中国研究生创新实践系列大赛

“华为杯”第十八届中国研究生

数学建模竞赛

学　　校	福州大学
参赛队号	21103860006
队员姓名	1.谢志强
	2.黄浩阳
	3.施嘉豪

中国研究生创新实践系列大赛

“华为杯”第十八届中国研究生

数学建模竞赛

题 目 信号干扰下的超宽带（UWB）精确定位问题

摘 要：

本文对信号干扰下的超宽带（UWB）精确定位问题进行探索和研究。主要创新点在于结合粒子群算法、最小二乘法等寻优算法针对正常数据和异常数据进行有差别拓展空间位置特征数据，并使用山岭回归、麻雀算法优化极限学习机及其他机器学习算法建立相应的预测模型和分类模型，进而完成各项任务。

问题一：本文对原始数据进行了预处理(1)针对不良数据，本文计算 4 个靶点与锚点的理论距离，统计理论距离与题目所给的测量距离之间的误差，绘制直方图并分析，最终剔除不良数据 2966 组，其中剔除正常数据中的不良数据 1052 组，异常数据中的不良数据 1914 组；(2)针对重复数据，本文使用 k-mean 聚类方法筛选出每一个点中具有代表性的数据，其中正常数据 1296 组，异常数据 1296 组作为后续建模训练和测试数据。

问题二：本文针对“正常数据”预测和“异常数据”预测，设计了两种定位模型（算法）：(1)对于“正常数据”预测，本文利用题目提供的靶点到 4 个锚点的距离，结合最小二乘法求解出其近似坐标作为拓展的特征数据进行预测训练，最后用山岭回归算法进一步预测其准确坐标。在测试结果中，x 轴平均误差为 36.22mm，y 轴平均误差为 30.00mm，z 轴平均误差为 159.88mm，二维平均误差为 52.43mm，三维平均误差为 175.71mm；(2)对于“异常数据”预测，本文根据锚点个数建立以锚点坐标为球心，以靶点到该锚点的距离为半径的球面方程，并以靶点到其中 3 个球面的距离之和作为目标函数进行粒子群算法寻优求解近似坐标，将求解的 4 个近似坐标作为特征数据，使用机器学习算法进一步预测其准确坐标。在测试结果中，x 轴平均误差为 135.44mm，y 轴平均误差为 128.09mm，z 轴平均误差为 373.44mm，二维平均误差为 193.83mm，三维平均误差为 437.50mm。

问题三：由于问题二的数学模型算法是利用空间几何关系所建立的，因此该数学模型（算法）可以满足在不同场景的使用。

问题四：根据题目要求，需要区分数据是否在信号干扰下采集，这是个二分类问题。本文利用同问题二方法对问题一处理后的数据进行特征拓展，并将拓展后的多维特征数据通过基于麻雀搜索算法的优化极限学习机模型进行分类训练测试，在测试结果中，正常数据正确个数为 252，错误个数为 3，异常数据正确个数为 231，错误个数为 25，整体分类准确率为 94.53%，精确率为 91.01%，召回率为 98.83%，整理分类结果较为理想。

问题五：本任务需要预测动态靶点的轨迹，但在靶点运动过程中存在随机性的 UWB 信号干扰。本文首先利用问题四中的分类模型来区分动态轨迹中的数据是否受到干扰，再使用问题二中的分类模型结合 Kmean 聚类算法找出定位异常的轨迹点，结合动态轨迹数据存在时间关联的特性对 z 轴坐标进行修正，最终预测出动态轨迹曲线。

关键字: UWB 定位，无监督聚类，寻优算法，数据特征拓展，机器学习

目录

1 问题重述

1.1 问题背景

UWB（Ultra-Wideband）技术也被称之为“超宽带”，又称之为脉冲无线电技术。这是一种无需任何载波，通过发送纳秒级脉冲而完成数据传输的短距离范围内无线通信技术，并且信号传输过程中的功耗仅仅有几十μW[1]。UWB 因其独有的特点，使其在军事、物联网等各个领域都有着广阔的应用。其中，基于 UWB 的定位技术具备实时的室内外精确跟踪能力，定位精度高，可达到厘米级甚至毫米级定位。UWB 在室内精确的定位将会对卫星导航起到一个极好的补充作用，可在军事及民用领域有广泛应用，比如：电力、医疗、化工行业、隧道施工、危险区域管控等[2]。

UWB 的定位技术有多种方法，本文仅考虑基于飞行时间（Time of Flight, TOF）的测距原理，它是 UWB 定位法中最常见的定位方法之一。TOF 测距技术属于双向测距技术，其通过计算信号在两个模块的飞行时间，再乘以光速求出两个模块之间的距离，这个距离肯定有不同程度。

在室内定位的应用中，UWB 技术可以实现厘米级的定位精度（一般指 2 维平面定位）,并具有良好的抗多径干扰和衰弱的性能以及具有较强的穿透能力。但由于室内环境复杂多变 UWB 通信信号极易受到遮挡[3]，虽然 UWB 技术具有穿透能力，但仍然会产生误差，在较强干扰时，数据会发生异常波动（通常是时间延时），基本无法完成室内定位，甚至会造成严重事故。因此，信号干扰下的超宽带（UWB）精确定位问题成为亟待解决的问题。

1.2 问题提出

题目提供了 3 个数据文件，其中一个文件存放了 324 个不同位置的编号以及 3 维坐标信息，另外两个文件分别存放了 UWB 无干扰的正常数据和 UWB 干扰的异常数据，要求利用该数据解决如下问题：

问题一：在数据获取过程中，由于采集设备、周围环境等因素的影响，可能导致数据出现错误或者偏差，原始数据的质量无法保证。其中往往会包含一些不良数据和重复数据值，需要首先进行数据筛选、数据清洗等一系列数据预处理操作。

问题二：分别对“正常数据”和“异常数据”，设计合适的数学模型（或算法），估计（或预测）出靶点的精确位置，并说明所建立的定位模型（或算法）的有效性；同时请利用你的定位模型（或算法）分别对附件 2 中提供的前 5 组（信号无干扰）数据和后 5 组（信号有干扰）数据进行精确定位（3 维坐标）。

问题三：训练数据仅采集于同一实验场景（实验场景 1），但定位模型应该能够在不同实际场景上使用，希望所建立的定位模型能够应用于不同场景。附件 3 中 10 组数据采集于下面实验场景 2（前 5 组数据信号无干扰，后 5 组数据信号有干扰），请分别用上述建立的定位模型，对这 10 组数据进行精确定位（3 维坐标）。

问题四：上述定位模型是在已知信号有、无干扰的条件下建立的，但 UWB 在采集数据时并不知道信号是否被干扰，所以判断信号是否被干扰是 UWB 精确

定位问题的重点和难点。利用问题一处理后的数据，建立数学模型(或算法)，以辨别数据是在信号无干扰下采集的还是在信号干扰下的采集的，并说明所建立的分类模型(或算法)的有效性；同时请用所建立的分类模型（或算法）辨别附件 4 中提供的 10 组数据（这 10 组数据同样采集于实验场景 1）是在信号无干扰还是在信号干扰下采集的。

问题五：利用静态点的定位模型，加上靶点自身运动规律，希望给出动态靶点的运动轨迹。附件 5 是对动态靶点采集的数据（一段时间内连续采集的多组数据），请注意，在采集这些数据时，会随机出现信号干扰，请对这个运动轨迹进行精确定位，最终画出这条运动轨迹图（数据采集来自实验场景 1）。

2 问题分析

问题一：根据题目要求，采集到的原始数据无法直接使用，附件 1 所提供的数据是由 UWB 数据采集设备直接记录的，原始采集数据包括时间及靶点到分别到四个锚点（A0，A1，A2，A3）的四个距离。原始数据在采集过程中可能由于环境、设备等因素，导致数据异常，且存在某些原因，导致采集的原始数据存在相同或者相似数据，如果直接使用这些不良数据得出的建模结果也会出现偏差。因此，本文利用题目提供的靶点真实坐标和锚点坐标计算二者之间的理论距离，进而计算理论距离和测量距离的总误差，从而使用直方图对总误差进行分析，发现“正常数据”和“异常数据”均存在数据异常情况。针对数据异常值情况，本文选用 3sigma 原则筛选原始数据中的异常值并将其剔除。另一方面，针对原始数据存在重复性的情况，本文选用 K-mean 聚类算法将各数据文件数据分为四类，并在每类数据中随机选出一组数据作为后续建立模型的数据集。

问题二：本文针对“正常数据”预测和“异常数据”预测，分别设计了两种定位模型（算法）。对于“正常数据”的特征提取，利用题目提供的靶点到 4 个锚点的距离，结合最小二乘法求解出其近似坐标作为拓展的特征数据，并将多维特征数据集通过山岭回归算法进一步预测其准确坐标。对“异常数据”预测，本文建立以锚点坐标为球心，以靶点到该锚点距离为半径的球面方程，以靶点到 3 个球面的距离之和作为目标函数结合粒子群算法寻优求解近似坐标。将求解的近似坐标作为特征数据，使用山岭回归算法进一步预测其准确坐标。

问题三：本文对于问题二建立的数学模型算法，是利用空间几何关系建立的，因此本文数学模型（算法）可以满足在不同场景的使用。

问题四：根据题目要求，需要区分数据是否在信号干扰下采集，这是个二分类问题。同问题二一样，本题的关键在于特征参数的提取，因此利用同问题二方法对问题一处理后的数据进行特征拓展，并将拓展后的多维特征数据通过基于麻雀搜索算法的优化极限学习机模型进行分类训练测试。

问题五：本任务需要预测动态靶点的轨迹，但在靶点运动过程中存在随机性的 UWB 信号干扰。本文首先利用问题四中的分类模型来区分动态轨迹中的数据是否受到干扰，再使用问题二中“正常预测模型”和“异常预测模型”分别对“正常数据”和“异常数据”进行坐标估计，最后结合动态轨迹数据存在时间关联的特性通过聚类算法对 z 轴坐标进行进一步修正，最终得出完整的动态轨迹图。

3 模型假设

1.本文认为题目提供的大部分数据都是可靠的。
2.假设再采集的轨迹数据时靶点是匀速运动的。
3.去除异常数据，不影响数据的整体分布情况。

4 符号说明

序号	符号	符号说明	单位
1	(x_0, y_0, z_0)	锚点 A0 坐标	mm
2	(x_0, y_0, z_0)	锚点 A1 坐标	mm
3	(x_0, y_0, z_0)	锚点 A2 坐标	mm
4	(x_0, y_0, z_0)	锚点 A3 坐标	mm
5	d_0	靶点与 A0 的距离测量值	mm
6	d_1	靶点与 A1 的距离测量值	mm
7	d_2	靶点与 A2 的距离测量值	mm
8	d_3	靶点与 A3 的距离测量值	mm
9	$d_0^{'}$	靶点估计坐标与 A0 的距离	mm
10	$d_1^{'}$	靶点估计坐标与 A1 的距离	mm
11	$d_2^{'}$	靶点估计坐标与 A2 的距离	mm
12	$d_3^{'}$	靶点估计坐标与 A3 的距离	mm
13	d_{0T}	靶点与 A0 的距离理论值	mm
14	d_{1T}	靶点与 A1 的距离理论值	mm
15	d_{2T}	靶点与 A2 的距离理论值	mm
16	d_{3T}	靶点与 A3 的距离理论值	mm
13	Δd_0	d_0与$d_0^{'}$的差值	mm
14	Δd_1	d_1与$d_1^{'}$的差值	mm
15	Δd_2	d_2与$d_2^{'}$的差值	mm
16	Δd_3	d_3与$d_3^{'}$的差值	mm
17	Δd_{0T}	d_0与$\mathrm{d_{0T}}$的差值	mm
18	Δd_{1T}	d_1与$\mathrm{d_{1T}}$的差值	mm
19	Δd_{2T}	d_2与$\mathrm{d_{2T}}$的差值	mm
20	Δd_{3T}	d_3与$\mathrm{d_{3T}}$的差值	mm
21	$\Delta \mathrm{D}$	$\Delta \mathrm{d_{0T}}$、$\Delta \mathrm{d_{1T}}$、$\Delta \mathrm{d_{2T}}$、$\Delta \mathrm{d_{3T}}$之和	mm

5 问题一数据预处理

5.1 UWB 数据格式整理

题目提供的数据文件格式为 TXT 文件，TXT 数据文件并不适合直接计算分析，因此本文将数据统一成 CSV 数据格式，便于后续分析、筛选和训练，如图 5-1 所示。每个 TXT 数据文件中 4 行数据为一组，每行第四个字段为锚点到靶点的距离，一共有 4 个距离，因此利用 python 将 4 个距离值进行提取，将正常数据和异常数据分别分成 324 个 CSV 文件，共计 648 个 CSV 文件。以靶点的标识为序，每个 CSV 文件保存一个尺寸为 n×7 的距离矩阵数据，其中 n 为每个文件的数据组数，每组数据均由 4 个测量距离及三维坐标 7 个参数组成。

图 5-1 数据提取

5.2 数据分析

由于附件 1 所给的数据不论是正常数据或异常数据均有可能出现不良数据（测量值明显反常或确实的数据），针对该问题，本文通过计算锚点到靶点的理论距离值并使用直方图统计其分布情况来分析所有数据的正确性。

距离计算公式如下：

$$|d_{0T} - d_0| = \Delta d_{0T} \tag{5-1}$$

$$|d_{1T} - d_1| = \Delta d_{1T} \tag{5-2}$$

$$|d_{2T} - d_2| = \Delta d_{2T} \tag{5-3}$$

$$|d_{3T} - d_3| = \Delta d_{3T} \tag{5-4}$$

$$\Delta d_{0T} + \Delta d_{1T} + \Delta d_{2T} + \Delta d_{3T} = \Delta D \tag{5-5}$$

通过上式 5-1~5-5 计算出每组数据的理论距离与实测距离之间的 4 个误差，以及误差之和ΔD，部分计算值如下表 5-1：

表 5-1　误差计算结果表

正常数据					异常数据				
Δd_{0T}	Δd_{1T}	Δd_{2T}	Δd_{3T}	ΔD	Δd_{0T}	Δd_{1T}	Δd_{2T}	Δd_{3T}	ΔD
62	51	51	77	242	457	51	51	77	638
74	22	3	105	232	357	7	6	85	475
55	43	0	96	196	404	43	0	96	544
103	22	69	114	310	356	22	69	114	562
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

观察上表可发现，正常数据的 4 个Δd_{0T}一般情况下较小，而异常数据中会有至少 1 个Δd_{1T}与其他的误差相差较大。提取所有数据的ΔD 绘制出下图 5-2 所示的误差分布直方图：

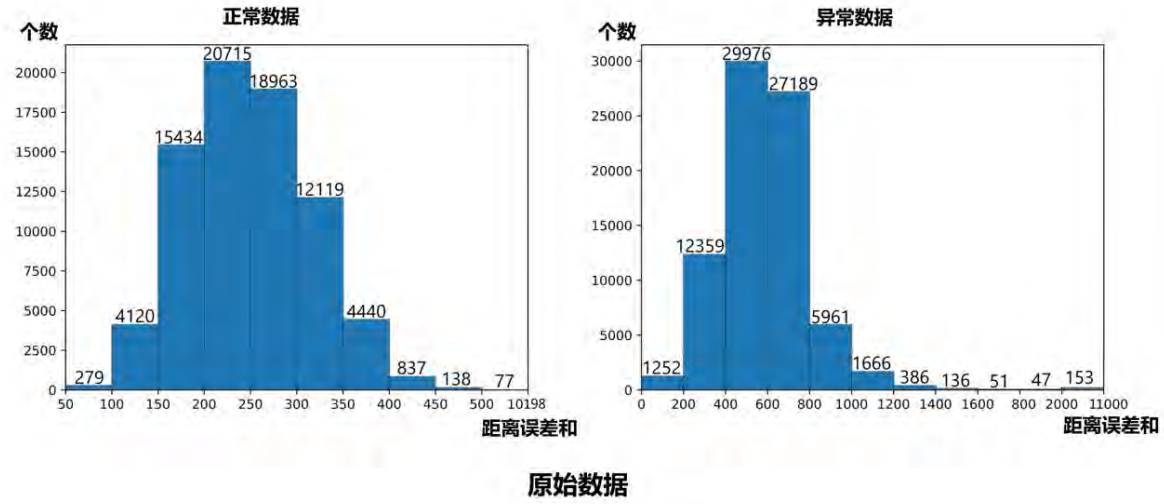

图 5-2　原始数据距离误差直方图

观察上图可知，不论是正常数据还是异常数据的距离误差和ΔD 总体近似正态分布，正常数据大部分分布在 100mm-400mm，数据较为紧凑，ΔD 平均值为 249.73mm。异常数据大部分分布在 200mm-1000mm 分布较广，ΔD 平均值为 585.36mm，比异常值多出 134.40%。通过上述分析，本文认为通过理论值与实际值的误差及误差和能够在一定程度上反映数据的特征及分布情况。

5.3 数据清洗

经 5.2 分析，考虑数据采集时可能由于 UWB 信号发射器的稳定性或周围环境的影响，导致小部分数据偏差太大，根据拉伊达准则，已经不属于随机误差，而是粗大误差，需要对不良数据进行第一次剔除。

拉依达准则为：

数值分布在（μ-σ,μ+σ)中的概率为 0.6826；

数值分布在（μ-2σ,μ+2σ)中的概率为 0.9545；

数值分布在（μ-3σ,μ+3σ)中的概率为 0.9973。

没有分布在（μ-3σ,μ+3σ)的数据大约占 0.3%，超出这个范围的属于几乎不会发生事件，使用拉依达准则筛除后，距离总误差分布直方图图如下图 5-3 所示：

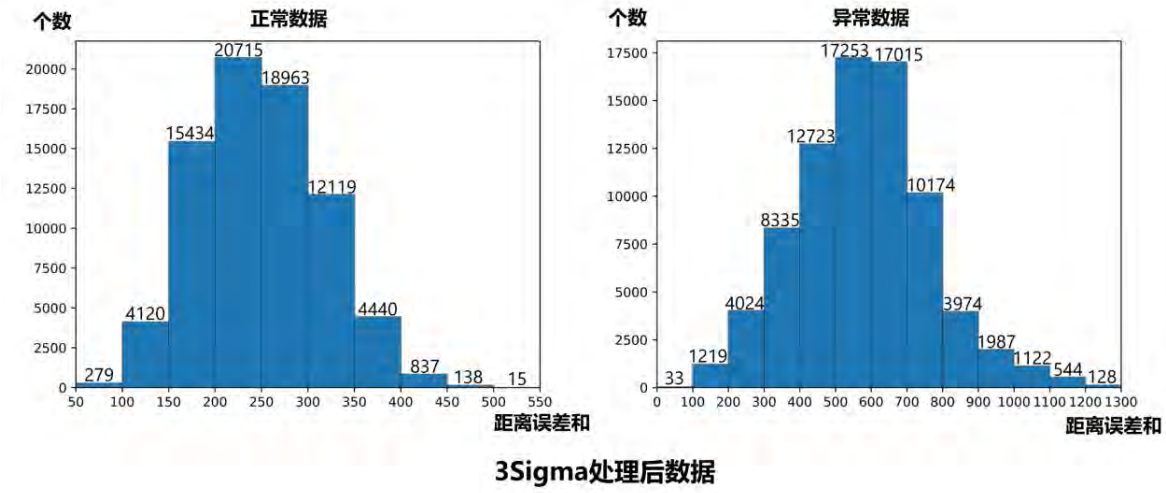

图 5-3 拉依达准处理后误差分布图

使用拉依达准则筛选数据后发现，“正常数据”和“异常数据”仍存在部分数据，其距离总误差很大，不符合实际情况。因此进行第二次数据剔除，将“正常数据”距离总误差大于 400mm 的数据剔除，同时将“异常数据”距离总误差小于 200mm 的剔除。剔除后的总误差分布情况如图 5-4 所示：

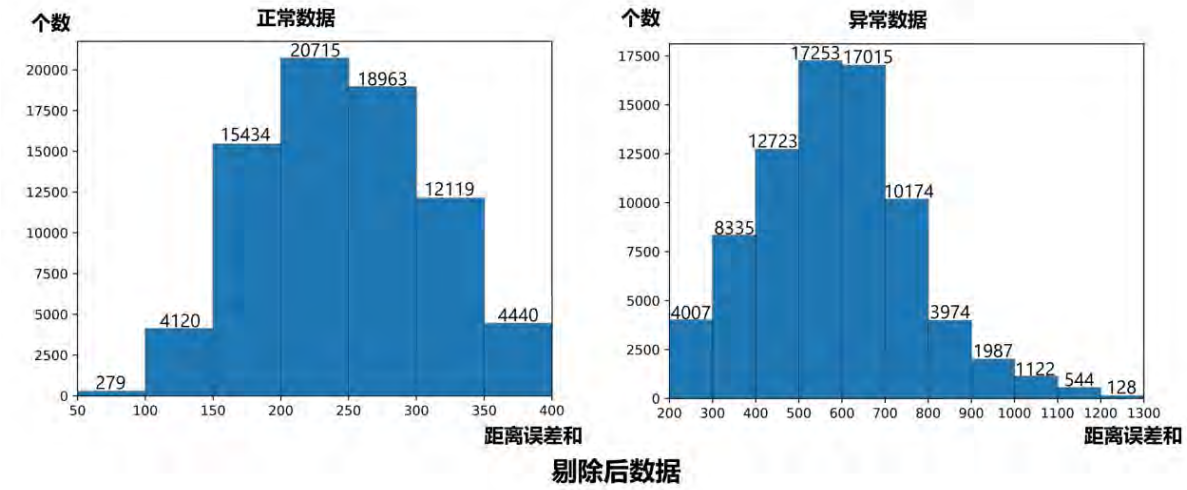

图 5-4 剔除异常值后误差分布图

5.4 数据筛选

进行数据清洗后，剩下的数据中还有一部分重复数据，或者十分相近的数据，一般的做法是直接对比找出相同或相似的数据将其删去。但是由于数据量较大，并且相似的数据较多，为避免选到相似度较大的数据，同时希望能够选择比较有代表性的数据，本文采用 K-means 算法对数据进行聚类，如图 5-5 所示。将正常和异常数据里的每个点数据分成四类后，选取距离簇质心最近的样本作为该类的代表数据。

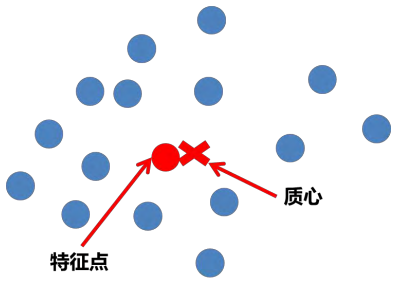

图 5-5 数据筛选

5.5 数据集划分

根据题目所提供的数据集可知，数据集是在 4 个不同的高度环境。每层高度有 81 个点且成均匀分布。因此本文将如图 5-6 所示在每层数据中均匀取出 16 个点，一共取出 64 个点作为测试集验证模型，其他点数据均作为训练集，其中每个点包括 4 组数据，故最终取得训练集 2080 组数据，测试集 512 组数据。这样划分数据集可以保证训练集、测试集的数据都在空间上均匀分布，一定程度上减少了数据因空间分布不均匀而导致的误差。

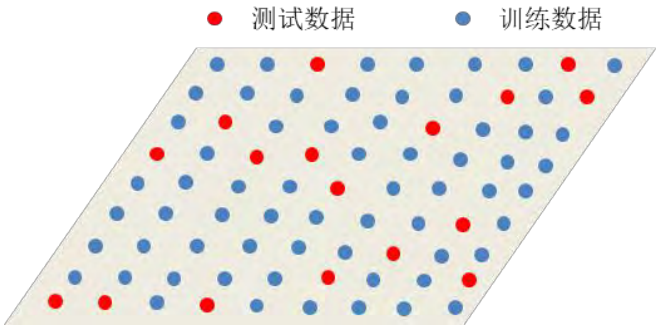

图 5-6 数据集划分示意图

5.6 数据处理结果

根据上述方法对不良数据进行处理，各类数据经处理后的结果如表 5-2 所示：

表 5-2 附件 1 数据预处理结果

数据类别	原始数据个数(组)	数据清洗后个数(组)	数据筛选后个数(组)
正常数据	77122	76070	1296
异常数据	79176	77260	1296

“正常数据”保留结果如表 5-3 所示：

表 5-3 正常数据保留结果

序号	d_0	d_1	d_2	d_3	x	y	z
1	760	4550	4550	6300	500	500	880
2	770	4550	4550	6300	500	500	880
3	760	4540	4550	6300	500	500	880
4	780	4550	4550	6300	500	500	880
5	1760	3610	4240	5320	1500	1000	1300
6	1770	3620	4230	5320	1500	1000	1300
7	1750	3600	4220	5320	1500	1000	1300
8	1770	3610	4230	5330	1500	1000	1300
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
1293	930	4100	4520	6000	1000	500	1300
1294	930	4090	4530	6010	1000	500	1300
1295	930	4100	4530	6000	1000	500	1300
1296	940	4090	4530	6010	1000	500	1300

“异常数据”保留结果如表 5-4 所示：

表 5-4 异常数据保留结果

序号	d_0	d_1	d_2	d_3	x	y	z
1	770	5040	4540	6300	500	500	880
2	1250	4550	4550	6300	500	500	880
3	770	5000	4540	6290	500	500	880
4	1280	4550	4550	6300	500	500	880
5	2000	3620	4230	5350	1500	1000	1300
6	1760	3800	4230	5280	1500	1000	1300
7	2160	3620	4250	5330	1500	1000	1300
8	1730	4120	4230	5320	1500	1000	1300
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
1293	1390	4090	4530	6010	1000	500	1300
1294	940	4100	4510	6480	1000	500	1300
1295	940	4090	4510	6520	1000	500	1300
1296	1450	4100	4530	6000	1000	500	1300

题中所给四个数据文件的数据保留结果如表 5-5 所示：

表 5-5 指定文件数据保留结果

序号	d_0	d_1	d_2	d_3	x	y	z
24.正常	3280	4660	2600	3910	1500	3000	880
	3280	4660	2590	3890	1500	3000	880
	3280	4660	2600	3900	1500	3000	880
	3280	4670	2600	3920	1500	3000	880
109.正常	4890	5310	2020	2880	2000	4500	1300
	4910	5320	2040	2880	2000	4500	1300
	4910	5310	2030	2870	2000	4500	1300
	4890	5330	2030	2880	2000	4500	1300
1.异常	770	5040	4540	6300	500	500	880
	1250	4550	4550	6300	500	500	880
	770	5000	4540	6290	500	500	880
	1280	4550	4550	6300	500	500	880
100.异常	1510	3560	5010	5710	1500	500	1300
	1510	3750	4690	5710	1500	500	1300
	1500	3570	5880	5690	1500	500	1300
	1510	3560	5350	5700	1500	500	1300

最终，本文根据 5.5 提取出具有空间分布均匀性的训练集数据 2080 组，测试集 512 组。

6 问题二定位模型

6.1 “正常数据”定位模型

6.1.1 特征提取

本文建立以锚点坐标为球心，以靶点到该锚点距离为半径的球面方程，由题目提供的信息可知场景 1 中有四个锚点。因此可以建立四个球面方程分别为：

以锚点 A0 为球心建立球面公式（6-1）：

$$(x-x_0)^2+(y-y_0)^2+(z-z_0)^2={d_0}^2 \tag{6-1}$$

以锚点 A1 为球心建立球面公式（6-2）：

$$(x-x_1)^2+(y-y_1)^2+(z-z_1)^2={d_1}^2 \tag{6-2}$$

以锚点 A2 为球心建立球面公式（6-3）：

$$(x-x_2)^2+(y-y_2)^2+(z-z_2)^2={d_2}^2 \tag{6-3}$$

以锚点 A3 为球心建立球面公式（6-4）：

$$(x-x_3)^2+(y-y_3)^2+(z-z_3)^2={d_3}^2 \tag{6-4}$$

然后使用粒子群寻优算法求解靶点的近似坐标，其目标函数为靶点到 3 个球面的距离之和（其示意图如图 6-1 所示）。以靶点到 3 个球面的距离之和为例，其目标函数如（6-5）所示：

$$f=\begin{cases}\left|d_0-\sqrt{(x-x_0)^2-(y-y_0)^2-(z-z_0)^2}\right| \\ +\left|d_1-\sqrt{(x-x_1)^2-(y-y_1)^2-(z-z_1)^2}\right| \\ +\left|d_2-\sqrt{(x-x_2)^2-(y-y_2)^2-(z-z_2)^2}\right|\end{cases} \tag{6-5}$$

除此之外，本文使用最小二乘法解析三位空间坐标作为粒子群算法中粒子的初始位置。最小二乘的思路是将 4 个球面方程相减，去除其中未知的二次项，将方程变为线性方程，从而进行求解。

已知 4 个锚点的坐标(x_0,y_0,z_0)，(x_1,y_1,z_1)，(x_2,y_2,z_2)，(x_3,y_3,z_3)以及靶点到锚点的距离d_0，d_1，d_2，d_3。求靶点的坐标(x,y,z)。由已知条件，可得如下方程组(6-6)：

$$\begin{cases}(x_0-x)^2+(y_0-y)^2+(z_0-z)^2={d_0}^2 \\ (x_1-x)^2+(y_1-y)^2+(z_1-z)^2={d_1}^2 \\ (x_2-x)^2+(y_2-y)^2+(z_2-z)^2={d_2}^2 \\ (x_3-x)^2+(y_3-y)^2+(z_3-z)^2={d_3}^2\end{cases} \tag{6-6}$$

用方程组的前 3 个方程减去第 4 个方程，可得到如下线性方程(6-6)：

$$\begin{cases} {x_0}^2 - {x_3}^2 - 2(x_0 - x_3)x + {y_0}^2 - {y_3}^2 - 2(y_0 - y_3)y \\ \quad + {z_0}^2 - {z_3}^2 - 2(z_0 - z_3)z = {d_0}^2 - {d_3}^2 \\ {x_1}^2 - {x_3}^2 - 2(x_1 - x_3)x + {y_1}^2 - {y_3}^2 - 2(y_1 - y_3)y \\ \quad + {z_1}^2 - {z_3}^2 - 2(z_1 - z_3)z = {d_1}^2 - {d_3}^2 \\ {x_2}^2 - {x_3}^2 - 2(x_2 - x_3)x + {y_2}^2 - {y_3}^2 - 2(y_2 - y_3)y \\ \quad + {z_2}^2 - {z_3}^2 - 2(z_2 - z_3)z = {d_2}^2 - {d_3}^2 \end{cases} \tag{6-7}$$

该方程可以表示为：

$$AX = b \tag{6-8}$$

其中：

$$A = \begin{bmatrix} 2(x_0 - x_3) & 2(y_0 - y_3) & 2(z_0 - z_3) \\ 2(x_1 - x_3) & 2(y_1 - y_3) & 2(z_1 - z_3) \\ 2(x_2 - x_3) & 2(y_2 - y_3) & 2(z_2 - z_3) \end{bmatrix} \tag{6-9}$$

$$X = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{6-10}$$

$$b = \begin{bmatrix} (x_0 + y_0 + z_0)^2 - (x_3 + y_3 + z_3)^2 - ({d_0}^2 - {d_3}^2) \\ (x_1 + y_1 + z_1)^2 - (x_3 + y_3 + z_3)^2 - ({d_0}^2 - {d_3}^2) \\ (x_2 + y_2 + z_2)^2 - (x_3 + y_3 + z_3)^2 - ({d_0}^2 - {d_3}^2) \end{bmatrix} \tag{6-11}$$

解得

$$X = [A^{-1}A]^{-1}A^T b \tag{6-12}$$

将靶点到 3 个球面距离之和作为 1 个目标函数，一共有 4 个锚点建立了 4 个球面，从而进行组合可以得出 4 个目标函数，使用粒子群寻优算法分别对这 4 个目标函数寻优可以求解出靶点的 4 个近似坐标。然后将求解的 4 个近似坐标和最小二乘求解的近似坐标作为后续预测算法的特征数据。

6.1.2 预测算法

对于位置预测，本文使用岭回归预测，岭回归是一种专用于共线性数据分析的有偏估计回归方法，实质上是一种改良的最小二乘估计法，通过放弃最小二乘法的无偏性，以损失部分信息、降低精度为代价获得回归系数更为符合实际、更可靠的回归方法，是对不适定问题进行回归分析。

回归分析中常用的最小二乘法是一种无偏估计。对于一个适定问题，X 通常是列满秩的

$$X\theta = y \tag{6-13}$$

采用最小二乘法，定义损失函数为残差的平方，最小化损失函数

$$\|X\theta - y\|^2 \tag{6-14}$$

上述优化问题可以采用梯度下降法进行求解，也可以采用如下公式进行求解：

$$\theta = (X^{T}X)^{-1}X^{T}y \tag{6-15}$$

当 X 不是列满秩时，或者某些列之间的线性相关性比较大时，$X^{T}X$ 的行列式接近于 0，即$X^{T}X$ 接近于奇异，上述问题变为一个不适定问题，此时，计算$(X^{T}X)^{-1}$时误差会很大，传统的最小二乘法缺乏稳定性与可靠性。

为了解决上述问题，我们需要将不适定问题转化为适定问题：我们为上述损失函数加上一个正则化项，变为

$$\|X\theta - y\|^{2} + \|\tau\theta\|^{2} \tag{6-16}$$

其中，我们定义，于是：

$$\theta(\alpha) = (X^{T}X + \alpha I)^{-1}X^{T}y \tag{6-17}$$

上式中，*I*是单位矩阵。

随着α的增大，θ(α)各元素的绝对值均趋于不断变小，它们相对于正确值θ_i的偏差也越来越大。α趋于无穷大时，θ(α)趋于 0。其中，θ(α)随α的改变而变化的轨迹，就称为岭迹。实际计算中可选的α值较多，选取岭迹图中较为稳定的值，即可确定α值。

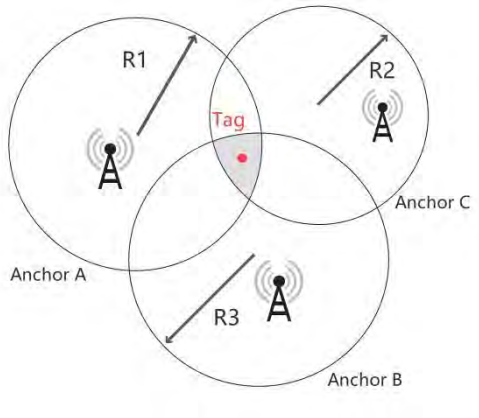

图 6-1 近似坐标求解示意图

6.2 “异常数据”定位模型

6.2.1 特征提取

“异常数据”特征提取方法与“正常数据”特征提取方法类似，但与“正常数据”特征提取不同之处在于“异常数据”特征提取没有将靶点到 4 个球面的距离之和作为目标函数，同时由于 4 个锚点中存在锚点信号被干扰，因此没有使用最小二乘法去解析靶点的三维坐标作为粒子群的初始位置，故粒子群初始位置将随机产生。最后将求解的 4 个近似坐标作为后续预测算法的特征数据。

6.2.2 预测算法

“异常数据”预测算法同样也是使用岭回归算法。

6.3 模型评估与对比

6.3.1 模型评估

对于训练集训练结果与测试集测试结果，本文利用式 6-18~6-22 进行各维坐标平均误差计算：

$$\delta_x=\frac{\sum_{i=1}^{n} X_i-X_0}{n} \tag{6-18}$$

$$\delta_y=\frac{\sum_{i=1}^{n} Y_i-Y_0}{n} \tag{6-19}$$

$$\delta_z=\frac{\sum_{i=1}^{n} Z_i-Z_0}{n} \tag{6-20}$$

$$\delta_{xy}=\frac{\sum_{i=1}^{n} \sqrt{(X_i-X_0)^2+(Y_i-Y_0)^2}}{n} \tag{6-21}$$

$$\delta_{xyz}=\frac{\sum_{i=1}^{n} \sqrt{(X_i-X_0)^2+(Y_i-Y_0)^2+(Z_i-Z_0)^2}}{n} \tag{6-22}$$

其中δ表示所有点各维坐标预测值与真实值之间的平均误差，X_0、Y_0、Z_0表示该点真实坐标。

训练集、测试集平均误差计算结果如表 6-1、6-2 所示：

表 6-1 训练集训练结果(mm)

预测模型	X 轴平均误差	Y 轴平均误差	Z 轴平均误差	(x,y)二维平均误差	三维平均误差
“正常数据”模型	36.22	28.83	146.94	51.45	163.46
“异常数据”模型	127.65	127.17	366.18	188.30	429.61

表 6-2 测试集测试结果(mm)

预测模型	X 轴平均误差	Y 轴平均误差	Z 轴平均误差	(x,y)二维平均误差	三维平均误差
“正常数据”模型	36.22	30.00	159.88	52.44	175.71
“异常数据”模型	135.44	128.09	373.44	193.83	437.50

由上表可得，训练集与测试集“正常数据”模型中 X 轴、Y 轴的平均误差基本保持在 25mm 左右，“异常数据”模型中 X 轴、Y 轴的平均误差基本保持在 70mm 左右，表明该预测模型对于 X 轴、Y 轴的坐标值估计较为准确；而对于 Z 轴的坐标值估计，“正常数据”模型的平均误差在 130mm 左右，“异常数据”模型的平均误差在 190mm 左右，表明该模型对于 Z 轴坐标的预测准确率偏低。对于三维坐标的误差，本文作出如下图 6-2、6-3 所示三维坐标误差分布图：

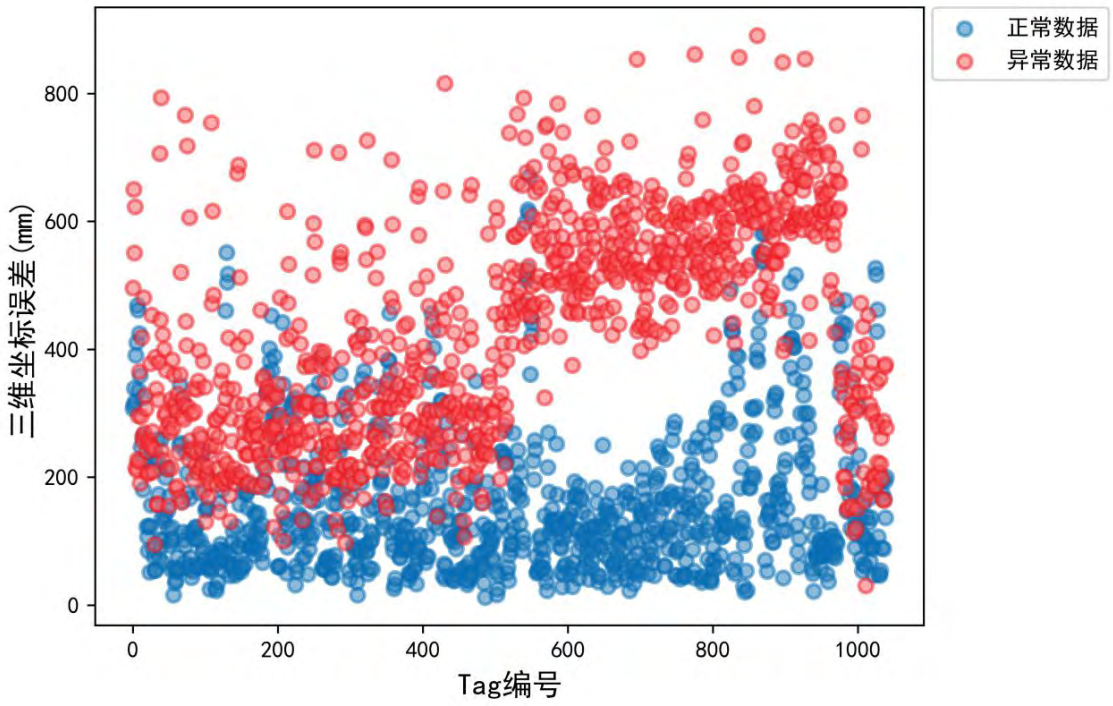

图 6-2 训练集三维坐标误差分布图

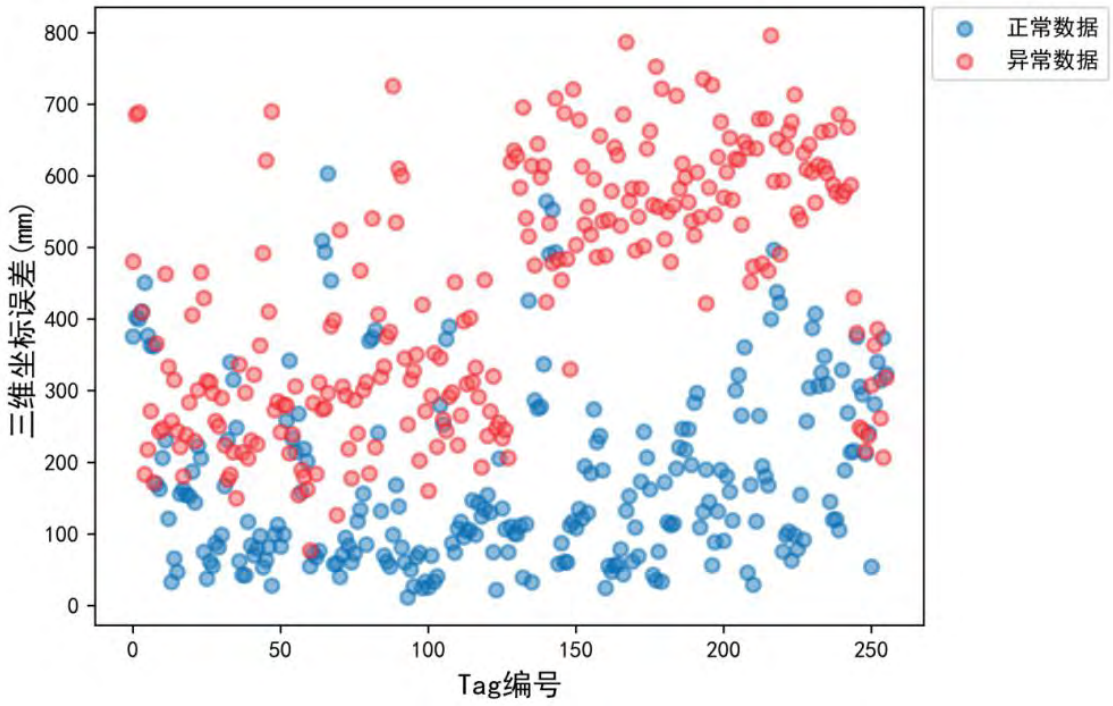

图 6-3 测试集三维坐标误差分布图

由上图可直观发现，训练集与测试集中正常数据的预测误差大部分分布于 200mm 以下，异常数据预测误差基本分布在 200mm-700mm，整体误差效果较为可观，由此可见本预测模型对于三维坐标值的预测具有较高的准确性。

6.3.2 模型对比

使用山岭回归算法直接对原始数据进行定位预测，其测试定位精度如下表6-3所示：

表6-3 原始数据定位测试结果(mm)

预测模型	X轴平均误差	Y轴平均误差	Z轴平均误差	(x,y)二维平均误差	三维平均误差
“正常数据”模型	47.96	47.62	366.72	75.15	381.84
“异常数据”模型	158.22	154.24	378.28	229.85	460.94

从表6-2和表6-3可知，建立数学模型拓展特征数据对定位精度有所提升，其定位精度提升效果如下表6-4所示：

表6-4 拓展数据定位精度提升(%)

预测模型	X轴提升精度	Y轴提升精度	Z轴提升精度	(x,y)二维提升精度	三维提升精度
“正常数据”模型	24.5	37.0	56.4	30.2	53.9
“异常数据”模型	14.4	16.9	1.2	15.7	5.1

6.4 附件数据预测

附件2中10组数据预测结果如下表6-5所示：

表6-5 附件2预测结果(mm)

序号	d_0	d_1	d_2	d_3	x	y	z
1	1320	3950	4540	5760	1190.17	690.15	1097.40
2	3580	2580	4610	3730	3182.22	1725.93	1109.07
3	2930	2600	4740	4420	2744.79	1182.64	1137.40
4	2740	2720	4670	4790	2458.82	1020.29	1894.33
5	2980	4310	2820	4320	1480.84	2528.10	1822.86
6	2230	3230	4910	5180	2066.41	726.91	1442.58
7	4520	1990	5600	3360	4208.50	1631.48	1397.67
8	2480	3530	4180	5070	1779.46	1239.60	1495.28
9	4220	2510	4670	3940	3586.68	2005.15	1543.76
10	5150	2120	5800	2770	4674.58	2099.85	1465.22

7 问题三不同场景应用

本文定位算法是根据求解近似坐标来拓展数据特征，建立定位模型。理论上，该模型不依赖于原始数据，因此适用于不同场景应用。

根据题目要求，本文使用问题二中的模型对附件 3 中的 10 组数据进行预测。预测结果如下表 7-1 所示：

表 7-1 预测结果(mm)

序号	d_0	d_1	d_2	d_3	x	y	z
1	4220	2580	3730	1450	3652.53	2210.99	1197.44
2	4500	1940	4420	1460	4201.33	1711.03	1081.03
3	3550	2510	3410	2140	3162.59	1735.10	1207.64
4	3300	3130	2900	2790	2577.20	1846.93	1539.67
5	720	4520	3050	5380	527.02	69.70	1396.36
6	5100	2200	4970	800	4551.81	2055.80	1446.32
7	2900	3210	3140	2890	2468.49	1541.39	1421.25
8	2380	3530	2320	3760	1771.69	1373.31	1482.01
9	2150	3220	3140	3640	1971.37	796.60	1450.66
10	1620	3950	2580	4440	1208.17	811.73	1489.45

8 问题四分类模型

8.1 特征提取

仅根据题目提供的靶点到四个锚点的距离很难判别其采集的数据是正常数据（锚点信号被干扰下采集的）还是异常数据（锚点信号无干扰下采集的）。为了解决此问题,本文首先对数据进行特征拓展，根据靶点到 4 个锚点的距离使用最小二乘法解的近似坐标，然后将求得的近似坐标反算与四个锚点的估计距离得到$d_0^{'}$，$d_1^{'}$，$d_2^{'}$，$d_3^{'}$，然后计算估计距离$d_0^{'}$，$d_1^{'}$，$d_2^{'}$，$d_3^{'}$与d_0，d_1，d_2，d_3的差值得到$\Delta\mathrm{d}_0$，$\Delta\mathrm{d}_1$，$\Delta\mathrm{d}_2$，$\Delta\mathrm{d}_3$。最后将距离差值和$\sum_{i=0}^{3}\Delta d_i$作为 1 个特征数据。最后计算出的特征数据有 5 维，其顺序为d_0，d_1，d_2，d_3，$\sum_{i=0}^{3}\Delta d_i$。

8.2 分类算法

8.2.1 极限学习机(ELM)算法

如图 8-1 所示，典型的单隐含层前馈神经网络结构由输入层、隐含层和输出层组成，三层均有神经元组成，输入层与隐含层、隐含层与输出层神经元间全连接。其中，若输入层有 n 个神经元，隐含层有 1 个神经元，输出层有 m 个神经元，设输入层与隐含层间的连接权值为ω：

$$\omega=\begin{bmatrix}\omega_{11} & \omega_{12} & \cdots & \omega_{1n}\\ \omega_{21} & \omega_{22} & \cdots & \omega_{2n}\\ \vdots & \vdots & & \vdots\\ \omega_{l1} & \omega_{l2} & \cdots & \omega_{ln}\end{bmatrix} \tag{8-1}$$

其中，ω_{ij}表示输入层第 i 个神经元与隐含层第 j 个神经元间的连接权值。

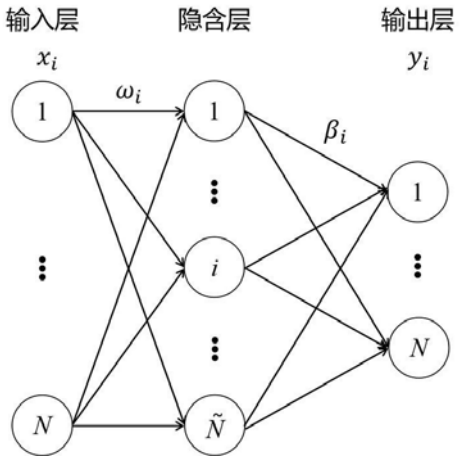

图 8-1 单隐层神经网络

设β为隐含层与输出层间的连接权值：

$$\beta = \begin{bmatrix} \beta_{11} & \beta_{12} & \cdots & \beta_{1m} \\ \beta_{21} & \beta_{22} & \cdots & \beta_{2m} \\ \vdots & \vdots & & \vdots \\ \beta_{l1} & \beta_{l2} & \cdots & \beta_{lm} \end{bmatrix} \tag{8-2}$$

其中，β_{jk}表示隐含层第 j 个神经元与输出层第 k 个神经元之间的连接权值。

设隐含层神经元的阈值 b 为：

$$b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_l \end{bmatrix} \tag{8-3}$$

设具有 Q 个样本的训练集输入矩阵 X 和输出矩阵 Y 分别为：

$$X = \begin{bmatrix} X_{11} & X_{12} & \cdots & X_{1Q} \\ X_{21} & X_{22} & \cdots & X_{2Q} \\ \vdots & \vdots & & \vdots \\ X_{n1} & X_{n2} & \cdots & X_{nQ} \end{bmatrix} \tag{8-4}$$

$$Y = \begin{bmatrix} Y_{11} & Y_{12} & \cdots & Y_{1Q} \\ Y_{21} & Y_{22} & \cdots & Y_{2Q} \\ \vdots & \vdots & & \vdots \\ Y_{m1} & Y_{m2} & \cdots & Y_{mQ} \end{bmatrix} \tag{8-5}$$

设隐含层神经元的激活函数为 g(x)，则由图 8-1 可得，网络的输出 T 为：

$$T = \left[t_1, t_2, \cdots, t_Q\right]_{m*Q} \tag{8-6}$$

设其中 t_j 为：

$$t_j = \left[t_{1j}, t_{2j}, \cdots, t_{mj}\right] = \begin{bmatrix} \sum_{i=1}^{t} \beta_{i1}\, g(\omega_i x_i + b_i) \\ \sum_{i=1}^{t} \beta_{i2}\, g(\omega_i x_i + b_i) \\ \vdots \\ \sum_{i=1}^{t} \beta_{im}\, g(\omega_i x_i + b_i) \end{bmatrix}, \quad (j = 1,2,\cdots,Q) \tag{8-7}$$

上式可表示为:

$$H\beta = T' \tag{8-8}$$

其中T'为矩阵 T 的转置；H 称为神经网络的隐含层输出矩阵，具体形式如下:

$$H(\omega_1, \cdots, \omega_i, b_1, \cdots, b_l, x_1, \cdots, x_Q) = \begin{bmatrix} g(\omega_1 x_1 + b_1) & \cdots & g(\omega_l x_1 + b_l) \\ \vdots & \ddots & \vdots \\ g(\omega_1 x_Q + b_1) & \cdots & g(\omega_l x_Q + b_l) \end{bmatrix} \tag{8-9}$$

ELM 的学习算法主要有以下几个步骤：

(1)确定隐含层神经元个数，设定输入层与隐含层间的连接权值ω和隐含层神经元的偏置 b；

(2)选择一个无限可微的函数作为隐含层神经元的激活函数，进而计算隐含层输出矩阵 H；

(3)计算输出层权值。

8.2.2 麻雀搜索算法(SSA)优化的极限学习机

麻雀搜索算法(SSA)是一种基于麻雀种群的觅食和反捕食行为的一种新型智能优化算法。由前文可知，ELM 的初始权值和阈值都是随机产生。本文利用 SSA 对初始权值和阈值进行优化。适应度函数设计为训练集的错误率与测试集的错误率的和，以期望使训练得到的网络在测试集和训练集上均有较好的结果：

$$\text{fitness} = \text{agrmin}(\text{TrainErrorRate} + \text{TestErrorRate}) \tag{8-10}$$

基于麻雀搜索算法(SSA)优化的 ELM 算法流程图如下：

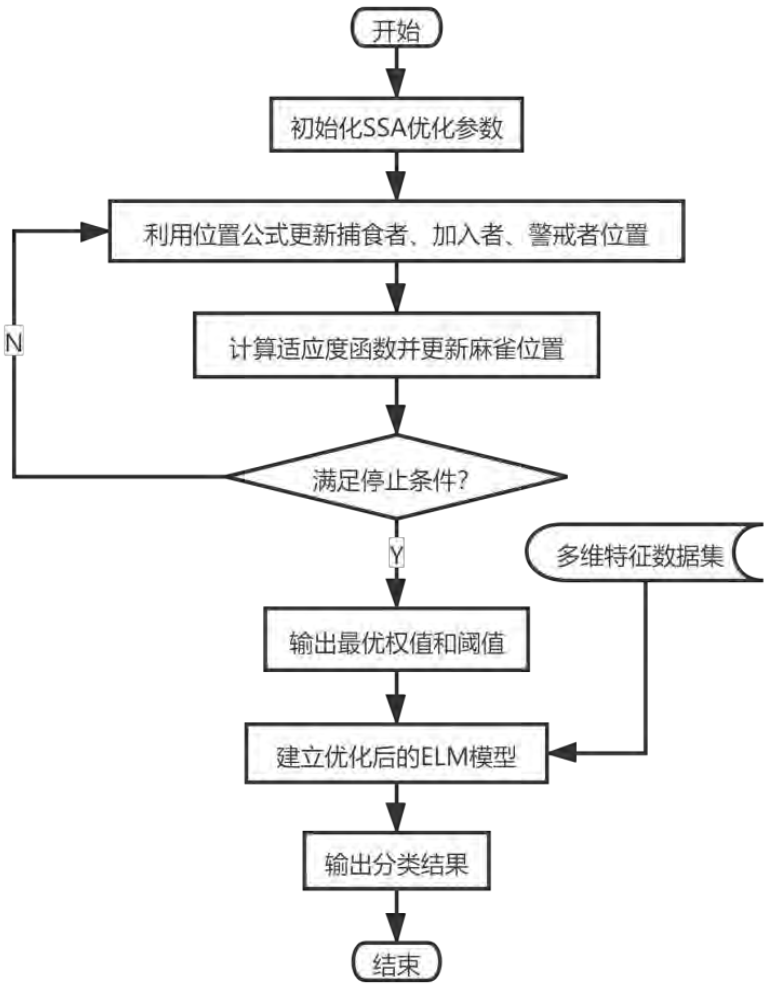

图 8-2 基于麻雀搜索算法(SSA)优化的 ELM 算法流程图

8.3 模型评估与对比

8.3.1 机器学习模型评估指标

在机器学习中，混淆矩阵是评价模型各方面精度的可视化工具，其通常用有监督学习。在分类任务中，各指标的计算基础都来自于对正负样本的分类结果，用混淆矩阵表示为：

真实值	预测值	
	正	负
正	TP	FN
负	FP	TN

图 8-3 混淆矩阵

其中，准确率为 $ACC = \frac{TP+TN}{TP+TN+FP+FN}$，其表示所有分类正确的样本占全部样本的比例；精确率为 $P = \frac{TP}{TP+FP}$，其表示预测是正例的结果中，确实是正例的比例；召回率为 $R = \frac{TP}{TP+FN}$，其表示所有正例的样本中，被找出的比例。

8.3.2 基于传统极限学习机的分类模型

经 5.5 划分后训练集总数为 2080 组，其中“正常”数据为 1040 组，“异常”数据为 1040 组。测试集总数为 512 组数据，其中“正常”数据为 256 组，“异常”数据为 256 组。使用基于极限学习机的分类模型的评估指标如下表 8-1 所示：

表 8-1 训练集及测试集模型评估指标

	准确率(Accuracy)	精确率(Precision)	召回率(Recall)
训练集	77.45%	70.98%	92.88%
测试集	77.54%	71.56%	91.41%

由表可知，用传统极限学习机算法作为分类模型的精度较差，基本无法解决区分正常和异常数据的问题，结合下图 8-4、8-5 所示训练集与测试集的混淆矩阵，虽然传统 ELM 分类模型对于预测正常数据的精度在 90%左右，但是对异常数据的预测性能较差，仅在 60%左右，由此判断出使用传统 ELM 分类模型不具有较高精度。

图 8-4 ELM 训练集混淆矩阵

图 8-5 ELM 测试集混淆矩阵

8.3.3 麻雀搜索算法优化极限学习机的分类模型

由 8.3.2 的模型评估可知，传统 ELM 分类预测的结果可信度较低，针对问题四提出的分辨出正常数据与异常数据，本文对 ELM 进一步优化，利用麻雀寻优算法，找出最优的权重矩阵、偏置矩阵，尽可能提升极限学习机的学习能力和分类有效性。为保证对比实验的准确性，SSA+ELM 分类模型使用的训练集和测试集均与传统 ELM 保持一致。

训练集及测试集准确率、精确率及召回率如下表 8-1 所示：

表 8-2 训练集及测试集模型评估指标

	准确率(Accuracy)	精确率(Precision)	召回率(Recall)
训练集	92.64%	88.13%	98.56%
测试集	94.53%	91.01%	98.83%

通过表 8-1 计算得出，经 SSA 算法对 ELM 参数进行寻优后得到的分类模型效果比传统 ELM 分类模型在该场景下训练集和测试集的准确率分别提升了 19.61%和 21.91%，预测能力显著增强。

图 8-6 SSA+ELM 训练集混淆矩阵

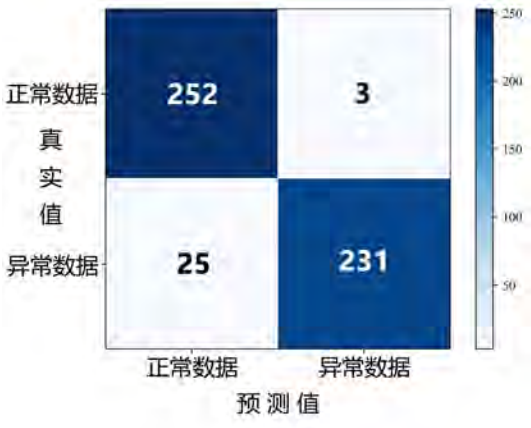

图 8-7 SSA+ELM 测试集混淆矩阵

依图 8-6 和图 8-7 可计算得出，SSA+ELM 分类模型的训练集和测试集对正常数据的预测精度分别为 98.56%和 98.83%，相比传统 ELM 分别提升了 6.12%和 8.12%。SSA+ELM 分类模型对异常数据的训练集、测试集预测精度分别为 86.73%和 90.23%,相比于 ELM 分别提升了 39.84%和 41.72%。

通过实验数据可以看出，传统 ELM 经 SSA 寻优后的分类模型的预测结果有效性大幅提高，同时由于本文所取测试集的锚点在空间分布均匀，具有代表性，因此可以认为该模型能够解决在场景 1 下对正常数据和异常数据的识别问题。

8.4 附件数据预测结果

根据题目要求，本文使用分类模型对附件 4 中的 10 组数据进行分类预测。其中“0”代表“异常数据”，“1”代表“正常数据”。预测结果如下表 8-3 所示：

表 8-3 附件 4 分类预测结果

序号	d_0	d_1	d_2	d_3	类别
1	2940	4290	2840	4190	1
2	5240	5360	2040	2940	0
3	4800	2610	4750	2550	1
4	5010	4120	3810	2020	0
5	2840	4490	2860	4190	0
6	5010	5320	1990	2930	1
7	5050	3740	3710	2070	0
8	5050	4110	3710	2110	0
9	4840	2600	4960	2700	1
10	2740	2720	4670	4790	0

9 问题五运动轨迹定位

本题是结合上述四个问题的综合性问题，该题需要结合使用上文所建立的特征数据拓展模型、预测模型及分类模型，具体解题思路如下：

（1）对附件 5 中原始数据的 TXT 文件进行处理，转化成一个包含锚点距离的四维 CSV 数据；

（2）通过最小二乘法与粒子群算法建立的特征数据拓展模型对上述数据进行特征提取，制作特征数据集；

（3）利用问题四中的分类模型判断各数据点是否受到干扰，将数据初步分为“正常数据”与“异常数据”，分别通过对应的预测模型预测各数据点坐标；

（4）不论“正常数据”还是“异常数据”，都需要检验坐标预测是否精确，即分为“定位正常点”与“定位异常点”，由题目所述，运动轨迹数据为一段时间内连续采集的多组数据，因此可以推断若出现“定位异常点”，其坐标与“定位正常点”距离较大，因此采用 Kmean 聚类算法对其分类成“定位正常点”和“定位异常点”；

（5）对判断为预测不精确，即“定位异常点”进行数据修正，修正的方式为线性插值；

（6）对轨迹点坐标进行修正后，绘制出轨迹预测曲线。

具体流程如图 9-1 所示：

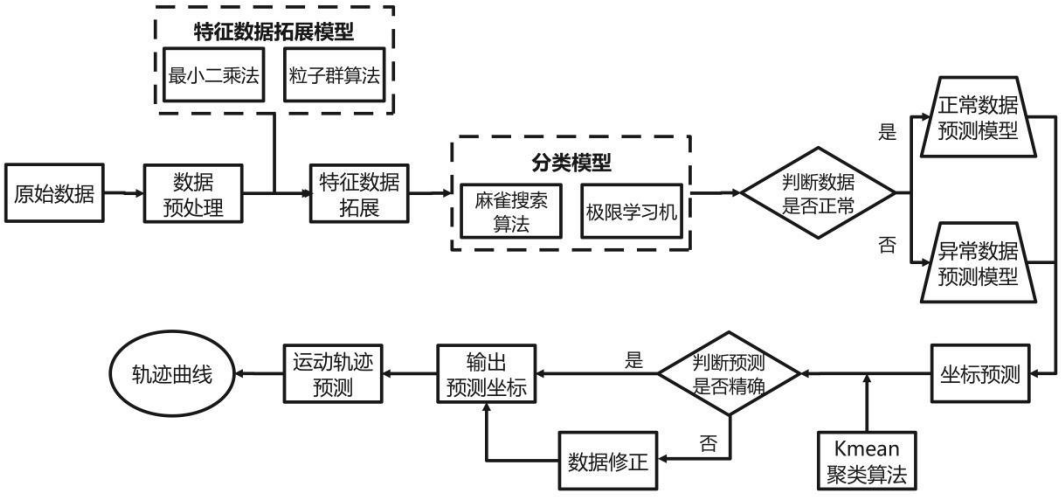

图 9-1 运动轨迹预测流程图

9.1 数据分类结果

由于预测模型分为“正常数据”预测模型和“异常数据”预测模型，故本题本文首先使用问题四中的分类模型对轨迹文件中 539 个点的数据进行分类，得出“正常数据”412 组，“异常数据”127 组如下表 9-1 所示：

表 9-1 轨迹文件数据分类结果

正常数据					异常数据				
序号	d_0	d_1	d_2	d_3	序号	d_0	d_1	d_2	d_3
1	810	4650	4570	6520	1	1510	4660	3870	5680
2	810	4650	4570	6510	2	4460	5180	2020	3410
3	810	4640	4580	6510	3	4440	5150	2050	3490
4	810	4640	4580	6520	4	4440	5120	2080	3510
5	800	4640	4610	6530	5	4450	5080	2120	3510
6	790	4640	4600	6530	6	4450	5040	2150	3520
7	790	4630	4610	6530	7	4430	5010	2180	3520
8	780	4630	4610	6530	8	4410	4990	2220	3520
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
409	6380	4460	4500	850	124	5870	3890	4780	1350
410	6390	4480	4500	850	125	5890	3940	4690	1330
411	6390	4480	4500	840	126	5900	3980	4620	1310
412	6390	4510	4500	830	127	5910	4000	4560	1310

9.2 轨迹预测结果

将上述的轨迹分类结果分别送入“正常数据”预测模型及“异常数据”预测模型进行轨迹预测，并分别从 x，y，z 轴、xy 平面和三维空间坐标反映对动态轨迹定位的结果。其中三维空间轨迹图如图 9-2 所示，x，y，z 轴、xy 平面如图 9-3 所示。由图可知，x 坐标随着点数的增加基本呈线性增加，y 坐标随着点数的增加基本呈线性“先增后减再增”分布，而 z 坐标基本在 1250mm-1750mm 间呈明显浮动。

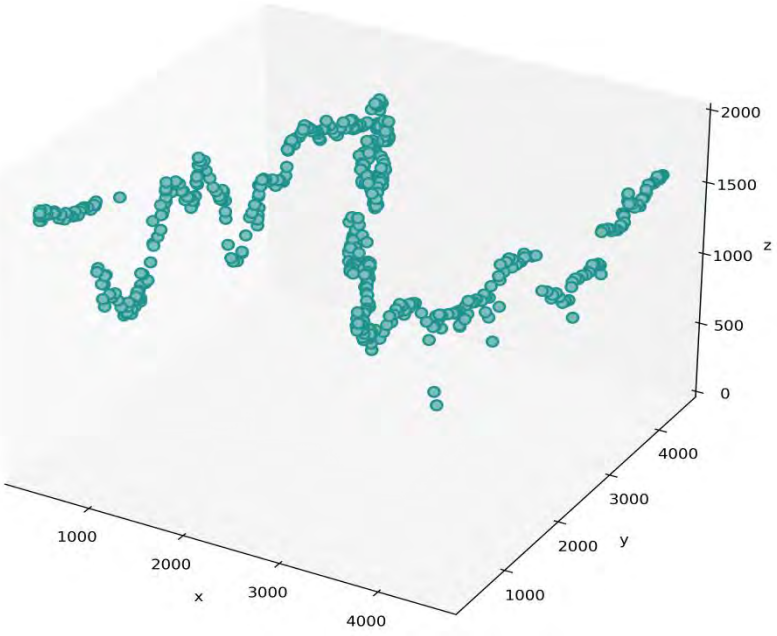

图 9-2 三维坐标轨迹图

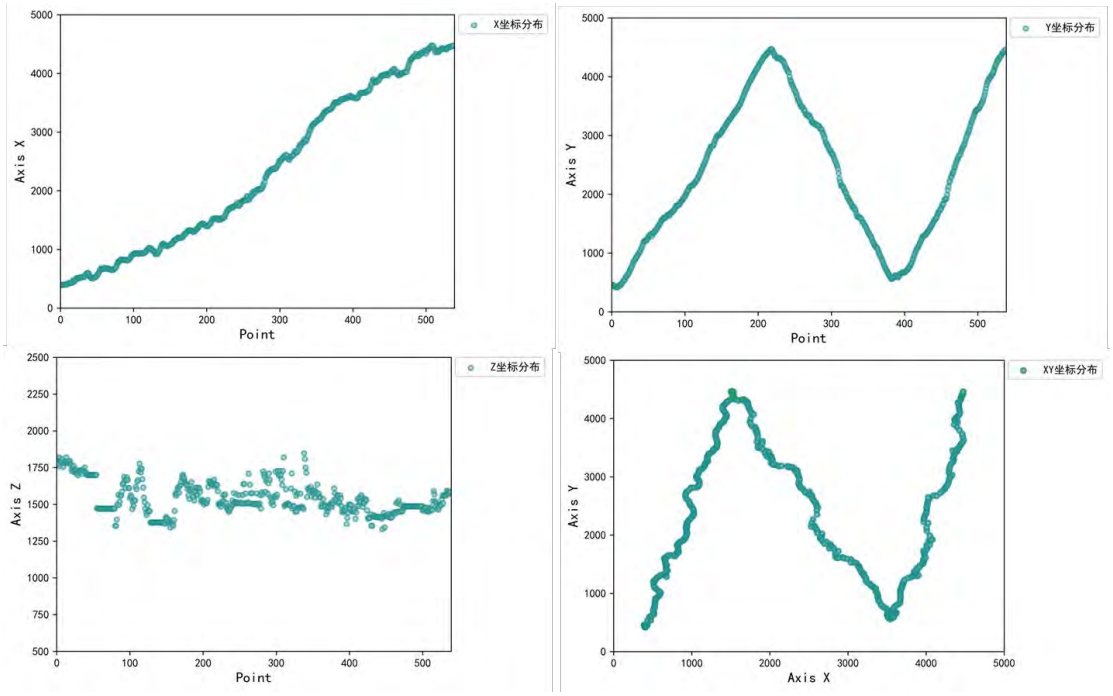

图 9-3 一维、二维坐标分布图

9.3 轨迹修正

结合图 9-2、9-3 可以发现，x、y 坐标分布基本满足线性关系，整体轨迹浮动较小，而 z 坐标存在明显浮动，因此本文在原轨迹基础上对 z 坐标进行 K-mean 聚类并分为“定位正常点”与“定位异常点”坐标两类，并针对“定位异常点”的 z 坐标进行线性插值修正，修正结果如图 9-4 所示，结合一、二、三维图形及轨迹修正结果，推断该轨迹为高度在 1500mm 左右的“N”形线性运动。

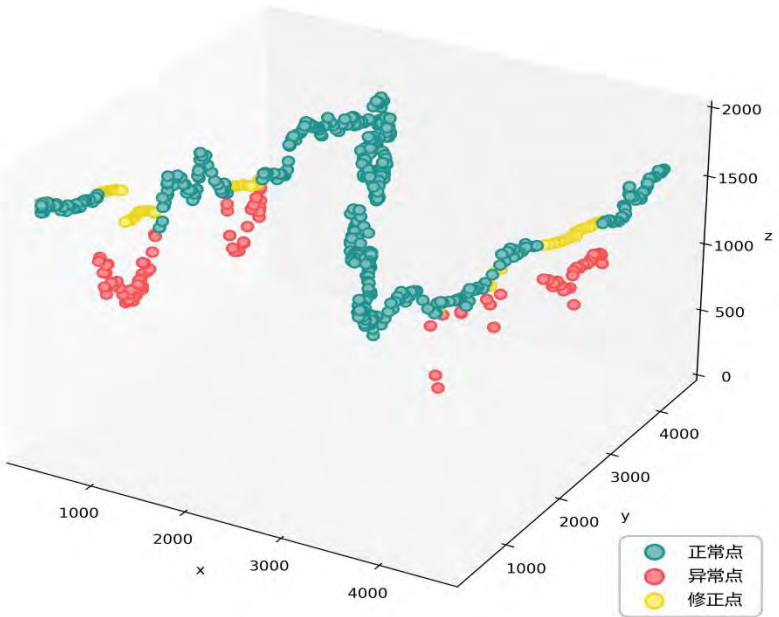

图 9-4 修正后的三维轨迹图

10 模型评价与改进

10.1 模型的优点

1. 在数据预处理方面，本文主要创新点在剔除标准的选择，即利用靶点真实坐标和锚点坐标的理论值与测量值之差作为参数，同时结合 3sigma 准则与误差直方图分析进行“无用”数据的多次剔除，保证数据清洗的可靠性。在数据筛选环节，本文利用 K-mean 聚类选取距离簇质心最近的点，保证最终保留的数据大部分具有代表性。

2. 对于预测数集的选取，本文对每一高度层中的采集点作均匀选取，保证训练集及测试集数据在空间均匀分布，减少数据因空间分布而出现的偶然误差，增加训练及测试结果的准确性。

3. 对于预测模型，本文的主要创新点在于结合最小二乘法及改进粒子群算法对原数据进行多维特征参数的寻优提取。若仅使用原始靶点到 4 个锚点的距离作为特征数据进行建模，数据采集的场景会影响到模型的性能。如果将模型运用到其他场景中，模型可能会失效。而该预测模型适用于不同场景下的定位。

4. 对于分类模型，本文使用以 ELM 为主框架的分类算法，ELM 算法学习速度快、泛化性能好，并且本文在原算法基础上加入 SSA 算法进行优化输出最佳初始权值和阈值，保证分类模型的不容易陷入局部最优。

5. 对于问题五的轨迹预测，本文的在原预测结果的基础上通过聚类选出“定位正常点”和“定位异常点”，从而对原轨迹进一步修正，使最终轨迹更趋于平滑。

10.2 模型的缺点

1. 本文使用粒子群寻优算法拓展数据特征，该方法虽具有较高寻优精度，但由于计算公式冗杂且数据集偏多，导致算法寻优时间较长。

2. 本文建立的预测模型虽然对于 X、Y 轴的坐标预测较为准确，但对于 Z 轴坐标的预测误差较大，导致后续的轨迹预测在 Z 轴上有明显浮动。

3. 本文建立的分类模型对于“异常数据”的预测准确率仍有待提高。

10.3 模型改进

1. 对于 Z 轴坐标的预测可单独设计一个优化 Z 轴坐标的预测模型，以此减小 Z 轴坐标的预测误差。

2. 在预测和分类时，可以利用主成分分析法等特征降维方法对多维特征进一步简化并选取最优特征参数。

3. 可利用其他更加优越的寻优算法代替粒子群算法，在减少计算量的同时也可减小最佳结果陷入局部最优的概率。

4. 本文在问题五中只对 Z 轴进行了坐标修正，但 X、Y 轴坐标由于“异常数据”的影响也存在一定程度的误差，故可对 X、Y、Z 三轴坐标同时进行线性插值修正，使整体轨迹相对于本文结果更加趋于平滑。

参考文献

[1] Kok M, Hol J D, Schon T B. Indoor Positioning Using Ultrawideband and Inertial Measurements[J]. IEEE Transactions on Vehicular Technology, 2015, 64(4): 1293—1303.

[2] Arias-De-Reyna E. A Cooperative Localization Algorithm for UWB Indoor Sensor Networks[J]. Wireless Personal Communications, 2013, 72(1):85—99.

[3] 缪希仁,范建威等，基站异常情况下基于改进极限学习机的超宽带室内定位方法，传感技术学报，2020，33（10）：1—10.

[4] GIGI24，到底什么是UWB超宽带技术？有哪些应用场景？，http://m.elecfans.com/article/1643155.html，2021年10月14日.

[5] 奥莉说，基于UWB的应用场景有哪些？UWB解决方案应用案例！，http://m.elecfans.com/article/1116045.html，2021年10月14日.

[6] 云酷科技，全面解析UWB及其应用场景，http://www.qianjia.com/zhike/html/2020-02/11_19601.html，2021年10月14日.

[7] 夏禹, 张效艇, 王嵘. TOA 定位算法的坐标解析优化方法综述[J]. 单片机与嵌入式系统应用, 2019, 019(012):15-18,22.

[8] Jack 旭，智能优化算法：麻雀搜索算法，https://blog.csdn.net/u011835903/article/details/108830958，2021 年 10 月 15 日.

[9] 杜如东. 基于 TOA 定位算法在三维空间的研究与改进[D].上海师范大学,2018.

[10] 刘泽恒,杨力,黄俊,齐志,吴建辉,时龙兴,张阳.基于无线通信基站三维定位问题的求解模型[J].数学的实践与认识,2017,47(14):183-191.

附录 A 数据预处理程序

(1) TXT 转 CSV 程序

```
import os
import csv
import re
path = "./原始数据/正常数据" #文件夹目录
path_csv ="./原始数据/正常数据 csv"
files= os.listdir(path) #得到文件夹下的所有文件名称
number=0

for file in files: #遍历文件夹
    position = path+'/'+ file #构造绝对路径，"\\"，其中一个'\'为转义符
    print(file)
    file_=file.split('.')
    csv_name=file_.copy()
    csv_name[2]='csv'
    csv_name = str.join(".",csv_name)
    number=number+1
    position_csv=path_csv+'/'+ csv_name

    with open(position, "r",encoding='utf-8') as f:    #打开文件
        lines = f.readlines()
    txt=[]

    if (int(file_[0])-1)//81==0:
        a_=int(file_[0])-0
        z_=str(88)
    if (int(file_[0])-1)//81==1:
        a_=int(file_[0])-81
        z_=str(130)
    if (int(file_[0])-1)//81==2:
        a_=int(file_[0])-162
        z_=str(170)
    if (int(file_[0])-1)//81==3:
        a_=int(file_[0])-243
        z_=str(200)
    a=int(a_-1)%9
    b=int(a_-1)//9
    print('a_:',a_,'z_:',z_)
    print('a:',a,'b:',b)
    f_=open(position_csv,"w",encoding='utf-8',newline='')
    csv_writer = csv.writer(f_)

```

```
    for line in lines:

        if "RR" in line:
            line_=line.split(':')
            txt.append(line_[5])
            if line_[4]=='3':
                if b%2==0:
                    x_=str((b+1)*50)
                    y_=str((a+1)*50)
                if b%2!=0:
                    x_=str((b+1)*50)
                    y_=str((9-a)*50)
                txt.insert(4,x_)
                txt.insert(5,y_)
                txt.insert(6,z_)
                txt.append('1')
                csv_writer.writerow(txt)
                txt=[]
print(x_,y_)
f_.close()
```

(2)Kmean 聚类数据处理程序

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.utils import shuffle
import numpy as np
import pandas as pd
import os
import csv
import re

path_csv ="./选取数据/训练集/正常数据"
position_csv = './聚类数据/训练正常数据集 2.csv'
f_=open(position_csv,"w",encoding='utf-8',newline='')
csv_writer = csv.writer(f_)
files= os.listdir(path_csv)
for file in files: #遍历文件夹
    url = path_csv+'/'+ file #构造绝对路径，"\\"，其中一个'\'为转义符
    print(file)
    data = pd. read_csv(url, sep=',',header=None)
    data=np.array(data)
    X_data=data[:,7:12]
    data_U=data[:,:7].tolist()

    # kmeans clustering
    kmeans= KMeans( 4, random_state=0)
    kmeans.fit(X_data)  # 训练模型
    labels= kmeans.predict(X_data)  # 预测分类
    X_list=X_data.tolist()
    print(len(X_data))
    for i in range(len(X_data)):
        data_U[i].append(labels[i])
    X_list_0=[]
    X_list_1=[]
    X_list_2=[]
    X_list_3=[]

    for i in range(len(data_U)):
        if data_U[i][7]==0:
            X_list_0.append(data_U[i])
        if data_U[i][7]==1:
            X_list_1.append(data_U[i])
        if data_U[i][7]==2:
            X_list_2.append(data_U[i])
        if data_U[i][7]==3:
```

```
                X_list_3.append(data_U[i])

    X_train_0=X_list_0[0]
    del X_train_0[7]
    csv_writer.writerow(X_train_0)
    print(X_train_0)

    X_train_1=X_list_1[0]
    del X_train_1[7]
    csv_writer.writerow(X_train_1)
    print(X_train_1)

    X_train_2=X_list_2[0]
    del X_train_2[7]
    csv_writer.writerow(X_train_2)
    print(X_train_2)

    X_train_3=X_list_3[0]
    del X_train_3[7]
    csv_writer.writerow(X_train_3)
    print(X_train_3)

f_.close()
```

附录 B 数据提取程序

(1)正常数据提取

```
import numpy as np
import random
import math
import pandas as pd
import csv

def fit_fun(X,A0,A1,A2,A3,D,label):  # 适应函数
    dis1=abs(math.sqrt((X[0]-A0[0])**2+(X[1]-A0[1])**2+(X[2]-A0[2])**2)-D[0])
    dis2=abs(math.sqrt((X[0]-A1[0])**2+(X[1]-A1[1])**2+(X[2]-A1[2])**2)-D[1])
    dis3=abs(math.sqrt((X[0]-A2[0])**2+(X[1]-A2[1])**2+(X[2]-A2[2])**2)-D[2])
    dis4=abs(math.sqrt((X[0]-A3[0])**2+(X[1]-A3[1])**2+(X[2]-A3[2])**2)-D[3])

    disa=dis1+dis2+dis3
    disb=dis1+dis2+dis4
    disc=dis1+dis3+dis4
    disd=dis2+dis3+dis4
    if label==0:
        return disa
    if label==1:
        return disb
    if label==2:
        return disc
    if label==3:
        return disd

class Particle:
    # 初始化
    def __init__(self, x_max, max_vel, dim, A0, A1, A2, A3, D, point_init,label):
        self.A0=A0
        self.A1=A1
        self.A2=A2
        self.A3=A3
        self.D=D
        self.label=label
        self.point=point_init.copy()
        self.__pos = self.point   # 粒子的位置
        # self.__pos = [random.uniform(0, x_max) for i in range(dim)]  # 粒子的位置
        self.__vel = [random.uniform(-max_vel, max_vel) for i in range(dim)]  # 粒子的速度
        self.__bestPos = [0.0 for i in range(dim)]  # 粒子最好的位置
        self.__fitnessValue = fit_fun(self.__pos,self.A0,self.A1,self.A2,self.A3,self.D,self.label)
```

```
# 适应度函数值

    def set_pos(self, i, value):
        self.__pos[i] = value

    def get_pos(self):
        return self.__pos

    def set_best_pos(self, i, value):
        self.__bestPos[i] = value

    def get_best_pos(self):
        return self.__bestPos

    def set_vel(self, i, value):
        self.__vel[i] = value

    def get_vel(self):
        return self.__vel

    def set_fitness_value(self, value):
        self.__fitnessValue = value

    def get_fitness_value(self):
        return self.__fitnessValue

'''''
最小二乘法估计 tag 位置
'''

def inin_point(A0,A1,A2,A3,D):
    A=[[2*(A0[0]-A3[0]),2*(A0[1]-A3[1]),2*(A0[2]-A3[2])],
       [2*(A1[0]-A3[0]),2*(A1[1]-A3[1]),2*(A1[2]-A3[2])],
       [2*(A2[0]-A3[0]),2*(A2[1]-A3[1]),2*(A2[2]-A3[2])]]
    A=np.mat(A)
    # print(A)
    B=[[(A0[0]**2+A0[1]**2+A0[2]**2)-(A3[0]**2+A3[1]**2+A3[2]**2)-((D[0])**2-(D[3])**2)],
       [(A1[0]**2+A1[1]**2+A1[2]**2)-(A3[0]**2+A3[1]**2+A3[2]**2)-((D[1])**2-(D[3])**2)],
       [(A2[0]**2+A2[1]**2+A2[2]**2)-(A3[0]**2+A3[1]**2+A3[2]**2)-((D[2])**2-(D[3])**2)]]
    B=np.mat(B)
    # print(B)
    x=((A.T*A).I)*(A.T)*B
    x=x.tolist()
    point_init=[]
```

```
85.         for i in x:
86.             point_init.append(i[0])
87.         # print(point_init,type(point_init))
88.         return point_init
89.
90.     class PSO:
91.         def __init__(self, dim, size, iter_num, x_max, max_vel, A0, A1, A2, A3, D, point_init,label,be
    st_fitness_value=float('Inf'), C1=2, C2=2, W=1):
92.             self.A0=A0
93.             self.A1=A1
94.             self.A2=A2
95.             self.A3=A3
96.             self.D=D
97.             self.label=label
98.             self.point=point_init
99.             self.C1 = C1
100.            self.C2 = C2
101.            self.W = W
102.            self.dim = dim  # 粒子的维度
103.            self.size = size  # 粒子个数
104.            self.iter_num = iter_num  # 迭代次数
105.            self.x_max = x_max
106.            self.max_vel = max_vel  # 粒子最大速度
107.            self.best_fitness_value = best_fitness_value
108.            self.best_position = [0.0 for i in range(dim)]  # 种群最优位置
109.            self.fitness_val_list = []  # 每次迭代最优适应值
110.
111.            # 对种群进行初始化
112.             self.Particle_list = [Particle(self.x_max, self.max_vel, self.dim,self.A0,self.A1,self.A2,
    self.A3,self.D,self.point,self.label) for i in range(self.size)]
113.
114.    def set_bestFitnessValue(self, value):
115.            self.best_fitness_value = value
116.
117.    def get_bestFitnessValue(self):
118.            return self.best_fitness_value
119.
120.    def set_bestPosition(self, i, value):
121.            self.best_position[i] = value
122.
123.    def get_bestPosition(self):
124.            return self.best_position
125.
126.    # 更新速度
```

```
127.     def update_vel(self, part):
128.         for i in range(self.dim):
129.             vel_value = self.W * part.get_vel()[i] + self.C1 * random.random() * (part.get_best_po
    s()[i] - part.get_pos()[i]) \
130.                         + self.C2 * random.random() * (self.get_bestPosition()[i] - part.get_pos()[i])
131.             if vel_value > self.max_vel:
132.                 vel_value = self.max_vel
133.             elif vel_value < -self.max_vel:
134.                 vel_value = -self.max_vel
135.
136.             # elif vel_value < 0:
137.             #     vel_value = 0
138.
139.             part.set_vel(i, vel_value)
140.
141.     # 更新位置
142.     def update_pos(self, part):
143.         for i in range(self.dim):
144.             pos_value = part.get_pos()[i] + part.get_vel()[i]
145.             part.set_pos(i, pos_value)
146.             value = fit_fun(part.get_pos(),self.A0,self.A1,self.A2,self.A3,self.D,self.label)
147.         if value < part.get_fitness_value():
148.             part.set_fitness_value(value)
149.             for i in range(self.dim):
150.                 part.set_best_pos(i, part.get_pos()[i])
151.         if value < self.get_bestFitnessValue():
152.             self.set_bestFitnessValue(value)
153.             for i in range(self.dim):
154.                 self.set_bestPosition(i, part.get_pos()[i])
155.
156.     def update(self):
157.         for i in range(self.iter_num):
158.             for part in self.Particle_list:
159.                 self.update_vel(part)  # 更新速度
160.                 self.update_pos(part)  # 更新位置
161.             self.fitness_val_list.append(self.get_bestFitnessValue())  # 每次迭代完把当前的最优适应度
    存到列表
162.         return self.fitness_val_list, self.get_bestPosition()
163.
164. def run(d):
165.     dim=3
166.     size=20
167.     iter_num=1000
```

```
    x_max=5000
    max_vel=5
    A0=[0,0,1300]
    A1=[5000,0,1700]
    A2=[0,5000,1700]
    A3=[5000,5000,1300]
    point_init=inin_point(A0,A1,A2,A3,d)

    calculate_list=[]
    for i in range(4):
        pso = PSO(dim, size, iter_num, x_max, max_vel,A0,A1,A2,A3,d,point_init,i)
        fit_var_list, best_pos = pso.update()
        calculate_list += best_pos
    return calculate_list+point_init
```

(2) 异常数据特征提取

```
import numpy as np
import random
import math
import pandas as pd
import csv

def fit_fun(X,A0,A1,A2,A3,D,label):  # 适应函数
    dis1=abs(math.sqrt((X[0]-A0[0])**2+(X[1]-A0[1])**2+(X[2]-A0[2])**2)-D[0])
    dis2=abs(math.sqrt((X[0]-A1[0])**2+(X[1]-A1[1])**2+(X[2]-A1[2])**2)-D[1])
    dis3=abs(math.sqrt((X[0]-A2[0])**2+(X[1]-A2[1])**2+(X[2]-A2[2])**2)-D[2])
    dis4=abs(math.sqrt((X[0]-A3[0])**2+(X[1]-A3[1])**2+(X[2]-A3[2])**2)-D[3])

    '''''
    四个基站距离估计 tag 位置，仅用于正常数据预测
    '''
    # dis=dis1+dis2+dis3+dis4
    '''''
    三个基站距离寻优估计 tag 位置
    '''
    disa=dis1+dis2+dis3
    disb=dis1+dis2+dis4
    disc=dis1+dis3+dis4
    disd=dis2+dis3+dis4
    if label==0:
        return disa
    elif label==1:
        return disb
    elif label==2:
        return disc
    elif label==3:
        return disd

class Particle:
    # 初始化
    def __init__(self, x_max, max_vel, dim, A0, A1, A2, A3, D, label):
        self.A0=A0
        self.A1=A1
        self.A2=A2
        self.A3=A3
        self.D=D
        self.label=label
        self.__pos = [random.uniform(0, x_max) for i in range(dim)]  # 粒子的位置
        self.__vel = [random.uniform(-max_vel, max_vel) for i in range(dim)]  # 粒子的速度
```

```
            self.__bestPos = [0.0 for i in range(dim)]  # 粒子最好的位置
            self.__fitnessValue = fit_fun(self.__pos,self.A0,self.A1,self.A2,self.A3,self.D,self.label)
        # 适应度函数值


    def set_pos(self, i, value):
        self.__pos[i] = value

    def get_pos(self):
        return self.__pos

    def set_best_pos(self, i, value):
        self.__bestPos[i] = value

    def get_best_pos(self):
        return self.__bestPos

    def set_vel(self, i, value):
        self.__vel[i] = value

    def get_vel(self):
        return self.__vel

    def set_fitness_value(self, value):
        self.__fitnessValue = value

    def get_fitness_value(self):
        return self.__fitnessValue

'''''
最小二乘法估计 tag 位置
'''

def inin_point(A0,A1,A2,A3,D):
    A=[[2*(A0[0]-A3[0]),2*(A0[1]-A3[1]),2*(A0[2]-A3[2])],
       [2*(A1[0]-A3[0]),2*(A1[1]-A3[1]),2*(A1[2]-A3[2])],
       [2*(A2[0]-A3[0]),2*(A2[1]-A3[1]),2*(A2[2]-A3[2])]]
    A=np.mat(A)
    # print(A)
    B=[[(A0[0]**2+A0[1]**2+A0[2]**2)-(A3[0]**2+A3[1]**2+A3[2]**2)-((D[0])**2-(D[3])**2)],
       [(A1[0]**2+A1[1]**2+A1[2]**2)-(A3[0]**2+A3[1]**2+A3[2]**2)-((D[1])**2-(D[3])**2)],
       [(A2[0]**2+A2[1]**2+A2[2]**2)-(A3[0]**2+A3[1]**2+A3[2]**2)-((D[2])**2-(D[3])**2)]]
    B=np.mat(B)
    # print(B)
```

```
87.         x=((A.T*A).I)*(A.T)*B
88.         x=x.tolist()
89.         point_init=[]
90.         for i in x:
91.             point_init.append(i[0])
92.         print(point_init,type(point_init))
93.         return point_init
94.
95.     class PSO:
96.         def __init__(self, dim, size, iter_num, x_max, max_vel, A0, A1, A2, A3, D,label,best_fitness_v
    alue=float('Inf'), C1=2, C2=2, W=1):
97.             self.A0=A0
98.             self.A1=A1
99.             self.A2=A2
100.            self.A3=A3
101.            self.D=D
102.            self.label=label
103.            self.C1 = C1
104.            self.C2 = C2
105.            self.W = W
106.            self.dim = dim  # 粒子的维度
107.            self.size = size  # 粒子个数
108.            self.iter_num = iter_num  # 迭代次数
109.            self.x_max = x_max
110.            self.max_vel = max_vel  # 粒子最大速度
111.            self.best_fitness_value = best_fitness_value
112.            self.best_position = [0.0 for i in range(dim)]  # 种群最优位置
113.            self.fitness_val_list = []  # 每次迭代最优适应值
114.
115.            # 对种群进行初始化
116.             self.Particle_list = [Particle(self.x_max, self.max_vel, self.dim,self.A0,self.A1,self.A2,
    self.A3,self.D,self.label) for i in range(self.size)]
117.
118.    def set_bestFitnessValue(self, value):
119.            self.best_fitness_value = value
120.
121.    def get_bestFitnessValue(self):
122.            return self.best_fitness_value
123.
124.    def set_bestPosition(self, i, value):
125.            self.best_position[i] = value
126.
127.    def get_bestPosition(self):
128.            return self.best_position
```

```
129.
130.     # 更新速度
131.     def update_vel(self, part):
132.         for i in range(self.dim):
133.             vel_value = self.W * part.get_vel()[i] + self.C1 * random.random() * (part.get_best_po
    s()[i] - part.get_pos()[i]) \
134.                     + self.C2 * random.random() * (self.get_bestPosition()[i] - part.get_pos()[i])
135.             if vel_value > self.max_vel:
136.                 vel_value = self.max_vel
137.             elif vel_value < -self.max_vel:
138.                 vel_value = -self.max_vel
139.
140.             # elif vel_value < 0:
141.             #     vel_value = 0
142.
143.             part.set_vel(i, vel_value)
144.
145.     # 更新位置
146.     def update_pos(self, part):
147.         for i in range(self.dim):
148.             pos_value = part.get_pos()[i] + part.get_vel()[i]
149.             part.set_pos(i, pos_value)
150.             value = fit_fun(part.get_pos(),self.A0,self.A1,self.A2,self.A3,self.D,self.label)
151.         if value < part.get_fitness_value():
152.             part.set_fitness_value(value)
153.             for i in range(self.dim):
154.                 part.set_best_pos(i, part.get_pos()[i])
155.         if value < self.get_bestFitnessValue():
156.             self.set_bestFitnessValue(value)
157.             for i in range(self.dim):
158.                 self.set_bestPosition(i, part.get_pos()[i])
159.
160.     def update(self):
161.         for i in range(self.iter_num):
162.             for part in self.Particle_list:
163.                 self.update_vel(part)  # 更新速度
164.                 self.update_pos(part)  # 更新位置
165.             self.fitness_val_list.append(self.get_bestFitnessValue())  # 每次迭代完把当前的最优适应度
    存到列表
166.         return self.fitness_val_list, self.get_bestPosition()
167.
168. def run(d):
169.     dim=3
```

```
    size=20
    iter_num=1000
    x_max=5000
    max_vel=5
    A0=[0,0,1300]
    A1=[5000,0,1700]
    A2=[0,5000,1700]
    A3=[5000,5000,1300]

    calculate_list=[]
    for i in range(4):
        pso = PSO(dim, size, iter_num, x_max, max_vel,A0,A1,A2,A3,d,i)
        fit_var_list, best_pos = pso.update()
        calculate_list += best_pos
    return calculate_list
```

附录 C 定位程序

(1) 正常数据预测

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import math
from sklearn import datasets, linear_model, discriminant_analysis
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler      #数据预处理
import csv
import joblib


def test_ridge(*data):
    X_train, X_test, y_train, y_test = data
    ridgeRegression = linear_model.Ridge()
    # ridgeRegression = linear_model.LinearRegression()
    rd=ridgeRegression.fit(X_train, y_train)
    joblib.dump(rd, "zc_model.pkl")

    return ridgeRegression.predict(X_test)

#测试不同的α值对预测性能的影响
def test_ridge_alpha(*data):
    X_train, X_test, y_train, y_test = data
    alphas = [0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10, 20, 50, 100, 200, 500, 1000]
    scores = []
    for i, alpha in enumerate(alphas):
        ridgeRegression = linear_model.Ridge(alpha=alpha)
        ridgeRegression.fit(X_train, y_train)
        #scores.append(ridgeRegression.score(X_test, y_test))
    return alphas, scores

def show_plot(alphas, scores):
    figure = plt.figure()
    ax = figure.add_subplot(1, 1, 1)
    ax.plot(alphas, scores)
    ax.set_xlabel(r"$\alpha$")
    ax.set_ylabel(r"score")
    ax.set_xscale("log")
    ax.set_title("Ridge")
    plt.show()
```

```
42.
43. if __name__ == '__main__':
44.
45.     A0=[0,0,130]
46.     A1=[500,0,170]
47.     A2=[0,500,170]
48.     A3=[500,500,130]
49.
50.     url = './预测/4 类/19 维训练正常.csv'
51.     data = pd. read_csv(url, sep=',',header=None)
52.     data=np.array(data)
53.     X_data=data[:,4:19]
54.     Y=data[:,19:22]
55.
56.     # url_test = './预测/场景 2/场景 2_正常.csv'
57.     # url_test = './预测/场景 1/场景 1_正常.csv'
58.     # url_test = './预测/4 类/19 维训练正常.csv'
59.     url_test = './预测/4 类/19 维测试正常.csv'
60.     data_test = pd. read_csv(url_test, sep=',',header=None)
61.     data_test=np.array(data_test)
62.
63.     X_data_test=data_test[:,4:19]
64.     Y_test=data_test[:,19:22]
65.
66.     X_train=X_data
67.     X_test=X_data_test
68.     Y_train=Y
69.     Y_test=Y_test
70.     print(Y_test)
71.     predict=test_ridge(X_train, X_test, Y_train, Y_test)
72.     print(predict)
73.
74.     dis=((predict[:,0]-Y_test[:,0])**2+(predict[:,1]-Y_test[:,1])**2+(predict[:,2]-Y_test[:,2])**2)
75.     dis_mean=np.sqrt(dis)
76.     print(np.mean(dis_mean))
```

(2) 异常数据预测

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import math
from sklearn import datasets, linear_model, discriminant_analysis
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler      #数据预处理
import csv
import joblib


def test_ridge(*data):
    X_train, X_test, y_train, y_test = data
    ridgeRegression = linear_model.Ridge()
    # ridgeRegression = linear_model.LinearRegression()
    rd=ridgeRegression.fit(X_train, y_train)
    joblib.dump(rd, "yc_model.pkl")

    #print("预测性能得分: %.2f" % ridgeRegression.score(X_test, y_test))
    return ridgeRegression.predict(X_test)

#测试不同的α值对预测性能的影响
def test_ridge_alpha(*data):
    X_train, X_test, y_train, y_test = data
    alphas = [0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10, 20, 50, 100, 200, 500, 1000]
    scores = []
    for i, alpha in enumerate(alphas):
        ridgeRegression = linear_model.Ridge(alpha=alpha)
        ridgeRegression.fit(X_train, y_train)
        #scores.append(ridgeRegression.score(X_test, y_test))
    return alphas, scores

def show_plot(alphas, scores):
    figure = plt.figure()
    ax = figure.add_subplot(1, 1, 1)
    ax.plot(alphas, scores)
    ax.set_xlabel(r"$\alpha$")
    ax.set_ylabel(r"score")
    ax.set_xscale("log")
    ax.set_title("Ridge")
    plt.show()


```

```
if __name__ == '__main__':

    A0=[0,0,130]
    A1=[500,0,170]
    A2=[0,500,170]
    A3=[500,500,130]

    url = './预测/16维训练异常.csv'
    data = pd. read_csv(url, sep=',',header=None)
    data=np.array(data)

    X_data=data[:,4:16]

    Y=data[:,16:19]

    # url_test = './预测/场景1/场景1_异常.csv'

    # url_test = './预测/16维训练异常.csv'
    url_test = './预测/16维测试异常.csv'
    # url_test = './预测/场景2/场景2_异常.csv'
    data_test = pd. read_csv(url_test, sep=',',header=None)
    data_test=np.array(data_test)

    X_data_test=data_test[:,4:16]
    Y_test=data_test[:,16:19]

    X_train=X_data
    X_test=X_data_test
    Y_train=Y
    Y_test=Y_test
    print(Y_test)

    predict=test_ridge(X_train, X_test, Y_train, Y_test)
    print(predict)

    dis=((predict[:,0]-Y_test[:,0])**2+(predict[:,1]-Y_test[:,1])**2+(predict[:,2]-Y_test[:,2])**2)

    dis_mean=np.sqrt(dis)
    print(np.mean(dis_mean))


    dis=((predict[:,0]-Y_test[:,0])**2+(predict[:,1]-Y_test[:,1])**2)
    dis_mean=np.sqrt(dis)
    print(np.mean(dis_mean))
```

附录 D 分类程序

```
clear all
clc
%% 导入数据
%事先分好训练集和测试集
load 2.d5特征数据集1016/d5trainData.mat %使用时需改
load 2.d5特征数据集1016/d5testData.mat %使用时需改
load 测试集/d5trackData.mat
%%
[Data_row1,Data_column1]=size(d5trainData); %使用时需改
[Data_row2,Data_column2]=size(d5testData); %使用时需改
Train_size=Data_row1;
Test_size=Data_row2;
d17Data=[d5trainData;d5testData];
[Data_row,Data_column]=size(d17Data);
% 训练数据
P_train = d5trainData(:,1:5)';        %数据维数不同要改索引
T_train = d5trainData(:,6)'+1;        %数据维数不同要改索引
% 轨迹测试数据
P_d5test=d5trackData(:,1:5)';      %分类预测
% 测试数据
P_test = d5testData(:,1:5)';
T_test = d5testData(:,6)'+1;
%% ------采用SSA-ELM进行分类----------------------------------------
%训练数据相关尺寸
R = size(P_train,1);
S = size(T_train,1);
N = 100;%隐含层个数
%% 定义麻雀优化参数
pop=20; %种群数量
Max_iteration=1; %  设定最大迭代次数
dim = N*R + N;%维度，即权值与阈值的个数
lb = [-1.*ones(1,N*R),zeros(1,N)];%下边界
ub = [ones(1,N*R),ones(1,N)];%上边界
fobj = @(x) fun(x,P_train,T_train,N,P_test,T_test);
[Best_pos,Best_score,SSA_curve]=SSA(pop,Max_iteration,lb,ub,dim,fobj); %开始优化
[fitness,IW,B,LW,TF,TYPE] = fun(Best_pos,P_train,T_train,N,P_test,T_test);%获取优化后的相关参数
figure
plot(SSA_curve,'linewidth',1.5);
grid on
xlabel('迭代次数')
ylabel('适应度函数')
title('SSA-ELM收敛曲线')
```

```
%% 保存权重和偏差矩阵
% weight=[IW B LW];       %IW:100x17   B:100x1 LW:100x2
% save('E:\办公\研究生\数学建模\SSA+ELM\2.d5 特征数据集 1016\weight.mat','weight')
%% 加载权重和偏差矩阵
load E:\办公\研究生\数学建模\SSA+ELM\2.d5 特征数据集 1016\acc9453zc9883yc9023/weight.mat
IW=weight(:,1:5);
B=weight(:,6);
LW=weight(:,7:8);
%% ELM 仿真测试
T_sim_1 = elmpredict(P_train,IW,B,LW,TF,TYPE);
T_sim_2 = elmpredict(P_test,IW,B,LW,TF,TYPE);
%% 轨迹预测
T_sim_3 = elmpredict(P_d5test,IW,B,LW,TF,TYPE)'-1;    %分类预测
save('E:\办公\研究生\数学建模\SSA+ELM\测试集\classResult.mat','T_sim_3')
csvwrite('E:\办公\研究生\数学建模\SSA+ELM\测试集\classResult.csv',T_sim_3)       %测试结果

%% 结果对比
disp(['麻雀优化 ELM 结果展示：----------------'])
result_1 = [T_train' T_sim_1'];
result_2 = [T_test' T_sim_2'];
% 训练集正确率
k1 = length(find(T_train == T_sim_1));
n1 = length(T_train);
Accuracy_1 = k1 / n1 * 100;
disp(['训练集正确率 Accuracy = ' num2str(Accuracy_1) '%(' num2str(k1) '/' num2str(n1) ')'])
% 测试集正确率
k2 = length(find(T_test == T_sim_2));
n2 = length(T_test);
Accuracy_2 = k2 / n2 * 100;
disp(['测试集正确率 Accuracy = ' num2str(Accuracy_2) '%(' num2str(k2) '/' num2str(n2) ')'])
%% 显示
count_Normal = length(find(T_train == 2));
count_Abnormal = length(find(T_train == 1));
%% 训练集混淆矩阵参数
train_number_Normal=length(find(T_sim_1 == 2 & T_train == 2));
train_number_Abnormal= length(find(T_sim_1 == 1 & T_train == 1));
%%
rate_Normal = count_Normal / 500;
rate_Abnormal = count_Abnormal / 500;
total_Normal = length(find(d17Data(:,6) == 1));    %数据维数不同要改索引
total_Abnormal = length(find(d17Data(:,6) == 0));    %数据维数不同要改索引

number_Normal = length(find(T_test == 2));
number_Abnormal = length(find(T_test == 1));
```

```
number_Normal_sim = length(find(T_sim_2 == 2 & T_test == 2));
number_Abnormal_sim = length(find(T_sim_2 == 1 & T_test == 1));
disp(['样本总数：' num2str(Data_row)...
      '  正常数据：' num2str(total_Normal)...
      '  异常数据：' num2str(total_Abnormal)]);
disp(['训练集样本总数：' num2str(Train_size)...
      '  正常数据：' num2str(count_Normal)...
      '  异常数据：' num2str(count_Abnormal)]);
disp(['训练集正常数据预测：' num2str(train_number_Normal)...
      '  预测错误：' num2str(count_Normal - train_number_Normal)...
      '  预测正确率 p1=' num2str(train_number_Normal/count_Normal*100) '%']);
disp(['训练集异常数据预测：' num2str(train_number_Abnormal)...
      '  预测错误：' num2str(count_Abnormal - train_number_Abnormal)...
      '  预测正确率 p2=' num2str(train_number_Abnormal/count_Abnormal*100) '%']);

disp(['测试集样本总数：' num2str(Test_size)...
      '  正常数据：' num2str(number_Normal)...
      '  异常数据：' num2str(number_Abnormal)]);
disp(['正常数据预测：' num2str(number_Normal_sim)...
      '  预测错误：' num2str(number_Normal - number_Normal_sim)...
      '  预测正确率 p1=' num2str(number_Normal_sim/number_Normal*100) '%']);
disp(['异常数据预测：' num2str(number_Abnormal_sim)...
      '  预测错误：' num2str(number_Abnormal - number_Abnormal_sim)...
      '  预测正确率 p2=' num2str(number_Abnormal_sim/number_Abnormal*100) '%']);


%% ------采用传统 ELM 进行分类----------------------------------------
%
  %% ELM 创建/训练
[IW1,B1,LW1,TF1,TYPE1] = elmtrain(P_train,T_train,100,'sig',1);
%% ELM 仿真测试
T_sim_11 = elmpredict(P_train,IW1,B1,LW1,TF1,TYPE1);
T_sim_22 = elmpredict(P_test,IW1,B1,LW1,TF1,TYPE1);
%% 训练集混淆矩阵参数
train_number_Normal1=length(find(T_sim_11 == 2 & T_train == 2));
train_number_Abnormal1= length(find(T_sim_11 == 1 & T_train == 1));
%% 结果对比
result_1 = [T_train' T_sim_11'];
result_2 = [T_test' T_sim_22'];
% 训练集正确率
disp(['传统 ELM 结果展示：----------------'])
k1 = length(find(T_train == T_sim_11));
n1 = length(T_train);
Accuracy_11 = k1 / n1 * 100;
```

```
disp(['训练集正确率 Accuracy = ' num2str(Accuracy_11) '%(' num2str(k1) '/' num2str(n1) ')'])
% 测试集正确率
k2 = length(find(T_test == T_sim_22));
n2 = length(T_test);
Accuracy_22 = k2 / n2 * 100;
disp(['测试集正确率 Accuracy = ' num2str(Accuracy_22) '%(' num2str(k2) '/' num2str(n2) ')'])
%% 显示
count_Normal = length(find(T_train == 2));
count_Abnormal = length(find(T_train == 1));
rate_Normal = count_Normal / 500;
rate_Abnormal = count_Abnormal / 500;
total_Normal = length(find(d17Data(:,6) == 1));    %数据维数不同要改索引
total_Abnormal = length(find(d17Data(:,6) == 0));    %数据维数不同要改索引
number_Normal = length(find(T_test == 2));
number_Abnormal = length(find(T_test == 1));
number_Normal_sim = length(find(T_sim_22 == 2 & T_test == 2));
number_Abnormal_sim = length(find(T_sim_22 == 1 & T_test == 1));
disp(['样本总数：' num2str(Data_row)...
      '  正常数据：' num2str(total_Normal)...
      '  异常数据：' num2str(total_Abnormal)]);
disp(['训练集样本总数：' num2str(Train_size)...
      '  正常数据：' num2str(count_Normal)...
      '  异常数据：' num2str(count_Abnormal)]);
disp(['测试集样本总数：' num2str(Test_size)...
      '  正常数据：' num2str(number_Normal)...
      '  异常数据：' num2str(number_Abnormal)]);

disp(['训练集正常数据预测：' num2str(train_number_Normal1)...
      '  预测错误：' num2str(count_Normal - train_number_Normal1)...
      '  预测正确率 p1=' num2str(train_number_Normal1/count_Normal*100) '%']);
disp(['训练集异常数据预测：' num2str(train_number_Abnormal1)...
      '  预测错误：' num2str(count_Abnormal - train_number_Abnormal1)...
      '  预测正确率 p2=' num2str(train_number_Abnormal1/count_Abnormal*100) '%']);

disp(['正常数据预测：' num2str(number_Normal_sim)...
      '  预测错误：' num2str(number_Normal - number_Normal_sim)...
      '  确诊率 p1=' num2str(number_Normal_sim/number_Normal*100) '%']);
disp(['异常数据预测：' num2str(number_Abnormal_sim)...
      '  预测错误：' num2str(number_Abnormal - number_Abnormal_sim)...
      '  确诊率 p2=' num2str(number_Abnormal_sim/number_Abnormal*100) '%']);

%% 麻雀优化算法
function [Best_pos,Best_score,curve]=SSA(pop,Max_iter,lb,ub,dim,fobj)

```

```
ST = 0.6;%预警值 0.6 0.7 0.2
PD = 0.7;%发现者的比列，剩下的是加入者
SD = 0.2;%意识到有危险麻雀的比重

PDNumber = pop*PD; %发现者数量
SDNumber = pop - pop*PD;%意识到有危险麻雀数量
if(max(size(ub)) == 1)
   ubub = ub.*ones(1,dim);
   lblb = lb.*ones(1,dim);
end

%种群初始化
X0=initialization(pop,dim,ub,lb);
X = X0;
%计算初始适应度值
fitness = zeros(1,pop);
for i = 1:pop
   fitness(i) =  fobj(X(i,:));
end
 [fitness, index]= sort(fitness);%排序
BestF = fitness(1);
WorstF = fitness(end);
GBestF = fitness(1);%全局最优适应度值
for i = 1:pop
    X(i,:) = X0(index(i),:);
end
curve=zeros(1,Max_iter);
GBestX = X(1,:);%全局最优位置
XX_new = X;
for i = 1: Max_iter
    BestF = fitness(1);
    WorstF = fitness(end);
    R2 = rand(1);
   for j = 1:PDNumber
      if(R2<ST)
          X_new(j,:) = X(j,:).*exp(-j/(rand(1)*Max_iter));
      else
          X_new(j,:) = X(j,:) + randn()*ones(1,dim);
      end
   end
   for j = PDNumber+1:pop
%       if(j>(pop/2))
       if(j>(pop - PDNumber)/2 + PDNumber)
          X_new(j,:)= randn().*exp((X(end,:) - X(j,:))/j^2);
```

```
        else
           %产生-1，1 的随机数
           A = ones(1,dim);
           for a = 1:dim
             if(rand()>0.5)
                 A(a) = -1;
             end
           end
           AA = A'*inv(A*A');
           X_new(j,:)= X(1,:) + abs(X(j,:) - X(1,:)).*AA';
        end
    end
    Temp = randperm(pop);
    SDchooseIndex = Temp(1:SDNumber);
    for j = 1:SDNumber
        if(fitness(SDchooseIndex(j))>BestF)
            X_new(SDchooseIndex(j),:) = X(1,:) + randn().*abs(X(SDchooseIndex(j),:) - X(1,:));
        elseif(fitness(SDchooseIndex(j))== BestF)
            K = 2*rand() -1;
             X_new(SDchooseIndex(j),:) = X(SDchooseIndex(j),:) + K.*(abs( X(SDchooseIndex(j),:) - X(
    end,:))./(fitness(SDchooseIndex(j)) - fitness(end) + 10^-8));
        end
    end
    %边界控制
    for j = 1:pop
        for a = 1: dim
            if(X_new(j,a)>ub)
                X_new(j,a) =ub(a);
            end
            if(X_new(j,a)<lb)
                X_new(j,a) =lb(a);
            end
        end
    end
    %更新位置
    for j=1:pop
     fitness_new(j) = fobj(X_new(j,:));
    end
    for j = 1:pop
     if(fitness_new(j) < GBestF)
        GBestF = fitness_new(j);
         GBestX = X_new(j,:);
     end
    end
```

```
    X = X_new;
    fitness = fitness_new;
     %排序更新
    [fitness, index]= sort(fitness);%排序
    BestF = fitness(1);
    WorstF = fitness(end);
    for j = 1:pop
       X(j,:) = X(index(j),:);
    end
    curve(i) = GBestF;
end
Best_pos =GBestX;
Best_score = curve(end);
end
%%  SSA+ELM 模型训练程序
function [IW,B,LW,TF,TYPE] = elmtrainNew(P,T,N,TF,TYPE,IW,B)
if nargin < 2
    error('ELM:Arguments','Not enough input arguments.');
end
if nargin < 3
    N = size(P,2);
end
if nargin < 4
    TF = 'sig';
end
if nargin < 5
    TYPE = 0;
end
if size(P,2) ~= size(T,2)
    error('ELM:Arguments','The columns of P and T must be same.');
end
[R,Q] = size(P);
if TYPE  == 1
    T  = ind2vec(T);
end
[S,Q] = size(T);
% Randomly Generate the Input Weight Matrix
% IW = rand(N,R) * 2 - 1;
% Randomly Generate the Bias Matrix
% B = rand(N,1);
BiasMatrix = repmat(B,1,Q);
% Calculate the Layer Output Matrix H
tempH = IW * P + BiasMatrix;
switch TF
```

```
    case 'sig'
        H = 1 ./ (1 + exp(-tempH));
    case 'sin'
        H = sin(tempH);
    case 'hardlim'
        H = hardlim(tempH);
end
% Calculate the Output Weight Matrix
LW = pinv(H') * T';
%%  传统 ELM 模型训练程序
function [IW,B,LW,TF,TYPE] = elmtrain(P,T,N,TF,TYPE)
if nargin < 2
    error('ELM:Arguments','Not enough input arguments.');
end
if nargin < 3
    N = size(P,2);
end
if nargin < 4
    TF = 'sig';
end
if nargin < 5
    TYPE = 0;
end
if size(P,2) ~= size(T,2)
    error('ELM:Arguments','The columns of P and T must be same.');
end
[R,Q] = size(P);
if TYPE  == 1
    T  = ind2vec(T);
end
[S,Q] = size(T);
% Randomly Generate the Input Weight Matrix
IW = rand(N,R) * 2 - 1;
% Randomly Generate the Bias Matrix
B = rand(N,1);
BiasMatrix = repmat(B,1,Q);
% Calculate the Layer Output Matrix H
tempH = IW * P + BiasMatrix;
switch TF
    case 'sig'
        H = 1 ./ (1 + exp(-tempH));
    case 'sin'
        H = sin(tempH);
    case 'hardlim'
```

```
        H = hardlim(tempH);
end
% Calculate the Output Weight Matrix
LW = pinv(H') * T';
%% 适应度函数，以训练集和测试集的分类错误率作为适应度值
function [fitness,IW,B,LW,TF,TYPE] = fun(x,P_train,T_train,N,P_test,T_test)
R = size(P_train,1);
S = size(T_train,1);
IW = x(1:N*R);
B = x(N*R+1:N*R+N*S);
IW = reshape(IW,[N,R]);
B = reshape(B,N,1);
TYPE = 1;%分类
TF = 'sig';
[IW,B,LW,TF,TYPE] = elmtrainNew(P_train,T_train,N,TF,TYPE,IW,B);
%% ELM 仿真测试
T_sim_1 = elmpredict(P_train,IW,B,LW,TF,TYPE);
T_sim_2 = elmpredict(P_test,IW,B,LW,TF,TYPE);
% 训练集正确率
k1 = length(find(T_train == T_sim_1));
n1 = length(T_train);
Accuracy_1 = k1 / n1;
% 测试集正确率
k2 = length(find(T_test == T_sim_2));
n2 = length(T_test);
Accuracy_2 = k2 / n2;
%错误率
fitness = 2 - Accuracy_1 - Accuracy_2;
end
```