的数据的处理，是 LightGBM 的强项。同时，LightGBM 可以直接学习带有缺失值的数据，不需要额外处理。在短时间内要想获得性能有保证的 baseline，LightGBM 几乎是一个最佳选择。在线下实验中，我们使用 LightGBM 模型对手工划分的验证集进行了推理，预测出来的 RSRP 值与真实的 RSRP 值相比，不仅均方误差 RMSE 达到 6.6 的较小数值，弱覆盖率 PCRR 也超过 0.53，在题目要求的评价指标下达到了比较有说服力的效果，更为重要的是，真实 RSRP 值和模型预测值的分布几乎一致，具有数值上极为相近的统计量，和极小的 KL-散度。LightGBM 也有一些不足之处。首先，要想做好一个基于 LightGBM 的回归任务，特征工程的质量尤为重要。在我们特征工程的这部分工作中，显然还有可以提升的空间，要想使 LightGBM 的性能最大化，就必须把特征工程做得更好。其次，在如此大规模的数据上使用集成学习模型，必然会造成推理速度慢，模型太大的问题。我们训练出来的 LightGBM 大小为 462M，是 CNN 模型的数十倍甚至更多。再者，因为本次生产环境的限制，我们暂时还无法把 LightGBM 模型部署到云端。

八、参考文献

[1] 江巧捷,林衡华,岳胜. 5G 传播模型分析[J]. 移动通信, 2018,42(10): 19-23.
[2] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich "Going Deeper with Convolutions CVPR 2015, pp.1-9
[3] 陈思伃. 无线网络优化中传播模型校正的研究[D]. 北京邮电大学，2011: 13-19
[4] 成澜.无线信道仿真与建模:[硕士学位论文].武汉：华中科技大学图书馆，2008
[5] SIGKDD Conference on Knowledge Discovery and Data Mining, 2016
[6] Wang.S.S, Wylie-Green.M.P. Geolocation Propagation Modeling for Cellular-Based Mobile Positioning. In:Vehicular Technology Conference, 2004. VTC2004-Fall. 2004 IEEE 60th. Irving:M Malkawi, 2001.5155~5159
[7] 党俊肖，武丽梅，马金辉. 基于 GIS 的移动通信信号传播预测及可视化仿真研究[J]. 无线通信技术，1003-8329(2015) 04-0001-06
[8] Zafra A，Gibaja E L，Ventura S. Multiple instance learning with multiple objective genetic programming for web mining [J]. Applied Soft Computing，2011，11: 93 -102.
[9] 苏华鸿，孙孺石，等. 蜂窝移动通信射频工程[M]. 第二版.北京: 人民邮电出版社，2007.
[10] https://www.kdnuggets.com/2018/01/gradient-boosting-tensorflow-vs-xgboost.html
[11] Tianqi Chen and Carlos Guestrin. XGBoost: A Scalable Tree Boosting System. In 22nd

[12]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.
[13]韦惠民．蜂窝移动通信技术[M]．第一版．西安: 西安电子科技大学出版社，2002
Min Lin, Qiang Chen, Shuicheng Yan Network In Network ICLR 2014
[14]Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun. Deep Residual Learning for Image Recognition. CVPR 2016

附录：

1.服务上线前的数据预处理、特征提取

```
import math
import numpy as np
from model_service.tfserving_model_service import TfServingBaseService
import pandas as pd


class mnist_service(TfServingBaseService):

  def _preprocess(self, data):
    preprocessed_data = {}
    filesDatas = []
    for k, v in data.items():
      for file_name, file_content in v.items():
        pb_data = pd.read_csv(file_content) # 17 cols, no RSRP

        # 特征提取
        pb_data = self._combine_features(pb_data)

        input_data = np.array(pb_data.get_values()[:, :], dtype=np.float32)
        # 删除通过特征选择找到的多余特征
        input_data = np.concatenate((input_data[:, :5], input_data[:, 6:]), axis=1)
        # 使用训练集的特征，减均值除方差归一化
        input_data -= np.array([615.29426412, 23.22653556, 172.58714154, 5.1276589, 3.48686738, 1
1.26828345, 530.73660944, -66.95205253, 3.67464835, 3.83187333])
        input_data /= np.array([1090.48577941, 9.59413055, 103.28729118, 2.3946823, 2.44741765,
2.51411823, 19.01568261, 186.9536307, 2.62639002, 2.81253494])
        print(file_name, input_data.shape)
        filesDatas.append(input_data)

    filesDatas = np.array(filesDatas, dtype=np.float32).reshape((-1, 10))
```

```
    preprocessed_data['myInput'] = filesDatas
    print("preprocessed_data[\'myInput\'].shape = ", preprocessed_data['myInput'].shape)

    return preprocessed_data


def _postprocess(self, data):
    infer_output = {"RSRP": []}
    for output_name, results in data.items():
        print(output_name, np.array(results).shape)
        infer_output["RSRP"] = results
    return infer_output


def _combine_features(self, train):
    ## (X1, Y1), (X2, Y2) -> dist
    xx = train['Cell X'] - train['X']
    yy = train['Cell Y'] - train['Y']
    dist = np.sqrt(xx * xx + yy * yy)

    ## compute height different from cell building to custom & real height of cell building
    diff = (train['Cell Building Height'] + train['Cell Altitude'] + train['Height']) \
           - (train['Building Height'] + train['Altitude'])
    arcs = (train['Electrical Downtilt'] + train['Mechanical Downtilt']) / 180 * math.pi
    for i, arc in enumerate(arcs):
        diff[i] -= dist[i] * math.tan(arc)

    real_height = train['Cell Building Height'] + train['Cell Altitude'] + train['Height']

    col_name = train.columns.tolist()
    col_name.insert(10, 'Height Difference')
    col_name.insert(9, 'Cell Height')
    col_name.insert(1, 'Distance')

    train1 = train.reindex(columns=col_name)
    train1 = train1.drop(columns=['Cell Building Height', 'Building Height',
                                  'Cell Altitude', 'Altitude',
                                  'Cell X', 'Cell Y', 'X', 'Y'])

    train1['Height Difference'] = diff
    train1['Cell Height'] = real_height
    train1['Distance'] = dist

    train1 = train1.drop(columns=['Cell Index'])
```

```
    col_name = train1.columns.tolist()
    col_name.insert(11, 'New Cell Clutter Index')
    col_name.insert(11, 'New Clutter Index')
    train1 = train1.reindex(columns=col_name)
    clutter_index_dict = {2: 1, 5: 2, 6: 3, 7: 4, 8: 5, 10: 6, 11: 7, 12: 8, 13: 9, \
                14: 10, 15: 11, 16: 12, 17: 13, 18: 14}

    new_clutter_index = np.zeros((train1.shape[0], ))
    new_cell_clutter_index = np.zeros((train1.shape[0], ))

    new_clutter_index= train1['Clutter Index'].map(clutter_index_dict)
    new_cell_clutter_index = train1['Cell Clutter Index'].map(clutter_index_dict)

    train1['New Clutter Index'] = new_clutter_index
    train1['New Cell Clutter Index'] = new_cell_clutter_index

    train1 = train1.drop(columns=['Clutter Index', 'Cell Clutter Index'])

    return train1
```

2. 异常值检测

```
import math
import numpy as np
import os
import pandas as pd

from sklearn.decomposition import PCA

import pickle
with open('./X_train_all.pkl', 'rb') as f:
  train_data = pickle.load(f)

# norm
train_mean = np.mean(train_data, axis=0)
train_var = np.std(train_data, axis=0)
train_data -= train_mean
train_data /= train_var

# pca
pca = PCA()
pca.fit(train_data)
transformed_data = pca.transform(train_data)
y = transformed_data # y = data x P
```

```
lambdas = pca.singular_values_

M = ((y*y)/(lambdas + 1e-8))
#q = 10
#major_components = M[:, range(q)]
major_components = np.sum(M, axis=1)
components = pd.DataFrame({'major_components': major_components})
thre = components.quantile(0.99)['major_components']
outliers_idx_pca = np.where(major_components > thre)[0]
```

3. 特征选择

```
import math
import numpy as np
import os
import pandas as pd
import sys
sys.path.append('./LightGBM/python-package/lightgbm')

import lightgbm as lgb

from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, LabelEncoder

path1 = './train_set1/'
files= os.listdir(path1)

X_train_all = np.zeros((0, 11))
X_val_all = np.zeros((0, 11))
y_train_all = np.zeros((0, ))
y_val_all = np.zeros((0, ))

import pickle
with open('./X_train_all.pkl', 'rb') as f:
    X_train_all = pickle.load(f)
with open('./X_val_all.pkl', 'rb') as f:
    X_val_all = pickle.load(f)
with open('./y_train_all.pkl', 'rb') as f:
    y_train_all = pickle.load(f)
with open('./y_val_all.pkl', 'rb') as f:
    y_val_all = pickle.load(f)
```

```
X_train_all = np.concatenate((X_train_all[:, :5], X_train_all[:, 6:]), axis=1)
X_val_all = np.concatenate((X_val_all[:, :5], X_val_all[:, 6:]), axis=1)

############## lightgbm baseline ##############

lgb_train = lgb.Dataset(X_train_all, label=y_train_all)
lgb_val= lgb.Dataset(X_val_all, label=y_val_all)

param = {
'num_leaves' : 15000,
#'max_depth' : 28,
'learning_rate' : 0.15,
'num_threads' : 4,
'objective' : 'regression_l2',
#'min_data_in_leaf' : 100,
'bagging_fraction' : 0.75,
'bagging_freq' : 1,
'metric' : 'l2_root' # RMSE
}

gbm = lgb.train(param,
            lgb_train,
            num_boost_round=1000,
            valid_sets=lgb_val,
            early_stopping_rounds=50,
            verbose_eval=50)
print(gbm.best_score)

path1 = './train_set1/'
files= os.listdir(path1)
train1 = pd.read_csv(path1 + files[0], encoding='gbk')
feature_imp = pd.DataFrame(sorted(zip(gbm.feature_importance(), train1.columns)), columns=['Value','
Feature'])
print('Feature importances:', feature_imp)

gbm.save_model('./model_gbm.txt')
```

4. 模型训练

```
import argparse
import numpy as np
import os
import tensorflow as tf

#os.environ["CUDA_VISIBLE_DEVICES"] = "2"

# get data
import pickle
with open('/ghome/zhanght/MathModel/X_train_all.pkl', 'rb') as f:
    X_train_all = pickle.load(f)

with open('/ghome/zhanght/MathModel/X_val_all.pkl', 'rb') as f:
    X_val_all = pickle.load(f)

with open('/ghome/zhanght/MathModel/y_train_all.pkl', 'rb') as f:
    y_train_all = pickle.load(f)

with open('/ghome/zhanght/MathModel/y_val_all.pkl', 'rb') as f:
    y_val_all = pickle.load(f)

# normalization
X_train_all = np.concatenate((X_train_all[:, :5], X_train_all[:, 6:]), axis=1)
X_val_all = np.concatenate((X_val_all[:, :5], X_val_all[:, 6:]), axis=1)
data_mean = np.mean(X_train_all, axis=0)
data_var = np.std(X_train_all, axis=0)
X_train_all -= data_mean
X_train_all /= data_var
X_val_all -= data_mean
X_val_all /= data_var

parser = argparse.ArgumentParser(description='NN_Model')
parser.add_argument('--reg', default=5e-4, type=float)
parser.add_argument('--lr', default=2e-4, type=float)
parser.add_argument('--dr', default=1.0, type=float)
parser.add_argument('--ds', default=1000, type=int)
parser.add_argument('--epoch', default=100, type=int)
parser.add_argument('--print_freq', default=500, type=int)
parser.add_argument('--batch_size', default=2048, type=int)
parser.add_argument('--save_path', default="/workspace/MathModel/models/conv7_64_res", type=str)
args = parser.parse_args()
```

```
# train a specific model

tf.reset_default_graph()

# define our input (e.g. the data that changes every batch)
# The first dim is None, and gets sets automatically based on batch size fed in
X = tf.placeholder(tf.float32, [None, 10], name="input_X")
y = tf.placeholder(tf.int64, [None])
#is_training = tf.placeholder(tf.bool, name="is_train")

## define model
def conv7_64_res_model(X):
  X = tf.reshape(X, [-1, 10, 1, 1])
  regularizers = 0

  # layer 1
  Wconv1 = tf.get_variable("Wconv1", shape=[5, 1, 1, 64])
  bconv1 = tf.get_variable("bconv1", shape=[64])
  conv1 = tf.nn.conv2d(X, Wconv1, strides=[1,1,1,1], padding='SAME') + bconv1
  #batch1 = tf.layers.batch_normalization(conv1, training=is_training)
  relu1 = tf.nn.relu(conv1)
  regularizers += tf.nn.l2_loss(Wconv1)

  Wconv2 = tf.get_variable("Wconv2", shape=[1, 1, 64, 64])
  bconv2 = tf.get_variable("bconv2", shape=[64])
  conv2 = tf.nn.conv2d(relu1, Wconv2, strides=[1,1,1,1], padding='SAME') + bconv2
  #batch2 = tf.layers.batch_normalization(conv2, training=is_training)
  relu2 = tf.nn.relu(conv2)
  regularizers += tf.nn.l2_loss(Wconv2)

  Wconv3 = tf.get_variable("Wconv3", shape=[3, 1, 64, 64])
  bconv3 = tf.get_variable("bconv3", shape=[64])
  conv3 = tf.nn.conv2d(relu2, Wconv3, strides=[1,1,1,1], padding='SAME') + bconv3
  #batch3 = tf.layers.batch_normalization(conv3, training=is_training)
  conv3 += relu1
  relu3 = tf.nn.relu(conv3)
  regularizers += tf.nn.l2_loss(Wconv3)

  Wconv3_2 = tf.get_variable("Wconv3_2", shape=[1, 1, 64, 128])
  bconv3_2 = tf.get_variable("bconv3_2", shape=[128])
  conv3_2 = tf.nn.conv2d(relu3, Wconv3_2, strides=[1,1,1,1], padding='SAME') + bconv3_2
  #batch3_2 = tf.layers.batch_normalization(conv3_2, training=is_training)
  relu3_2 = tf.nn.relu(conv3_2)
```

```
regularizers += tf.nn.l2_loss(Wconv3_2)

Wconv4 = tf.get_variable("Wconv4", shape=[1, 1, 128, 128])
bconv4 = tf.get_variable("bconv4", shape=[128])
conv4 = tf.nn.conv2d(relu3_2, Wconv4, strides=[1,1,1,1], padding='SAME') + bconv4
#batch4 = tf.layers.batch_normalization(conv4, training=is_training)
relu4 = tf.nn.relu(conv4)
regularizers += tf.nn.l2_loss(Wconv4)

Wconv5 = tf.get_variable("Wconv5", shape=[3, 1, 128, 128])
bconv5 = tf.get_variable("bconv5", shape=[128])
conv5 = tf.nn.conv2d(relu4, Wconv5, strides=[1,1,1,1], padding='SAME') + bconv5
#batch5 = tf.layers.batch_normalization(conv5, training=is_training)
conv5 += relu3_2
relu5 = tf.nn.relu(conv5)
regularizers += tf.nn.l2_loss(Wconv5)

Wconv5_2 = tf.get_variable("Wconv5_2", shape=[1, 1, 128, 256])
bconv5_2 = tf.get_variable("bconv5_2", shape=[256])
conv5_2 = tf.nn.conv2d(relu5, Wconv5_2, strides=[1,1,1,1], padding='SAME') + bconv5_2
#batch5_2 = tf.layers.batch_normalization(conv5_2, training=is_training)
relu5_2 = tf.nn.relu(conv5_2)
regularizers += tf.nn.l2_loss(Wconv5_2)

Wconv6 = tf.get_variable("Wconv6", shape=[1, 1, 256, 256])
bconv6 = tf.get_variable("bconv6", shape=[256])
conv6 = tf.nn.conv2d(relu5_2, Wconv6, strides=[1,1,1,1], padding='SAME') + bconv6
#batch6 = tf.layers.batch_normalization(conv6, training=is_training)
relu6 = tf.nn.relu(conv6)
regularizers += tf.nn.l2_loss(Wconv6)

Wconv7 = tf.get_variable("Wconv7", shape=[3, 1, 256, 256])
bconv7 = tf.get_variable("bconv7", shape=[256])
conv7 = tf.nn.conv2d(relu6, Wconv7, strides=[1,1,1,1], padding='SAME') + bconv6
#batch7 = tf.layers.batch_normalization(conv7, training=is_training)
conv7 += relu5_2
relu7 = tf.nn.relu(conv7)
regularizers += tf.nn.l2_loss(Wconv7)

# layer5: 1 fc
W1 = tf.get_variable("W1", shape=[10 * 256, 1])
b1 = tf.get_variable("b1", shape=[1])
relu7 = tf.reshape(relu7, [-1, 10 * 256])
y_out = tf.matmul(relu7, W1) + b1
```

```
    y_out = tf.reshape(y_out, [-1, ])
    regularizers += tf.nn.l2_loss(W1)

    return y_out, regularizers


def icp_res_model(X):
    X = tf.reshape(X, [-1, 10, 1, 1])
    regularizers = 0

    # layer 1
    Wconv1 = tf.get_variable("Wconv1", shape=[5, 1, 1, 64])
    bconv1 = tf.get_variable("bconv1", shape=[64])
    conv1 = tf.nn.conv2d(X, Wconv1, strides=[1,1,1,1], padding='SAME') + bconv1
    #batch1 = tf.layers.batch_normalization(conv1, training=is_training)
    relu1 = tf.nn.relu(conv1)
    regularizers += tf.nn.l2_loss(Wconv1)

    # layer 2
    Wconv2 = tf.get_variable("Wconv2", shape=[1, 1, 64, 128])
    bconv2 = tf.get_variable("bconv2", shape=[128])
    conv2 = tf.nn.conv2d(relu1, Wconv2, strides=[1,1,1,1], padding='SAME') + bconv2
    #batch2 = tf.layers.batch_normalization(conv2, training=is_training)
    relu2 = tf.nn.relu(conv2)
    regularizers += tf.nn.l2_loss(Wconv2)

    # layer 3, with a shortcut
    Wconv3 = tf.get_variable("Wconv3", shape=[3, 1, 128, 128])
    bconv3 = tf.get_variable("bconv3", shape=[128])
    conv3 = tf.nn.conv2d(relu2, Wconv3, strides=[1,1,1,1], padding='SAME') + bconv3
    #batch3 = tf.layers.batch_normalization(conv3, training=is_training)
    #batch3 += relu1
    relu3 = tf.nn.relu(conv3)
    regularizers += tf.nn.l2_loss(Wconv3)

    # inception residual layer4
    Wconv4_1 = tf.get_variable("Wconv4_1", shape=[1, 1, 128, 64])
    bconv4_1 = tf.get_variable("bconv4_1", shape=[64])
    conv4_1 = tf.nn.conv2d(relu3, Wconv4_1, strides=[1,1,1,1], padding='SAME') + bconv4_1
    #batch4_1 = tf.layers.batch_normalization(conv4_1, training=is_training)
    relu4_1 = tf.nn.relu(conv4_1)
    regularizers += tf.nn.l2_loss(Wconv4_1)

    Wconv4_2a = tf.get_variable("Wconv4_2a", shape=[1, 1, 128, 32])
```

```
bconv4_2a = tf.get_variable("bconv4_2a", shape=[32])
conv4_2a = tf.nn.conv2d(relu3, Wconv4_2a, strides=[1,1,1,1], padding='SAME') + bconv4_2a
#batch4_2a = tf.layers.batch_normalization(conv4_2a, training=is_training)
relu4_2a = tf.nn.relu(conv4_2a)
regularizers += tf.nn.l2_loss(Wconv4_2a)

Wconv4_2b = tf.get_variable("Wconv4_2b", shape=[3, 1, 32, 64])
bconv4_2b = tf.get_variable("bconv4_2b", shape=[64])
conv4_2b = tf.nn.conv2d(relu4_2a, Wconv4_2b, strides=[1,1,1,1], padding='SAME') + bconv4_2b
#batch4_2b = tf.layers.batch_normalization(conv4_2b, training=is_training)
relu4_2b = tf.nn.relu(conv4_2b)
regularizers += tf.nn.l2_loss(Wconv4_2b)

Wconv4_3a = tf.get_variable("Wconv4_3a", shape=[1, 1, 128, 32])
bconv4_3a = tf.get_variable("bconv4_3a", shape=[32])
conv4_3a = tf.nn.conv2d(relu3, Wconv4_3a, strides=[1,1,1,1], padding='SAME') + bconv4_3a
#batch4_3a = tf.layers.batch_normalization(conv4_3a, training=is_training)
relu4_3a = tf.nn.relu(conv4_3a)
regularizers += tf.nn.l2_loss(Wconv4_3a)

Wconv4_3b = tf.get_variable("Wconv4_3b", shape=[5, 1, 32, 64])
bconv4_3b = tf.get_variable("bconv4_3b", shape=[64])
conv4_3b = tf.nn.conv2d(relu4_3a, Wconv4_3b, strides=[1,1,1,1], padding='SAME') + bconv4_3b
#batch4_3b = tf.layers.batch_normalization(conv4_3b, training=is_training)
relu4_3b = tf.nn.relu(conv4_3b)
regularizers += tf.nn.l2_loss(Wconv4_3b)

Wconv4_4a = tf.get_variable("Wconv4_4a", shape=[1, 1, 128, 32])
bconv4_4a = tf.get_variable("bconv4_4a", shape=[32])
conv4_4a = tf.nn.conv2d(relu3, Wconv4_4a, strides=[1,1,1,1], padding='SAME') + bconv4_4a
#batch4_4a = tf.layers.batch_normalization(conv4_4a, training=is_training)
relu4_4a = tf.nn.relu(conv4_4a)
regularizers += tf.nn.l2_loss(Wconv4_4a)

Wconv4_4b = tf.get_variable("Wconv4_4b", shape=[3, 1, 32, 64])
bconv4_4b = tf.get_variable("bconv4_4b", shape=[64])
conv4_4b = tf.nn.atrous_conv2d(relu4_4a, Wconv4_4b, rate=2, padding='SAME') + bconv4_4b
#batch4_4b = tf.layers.batch_normalization(conv4_4b, training=is_training)
relu4_4b = tf.nn.relu(conv4_4b)
regularizers += tf.nn.l2_loss(Wconv4_4b)

relu4 = tf.concat([relu4_1, relu4_2b, relu4_3b, relu4_4b], 3)

# inception residual layer5
```

```
Wconv5_1 = tf.get_variable("Wconv5_1", shape=[1, 1, 256, 128])
bconv5_1 = tf.get_variable("bconv5_1", shape=[128])
conv5_1 = tf.nn.conv2d(relu4, Wconv5_1, strides=[1,1,1,1], padding='SAME') + bconv5_1
#batch5_1 = tf.layers.batch_normalization(conv5_1, training=is_training)
relu5_1 = tf.nn.relu(conv5_1)
regularizers += tf.nn.l2_loss(Wconv5_1)

Wconv5_2a = tf.get_variable("Wconv5_2a", shape=[1, 1, 256, 64])
bconv5_2a = tf.get_variable("bconv5_2a", shape=[64])
conv5_2a = tf.nn.conv2d(relu4, Wconv5_2a, strides=[1,1,1,1], padding='SAME') + bconv5_2a
#batch5_2a = tf.layers.batch_normalization(conv5_2a, training=is_training)
relu5_2a = tf.nn.relu(conv5_2a)
regularizers += tf.nn.l2_loss(Wconv5_2a)

Wconv5_2b = tf.get_variable("Wconv5_2b", shape=[3, 1, 64, 128])
bconv5_2b = tf.get_variable("bconv5_2b", shape=[128])
conv5_2b = tf.nn.conv2d(relu5_2a, Wconv5_2b, strides=[1,1,1,1], padding='SAME') + bconv5_2b
#batch5_2b = tf.layers.batch_normalization(conv5_2b, training=is_training)
relu5_2b = tf.nn.relu(conv5_2b)
regularizers += tf.nn.l2_loss(Wconv5_2b)

Wconv5_3a = tf.get_variable("Wconv5_3a", shape=[1, 1, 256, 64])
bconv5_3a = tf.get_variable("bconv5_3a", shape=[64])
conv5_3a = tf.nn.conv2d(relu4, Wconv5_3a, strides=[1,1,1,1], padding='SAME') + bconv5_3a
#batch5_3a = tf.layers.batch_normalization(conv5_3a, training=is_training)
relu5_3a = tf.nn.relu(conv5_3a)
regularizers += tf.nn.l2_loss(Wconv5_3a)

Wconv5_3b = tf.get_variable("Wconv5_3b", shape=[5, 1, 64, 128])
bconv5_3b = tf.get_variable("bconv5_3b", shape=[128])
conv5_3b = tf.nn.conv2d(relu5_3a, Wconv5_3b, strides=[1,1,1,1], padding='SAME') + bconv5_3b
#batch5_3b = tf.layers.batch_normalization(conv5_3b, training=is_training)
relu5_3b = tf.nn.relu(conv5_3b)
regularizers += tf.nn.l2_loss(Wconv5_3b)

Wconv5_4a = tf.get_variable("Wconv5_4a", shape=[1, 1, 256, 64])
bconv5_4a = tf.get_variable("bconv5_4a", shape=[64])
conv5_4a = tf.nn.conv2d(relu4, Wconv5_4a, strides=[1,1,1,1], padding='SAME') + bconv5_4a
#batch5_4a = tf.layers.batch_normalization(conv5_4a, training=is_training)
relu5_4a = tf.nn.relu(conv5_4a)
regularizers += tf.nn.l2_loss(Wconv5_4a)

Wconv5_4b = tf.get_variable("Wconv5_4b", shape=[3, 1, 64, 128])
bconv5_4b = tf.get_variable("bconv5_4b", shape=[128])
```

```
    conv5_4b = tf.nn.atrous_conv2d(relu5_4a, Wconv5_4b, rate=2, padding='SAME') + bconv5_4b
    #batch5_4b = tf.layers.batch_normalization(conv5_4b, training=is_training)
    relu5_4b = tf.nn.relu(conv5_4b)
    regularizers += tf.nn.l2_loss(Wconv5_4b)

    relu5 = tf.concat([relu5_1, relu5_2b, relu5_3b, relu5_4b], 3)

    # 1 fc
    W1 = tf.get_variable("W1", shape=[10 * 512, 1])
    b1 = tf.get_variable("b1", shape=[1])
    relu6 = tf.reshape(relu5, [-1, 10 * 512])
    y_out = tf.matmul(relu6, W1) + b1
    y_out = tf.reshape(y_out, [-1, ])
    regularizers += tf.nn.l2_loss(W1)

    return y_out, regularizers


#y_out, regularizers = conv7_64_res_model(X)
y_out, regularizers = icp_res_model(X)
my_y_out = tf.reshape(y_out, [-1, 1], name="output_y")

total_loss = tf.losses.mean_squared_error(y, y_out)
mean_loss = tf.reduce_mean(total_loss + regularizers * args.reg)

# define our optimizer
global_step = tf.Variable(0, trainable=False)
starter_learning_rate = args.lr
learning_rate = tf.train.exponential_decay(starter_learning_rate, global_step, args.ds, args.dr, staircase=Tr
ue)
optimizer = tf.train.AdamOptimizer(learning_rate) # select optimizer and set learning rate
train_step = optimizer.minimize(mean_loss)

# batch normalization in tensorflow requires this extra dependency
'''extra_update_ops = tf.get_collection(tf.GraphKeys.UPDATE_OPS)
train_step = optimizer.minimize(mean_loss)
train_step = tf.group([train_step, extra_update_ops])'''
'''with tf.control_dependencies(extra_update_ops):
    train_step = optimizer.minimize(mean_loss)'''

#init_op = tf.initialize_all_variables()

sess = tf.Session()
sess.run(tf.global_variables_initializer())
```

```
print('Training')


def run_model(session, predict, loss_val, Xd, yd,
              epoch=1, batch_size=64, print_every=100,
              training=None):
    # shuffle indicies
    train_indices = np.arange(Xd.shape[0])
    if training is not None:
        np.random.shuffle(train_indices)

    training_now = training is not None

    # setting up variables we want to compute (and optimizing)
    # if we have a training function, add that to things we compute
    variables = [loss_val, predict]
    if training_now:
        variables += [training]

    # counter
    iter_cnt = 0
    # keep track of losses and accuracy
    losses = []
    y_output = np.zeros((0, ))
    # make sure we iterate over the dataset once
    for i in range(int(np.ceil(Xd.shape[0] / batch_size))):
        # generate indicies for the batch
        start_idx = (i * batch_size) % Xd.shape[0]
        idx = train_indices[start_idx: start_idx + batch_size]

        # create a feed dictionary for this batch
        feed_dict = {X: Xd[idx,:],
                     y: yd[idx],}
                     #is_training: training_now}
        # get batch size
        actual_batch_size = yd[idx].shape[0]

        # have tensorflow compute loss and correct predictions
        # and (if given) perform a training step
        if training_now:
            loss, output, _ = session.run(variables, feed_dict=feed_dict)
        else:
            loss, output = session.run(variables, feed_dict=feed_dict)
        y_output = np.concatenate((y_output, output))
```

```
        # aggregate performance stats
        losses.append(loss * actual_batch_size)

        # print every now and then
        if training_now and (iter_cnt % print_every) == 0:
            print("Iteration {0}: with minibatch training loss = {1:.3f}"\
                .format(iter_cnt, loss))
        iter_cnt += 1

    total_loss = np.sum(losses) / Xd.shape[0]
    print("Epoch {1}, Overall loss = {0:.3f}"\
        .format(total_loss, epoch + 1))

    return total_loss, y_output


def CaculatePcrr(y_true, y_pred):
    t = -103
    tp = len(y_true[(y_true < t) & (y_pred < t)])
    fp = len(y_true[(y_true >= t) & (y_pred < t)])
    fn = len(y_true[(y_true < t) & (y_pred >= t)])
    precision = tp / (tp + fp)
    recall = tp / (tp + fn)
    pcrr = 2 * (precision * recall) / (precision + recall)
    return pcrr


for i in range(args.epoch):
    run_model(sess, y_out, mean_loss, X_train_all, y_train_all, i, args.batch_size, args.print_freq, train_step)
    _, y_output = run_model(sess, y_out, mean_loss, X_val_all, y_val_all, i, args.batch_size)
    pcrr = CaculatePcrr(y_val_all, y_output)
    print("pcrr: {}".format(pcrr))
print("Finish Training !")

tf.saved_model.simple_save(sess, args.save_path,
                           inputs={"myInput":  X},
                           outputs={"myOutput": my_y_out})

sess.close()
```