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	景德镇陶瓷大学
参赛队号	21104080004
队员姓名	1.肖佳涛
	2.覃文辉
	3.程文纺

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 基于 EKF 和 BP 神经网络的超宽带定位抗干扰综合研究

摘 要：

本文主要研究了 UWB 的精确定位问题，针对室内环境复杂多变的特点，用卡尔曼滤波算法（EKF）来解决在信号干扰下数据异常波动的问题，精确超宽带定位。为了进一步提高定位精度，减小硬件自身以及其他问题对定位的干扰，加入了 BP 神经网络与集成学习思想的分类模型来提高精度。

针对任务一，本题主要研究数据为定点多次重复测量的数据，重复度相似度较高，且存在无意义数据维度，以及完全重复的数据，采用 3σ 原则服从正态分布，剔除异常分布；应用聚类算法进行相似数据分类，剔除相似数据，最后利用 Pandas 数据分析工具抓取原始数据，得到 648 个符合该任务要求的数据文件，在附录部分将重点展示题目要求的指定文件的数据，每个文件最后保留 30 条数据。

针对任务二，首先取得任务一处理后的数据集，分为“正常数据”与“异常数据”两大类，然后根据附件 1 中 Tag 坐标信息人为的给这些数据打上信号有无干扰的标签，将这些数据整理合并成两个数据集：信号无干扰数据集与有干扰数据集。

针对该任务问题（1），分别创建两个 BP 神经网络定位模型，将数据集分别导入训练得到两个定位模型：信号无干扰定位模型和信号有干扰定位模型。最后将实验场景信息与处理后的附件 2 信息导入模型即可预测出靶点 Tag 坐标，其预测精度的损失值 loss 较低，为 100 左右，模型拟合效果较好。

针对该任务问题（2），在获得定位模型后，使用定位模型对任务一中处理后的数据进行预测，得到预测的靶点坐标信息，将靶点坐标信息与对应的真实靶点坐标信息带入欧式距离公式获得距离损失，做为定位模型的 3 维，2 维以及 1 维精度准则，损失值越小精度越高，损失值分别为 3 维：183.83；2 维：XOY 平面 207.60，XOZ 平面 172.74，YOZ 平面 168.64；1 维：X 轴 210.94，Y 轴 204.20，Z 轴 123.21，损失值接近 0，精度较高。

针对任务三，首先将附件 3 中的数据进行处理，获得同任务一最后文件相同格式的数据，将这些数据和场景 2 的信息数据分别导入两个模型，即可预测出场景 2 中靶点的位置信息。

针对任务四，要想实现对数据判别信号是否受到干扰，必须指定一个分类模型，由于常用的分类模型，如 KNN，SVM，RF 等模型各自都有优缺点，分类结果并不一定完全可靠，所以采取集成学习的思想，使用多个分类模型分别进行分类预测，最后将它们的结果进行比较投票，此时的分类结果具有较高的说服力。另外任务一的数据都是已知信号有无干扰采集的数据，所以采用任务一清洗后的数据与对应的有无干扰状态做为训练集供分类模型使用。使用训练好的集成分类模型，即可成功预测出附件 4 的数据类别。

针对任务五，该任务在场景 1 进行实验，靶点不再静止，而是处于运动状态，另外采集到的数据无干扰信号，有的是有干扰信号数据，所以需要先使用分类模型判断出数据的类别，再输入后续的定位模型中进行轨迹预测。卡尔曼滤波具有良好的动态跟踪定位的效果，扩展卡尔曼滤波可以将非线性系统近似线性化，对于该任务扩展卡尔曼滤波较为适用。故不同于任务二、三的预测流程，此处先加入扩展卡尔曼滤波定位模型进行靶点坐标预测，再加入 BP 神经定位模型，两者结合进行定位，最后可以得到高精度的坐标预测。根据模型预测，最后得出结论：靶点运动轨迹大致呈现一个‘Z’字型，运动高度基本为 1500mm 左右，起点坐标大致为（632mm，622mm，1613mm），终点坐标大致为（4521mm，4514mm，1529mm），并且在（950mm，220mm，1540mm）与（3600mm，668mm，1541mm）处均停留了 40 秒左右。假设每一段运动为匀速运动，结合数据的时间戳分析得知，第一段路程的运动速率约为 70mm/s，第二段路程的运动速率约为 51mm/s，最后一段的运动速率约为 138mm/s。

关键词：扩展卡尔曼滤波，BP 神经网络，集成学习，UWB 定位

目录

1. 问题重述

1.1 问题背景

UWB（Ultra-Wideband）[1]是一种无载波通信技术，又称“超宽带”或脉冲无线电技术。出现于 20 世纪 60 年代，起初在运用于雷达系统与战场通讯，随后探索于民用领域。其具有定位准确、安全性高、功耗低、数据传输速度快、多径分辨能力强等特点，因此受到各个国家的重视，随着越来越多的团队的加入，纷纷开展了研究计划，不断在各个领域获得重大发现，最终在电力、医疗、化工行业、隧道施工、家电设备等各个领域被广泛应用。

TOF[2]测距方法属于双向测距技术，它主要利用信号在两个异步收发机之间往返的飞行时间来测量节点间的距离。TOF 测距技术不同于传统的测距技术，是飞行时差测距的方法。

现如今，随着科技的不断发展，对生活质量的要求越来越高，室内定位是人们日益关注的话题，因此市场上出现了更多的定位服务系统，比如连接信号的 WIFI，手机上的蓝牙，遥控器上的红外线以及超宽带等技术，这些不断兴起的服务系统主要运用的就是 UWB 定位技术。但是由于建筑物的增加，信号穿透墙壁的能力较弱，信号在遇到遮挡物时还容易受到折射和反射的影响，这些都会对信号强弱造成一定的影响，严重的还会直接遭遇信号中断，如在地下停车场、电梯、偏僻角落等区域信号很弱，会出现明显的偏差，这些对人类的生活造成很大的不便，这在一定程度上也阻碍了室内定位系统的发展。因此，在室内定位中如何在各种信号干扰下依旧精确定位成为首要要考虑和解决的问题。

1.2 问题描述

本研究为解决信号干扰下的超宽带精确定位问题，题目给出 5 个附件数据，附件 1 是 UWB 数据集，包括无干扰数据、有干扰数据以及 Tag 坐标信息；附件 2、3、4 给出了实验场景的测试集；附件 5 是动态轨迹数据。要求利用该数据解决如下问题：

任务一：附件 1 是靶点 Tag 分别在 324 个固定点，以有无信号干扰分为两组，在不同时刻下的状态数据组成的 UWB 数据集。要求建立方法提取每个数据文件所需数值，转换为二维表，并清洗数据，最后列举出指定数据文件的保留数据。

任务二：结合任务一处理后数据，分别对“无干扰数据”和“有干扰数据”建立适当数学模型，以预测 Tag 的精确位置，并给出定位模型 1 维、2 维、3 维的精度。最后利用该数学模型对附件 2 中的数据进行精确定位。

任务三：应用任务二中建立的定位模型，改变实验场景，分别对附件 3 中 5 组无信号干扰数据、5 组有信号干扰数据进行精确定位。

任务四：利用任务一处理后的数据，建立恰当的数学分类模型来区分数据采集时信号是否有干扰，并说明所建立的分类模型的精度。最后结合分类模型，判断附件 4 所提供的数据信号受干扰情况。

任务五：结合静态点定位模型，分析靶点自身运动规律，建立适当模型，对附件 5 中动态靶点的运动轨迹进行精确定位。

1.3 已知条件说明

本文将从两个角度对已知条件进行说明：

实验场景布置数据，锚点与靶点的信息；

场景中实验收集到的数据(下文简称实验数据)解释说明。

1.3.1 实验场景布置

实验分为两个场景，每个场景都会安放 1 个靶点（Tag）与 4 个锚点(anchor)。每个场景的所有锚点静止不动，靶点可能静止可能运动。在没有特别说明时实验数据默认为靶点静止时收集到的数据。场景数据如表 1-1 所示：

表 1-1　场景数据（单位：mm）

		实验场景 1	实验场景 2
靶点（Tag）范围		5000×5000×3000	5000×3000×3000
锚点（anchor）位置	A_0	（0，0，1300）	（0，0，1200）
	A_1	（5000，0，1700）	（5000，0，1600）
	A_2	（0，5000，1700）	（0，3000，1600）
	A_3	（5000，5000，1300）	（5000，3000，1200）

1.3.2 实验数据

本文共有 5 大附件数据，分别为：

（1）附件 1：UWB 数据集，包含在场景 1 时收集到的信号无干扰数据与信号有干扰数据，以及靶点 Tag 坐标信息；

（2）附件 2：测试集（实验场景 1），包含 5 组信号无干扰数据与 5 组信号有干扰数据，共 40 条数据；

（3）附件 3：测试集（实验场景 2），包含 5 组信号无干扰数据与 5 组信号有干扰数据，共 40 条数据；

（4）附件 4：测试集（实验场景 1），包含 10 组共 40 条数据，每组数据是否是在信号有无干扰的情况下收集得来未知；

（5）附件 5：动态轨迹数据，在场景 1 中收集得来，靶点处于运动状态。

2. 模型条件假设

[1] 假设实验场景均位于封闭空间，不受外界天气因素影响。

[2] 假设实验过程中不存在由网络波动导致的误差。

[3] 假设靶点运动时只受预设程序控制，不存在由信号传输或其他原因引起的卡顿。

[4] 假设在实验时锚点全程固定不动，不存在装置松脱的现象。

[5] 假设实验装置电力充足，不存在因电力电压问题导致的误差。

3. 符号说明

符号	含义说明
A_i，$i = 0,1,2,3$	锚点（anchor）
d_i，$i = 0,1,2,3$	A_i锚点到 Tag 靶点的测距
Tag	靶点（Tag）
BP	Back Propagation
KF	卡尔曼滤波
EKF	扩展卡尔曼滤波
MSE	均方误差

4. 模型建立过程中用到的相关知识

4.1 3σ原则

3σ原则又名拉依达准则，它建立在正态分布的等精度重复测量基础上。假设某组数据只包含随机误差，然后对这组数据处理得到标准偏差σ。如果该组数据中某个测量值的残余误差的绝对值$v_i > 3\sigma$，则该值为坏值，应该剔除。通常把±3σ作为极限误差界限，落在界限以外的概率不到 0.3%，发生的概率很小，所以存在 3σ准则。一般应用于多次测量（$n_i \geq 30$）的情况中。

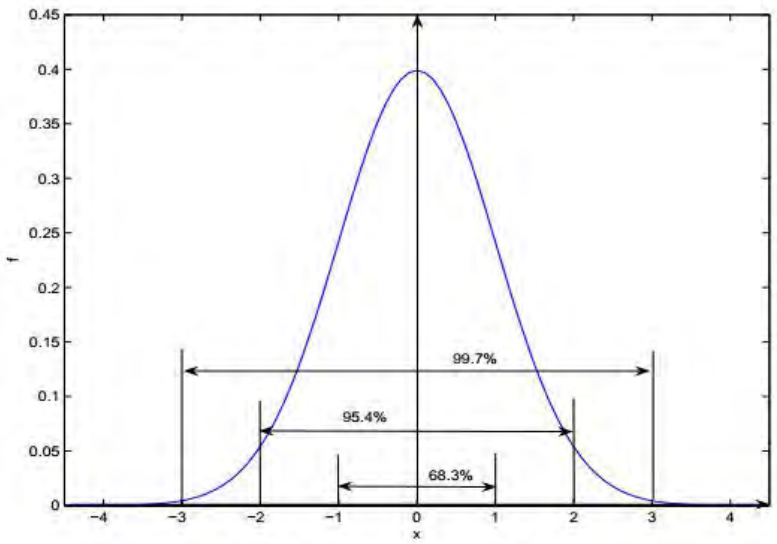


图 4-1 标准正态分布和对应区间上积分（面积）的百分比

4.2 K-Means 聚类算法

K 均值聚类算法（K-Means Clustering Algorithm）[3]是一种迭代求解的无监督聚类算法。该算法可将数据集分为 k 组。首先随机选取 k 个点作为 k 个中心，然后其余点到每个中心点的距离，按照距离大小将它们分成 k 组，再计算每组新的中心点，直到新的中心点和初始中心点重合便结束（是否重合可以设置一个阈值来作为标准）。

对于 K 值的选取，一般根据对数据的先验经验进行选择，如果没有先验经验时，可以通过交叉验证设置多个 K 值，最后根据肘部法则选取一个合适的 K 值。

4.3 最小二乘解

最小二乘解又称最小平方法，在数据曲线拟合或者线性方程的求解中，利用最小二乘解求得的结果称为最小二乘解，这是一种数学优化方法，它通过最小化误差的平方和寻找数据的最佳数据匹配。

4.4 BP 神经网络

BP（Back Propagation）神经网络[4][5]是 1986 年由 McClelland 和 Rumelhart 为首的科学家提出概念，它是一种按照误差逆向传播算法训练的多层前馈神经网络，是目前广泛应用的神经网络模型之一，也是如今深度学习算法的基础。

BP 算法的基本思想是，运作过程分为信号的正向传播和反向传播，由 1 个输入层，多个隐藏层以及 1 个输出层组成，每一层包含多个神经元，BP 算法结构如 4-2 所示：

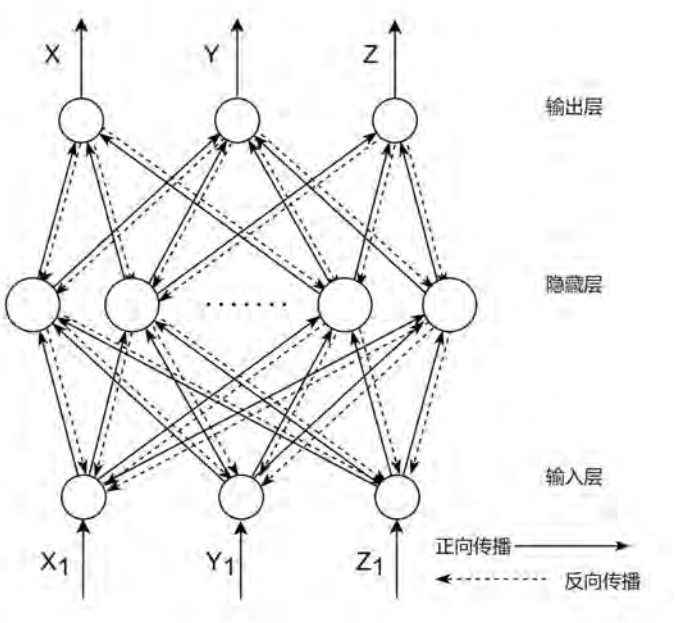


图 4-2 BP 神经网络结构图

4.5 F1 分数

F1 分数（F1 Score），是统计学中用来衡量二分类模型精确度的一种指标。它兼顾了分类模型的精确率和召回率。

假设 P 表示预测的正类，N 代表预测的负类，T 代表真实的正类，F 代表真实的负类。在进行二分类问题时，预测结果将会出现如表 4-1 所示的结果：

表 4-1 混淆矩阵

结果 / 预测 / 真实	正例	负例
正例	TP	FN
负例	FP	TN

TP (True Positive)：本类标签预测为本类

FP (False Positive)：其他类预测为本类

FN (False Negative)：本类标签预测为其他类

精准度 (Precision)：在预测结果中，预测出来的正例数量占该预测总量的比例

$$\text{Precision} = \text{TP} / (\text{TP} + \text{FP}) \tag{4-1}$$

召回率 (Recall)：预测出来的该类数量占该类所有数量的比例

$$\text{Recall} = \text{TP} / (\text{TP} + \text{FN}) \tag{4-2}$$

F1-Score：F1 分数，综合了精准度和召回率，用于综合反映整体指标

$$\text{F1} = 2\ (\text{Precision} \times \text{Recall}) / (\text{Precision} + \text{Recall}) \tag{4-3}$$

F1 指标取值在 0 到 1 之间，越接近 1 说明分类效果越好。

4.6 均方误差

均方误差（Mean Squared Error）简称 MSE 函数一般用来检测模型的预测值和真实值之间的偏差。

$$\text{MSE} = \frac{1}{\text{N}}\sum_{\text{n}=1}^{\text{n}} (\text{y}_\text{n} - \overline{\text{y}}_\text{n})^2 \tag{4-4}$$

由公式（4-1）可以看出 MSE 是真实值和预测值差值的平凡然后求和，范围是[0,+∞)，误差越大该值越大，理论上 MSE 为 0 时模型拟合效果越好。

4.7 卡尔曼滤波原理

在许多工程实践中，往往不能直接测量获取状态变量。卡尔曼滤波（KF，Kalman Filtering）[6]是一种高效的自回归波器，可以对状态变量进行自修正，能够有效地降低随机噪声的影响，在线性系统中，卡尔曼滤波是最优滤波器。在几何上，卡尔曼滤波是将状态变量在观测量生成的线性空间上的射影。在已知上一时刻状态估计值的基础上卡尔曼滤波器只需获取当前状态的观测值就可以计算出当前状态的估计值，不需要记录观测或估计的历史信息。用如下状态空间模型描述的动态系统：

$$\begin{cases} x_k = F_k x_{k-1} + B_k u_k + \Gamma w_k \\ z_k = H_k x_k + v_k \end{cases} \tag{4-5}$$

公式（4-5）中，第 1 个方程为卡尔曼滤波状态估计方程，第 2 个方程为卡尔曼观测方程，式中变量含义如表 4-2 所示：

表 4-2 卡尔曼方程符号说明表

序号	变量	名称和作用
1	x_k	状态向量，估计预测状态
2	F_k	$k-1$时刻到k时刻的状态转移矩阵
3	B_k	控制矩阵
4	u_x	控制向量
5	Γ	为噪声驱动矩阵
6	w_k	符合零均值且协方差矩阵为Q_k多元正态分布的过程噪声
7	z_k	观测向量，记录观测值z_k
8	H_k	观测矩阵
9	v_k	符合零均值且协方差矩阵为R_k的正态分布观测噪声

从（4-5）式可知，状态方程建立了$k-1$时刻和k时刻状态向量间的关系，观测方程建立了k时刻状态向量和观测向量间的关系，因此在给定初始值后，可以得到任意时刻的估计值。

在 KF 算法中，$x_{k,k}$表示k时刻状态的最佳估计，$P_{k,k}$为后验估计误差协方差矩阵，用以度量分析状态估计值的精确程度，$P_{k,k-1}$为预测估计协方差矩阵，用于度量分析状态预测值的精确程度。具体算法流程如下：

（1）状态预测：

$$x_{k,k-1} = F_k x_{k-1,k-1} + B_k u_k \tag{4-6}$$

（2）计算预测估计值协方差矩阵：

$$P_{k,k-1} = F_k P_{k-1,k-1} F_k^{\ T} + Q_k \tag{4-7}$$

（3）计算最优卡尔曼增益矩阵：

$$K_k = P_{k,k-1} H_k (H_k P_{k,k-1} H_k^T + R_k)^{-1} \tag{4-8}$$

（4）更新状态估计：

$$x_{k,k} = x_{k,k-1} + K_k(z_k - H_k x_{k,k-1}) \tag{4-9}$$

（5）更新后验估计误差协方差矩阵：

$$P_{k,k} = (I - K_k H_k)P_{k,k-1} \tag{4-10}$$

4.8 扩展卡尔曼滤波原理

卡尔曼滤波器主要应用于线性条件，而实际应用往往是非线性的问题，可以使用扩展卡尔曼滤波算法（EKF，Extended Kalman Filter）[7]。扩展卡尔曼滤波通过 Taylar 级数展开将非线性系统进行线性化。

在扩展卡尔曼滤波中，对应的状态估计方程和观测量观测方程如公式所示。

$$\begin{cases} x_k = f_k(x_{k-1}, u_k) + \Gamma w_k \\ z_k = h_k(x_k) + v_k \end{cases} \tag{4-11}$$

（4-11）式中，过程噪声 w_k 符合 $N(0, Q_k)$ 分布，观测噪声 v_k 符合 $N(0, R_k)$ 分布。与卡尔曼滤波算法类似，具体的算法流程如下：

（1）状态预测：

$$x_{k,k-1} = f(x_{k-1,k-1}, u_k) \tag{4-12}$$

（2）计算预测估计值协方差矩阵：

$$P_{k,k-1} = F_k P_{k-1,k-1} F_k^T + Q_k \tag{4-13}$$

（3）计算最优卡尔曼增益矩阵：

$$K_k = P_{k,k-1} H_k (H_k P_{k,k-1} H_k^T + R_k)^{-1} \tag{4-14}$$

（4）更新状态估计：

$$\mathrm{x}_{k,k} = x_{k,k-1} + K_k(z_k - h(x_{k,k-1})) \tag{4-15}$$

（5）更新后验估计误差协方差矩阵：

$$P_{k,k} = (I - K_k H_k)P_{k,k-1} \tag{4-16}$$

（6）对函数 f 和函数 h 求雅克比矩阵，可求得 F_k 和 H_k 矩阵，公式如下

$$F_k = \left.\frac{\partial f}{\partial x}\right|_{x_{k,k-1}u_k} \tag{4-17}$$

$$H_k = \left.\frac{\partial h}{\partial x}\right|_{x_{k,k-1}} \tag{4-18}$$

5. 模型的建立与求解

5.1 任务一模型建立与求解

5.1.1 任务一问题分析

附件 1 中 UWB 的数据分为无干扰数据、有干扰数据与 Tag 坐标信息，Tag 坐标信息表中含有 324 个坐标，对应于“异常数据”文件夹中有 324 个无干扰数据文件，同理，“正常数据”文件夹中有 324 个有干扰数据文件。Tag 坐标信息文件中发现编号 148 与编号 150 之间的记录为“194: 400 250 130”，后文中同样存在编号 194 的记录，可认定为记录编号错误，改正为“149: 400 250 130”.

每次采集数据时，Tag 会在每个坐标点固定，根据时间戳规律 Tag 与 anchor 按每 200-300ms 发送、接收一次信号。在同一 Tag 位置上，无干扰数据、有干扰数据文件中，均是多组重复采集的数据。每个样本有 4 条数据记录，每条记录由冒号间隔参数，为了方便后续使用数据，需要将数据进行取舍、计算、规格化。重复采集的数据更需要找出差异性数据，故需要进行去重去相似，同时也需要避免外界因素带来的异常值数据。

5.1.2 模型数据准备

以附件 1 中“1.正常.txt”前四条数据为例，每个 txt 文件中，每四条数据为一组测距值，原始数据说明如图 5-1 所示：

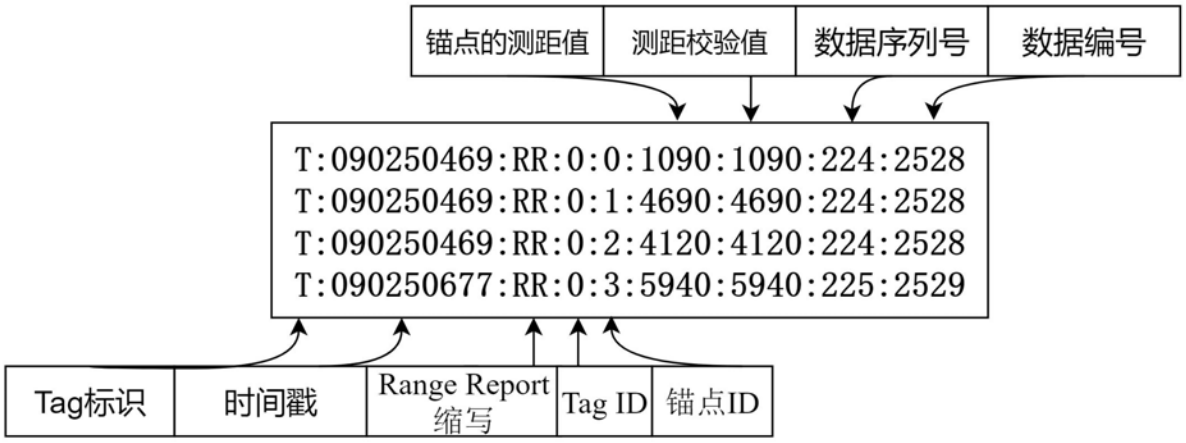


图 5-1 原始数据说明图

5.1.3 模型建立流程

数据预处理流程图如图 5-2 所示：

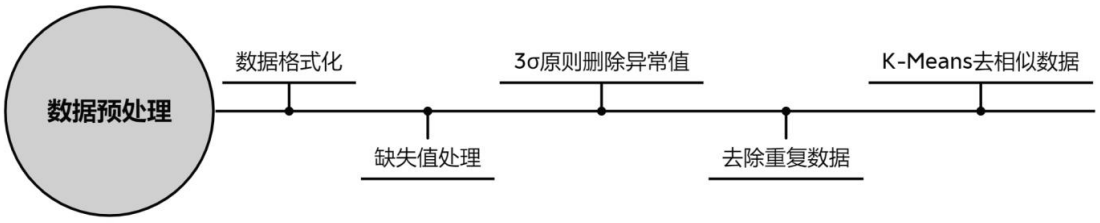


图 5-2 任务一模型流程图

（1）数据格式化

根据数据文件说明，在无干扰下（正常）采集的数据集与有干扰下（异常）采集的数据集中，需要删除文件中无意义的数据（如首行、Tag 标识、RR 标记等），其中的“数据序列号”的作用可由数据编号代替，来表示数据组别，同样去除。检查所有锚点的测距值及其校验值均为相等，故仅保留测距值。对于每个样本组内时间戳存在多个数据，通过观察组内时间戳差异很小，本文统一取样本组第一条时间戳为本样本时间戳数据。

（2）部分缺失值处理

考虑到可能存在信号接收获取或者是存储过程中的硬件异常，数据集中可能存在缺失。获取数据后，首先检测数据是否有缺失的空值，由于每四条数据为一组，如果有的话将缺失值补充完整而不是删除整条数据，处理完成后进行下一步。

（3）3σ原则删除异常值

对于异常值，本文选用 3σ原则去除。每个数据集中的数据均为大量的重复测量，假设数据集中的所有检测数据只含随机误差且服从正态分布，用样本均值代替数学期望μ，则数值分布在$(\mu-3\sigma，\mu+3\sigma)$之间的概率约为 99.74%，可认为超出该区间的数据为异常数据，并将其删除。

（4）去重处理

本任务中的数据以时间为重复测量数据，必然存在大量重复值，以 4 个锚点到靶点距离为搜寻依据，直接剔除重复记录，每个文件仅保留第一次出现的记录，记录之间的时间顺序不变。

（5）K-Means 去相似数据

将上一步得到的数据使用 K-Means 算法进行聚类，每个类别中的数据认为是相似数据，类别之间的数据认为不相似，然后取每个类别中的第一条数据，最后保留 30 条数据做为最终处理结果。

5.1.4 数据处理结果

处理后，最终得到每条记录为一个样本，每个文件里面是同一位置多条距离测量信息，共得到 324 个“正常数据”文件，324 个“异常数据”文件，保存为“xls”格式。此处例举四个文件部分数据，具体数据详见附录一，其他数据详见上传附件。

表 5-1 “24.正常.txt”文件数据处理后部分数据

时间戳(ms)	$d_0(\mathrm{mm})$ (A_0到靶点距离)	$d_1(\mathrm{mm})$ (A_1到靶点距离)	$d_2(\mathrm{mm})$ (A_2到靶点距离)	$d_3(\mathrm{mm})$ (A_3到靶点距离)
094843811	3280	4660	2600	3910
094844019	3280	4670	2600	3920
.....				
094910925	3300	4650	2610	3900
094912796	3300	4660	2610	3920

表 5-2 “109.正常.txt”文件数据处理后部分数据

时间戳(ms)	$d_0(\mathrm{mm})$ (A_0到靶点距离)	$d_1(\mathrm{mm})$ (A_1到靶点距离)	$d_2(\mathrm{mm})$ (A_2到靶点距离)	$d_3(\mathrm{mm})$ (A_3到靶点距离)
144156086	4910	5310	2030	2870
144156294	4920	5310	2020	2870
.....				
144235391	4910	5330	2040	2900
144244123	4930	5340	2020	2880

表 5-3 “1.异常.txt”文件数据处理后部分数据

时间戳(ms)	d_0(mm) (A_0到靶点距离)	d_1(mm) (A_1到靶点距离)	d_2(mm) (A_2到靶点距离)	d_3(mm) (A_3到靶点距离)
090531088	1280	4550	4550	6300
090531296	1260	4550	4550	6300
.....				
090626563	780	5050	4550	6300
090640431	780	5010	4550	6280

表 5-4 “100.异常.txt”文件数据处理后部分数据

时间戳(ms)	d_0(mm) (A_0到靶点距离)	d_1(mm) (A_1到靶点距离)	d_2(mm) (A_2到靶点距离)	d_3(mm) (A_3到靶点距离)
115709181	1510	3750	4690	5710
115709389	1520	3720	4680	5720
.....				
115758725	1520	3560	4910	5690
115806212	1510	3560	4810	5700

5.2 任务二模型建立与求解

5.2.1 任务二问题分析

结合任务一处理后数据，建立 Tag 精确位置的预测模型。首先结合 UWB 定位原理出发，建立基础数学计算模型。再结合 UWB 信号特点，其利用较宽的频谱传输较低的频率，能达到 0.1-0.15m 的定位精度，有很强的穿透力，但依然存在非视距（NLOS，Non Line of Sight）[8]产生的误差。当靶点与锚点间有遮挡时，UWB 信号就不能直接到达靶点，会通过反射、折射、衍射等方式绕过障碍物，得到的测距值往往比实际真实值要大，另外定位过程中还会受到环境中的噪声影响。为了提高 UWB 的测量精度，使模型更加准确，排除其他情况的干扰，因此需要加入 BP 神经网络对模型进行完善。

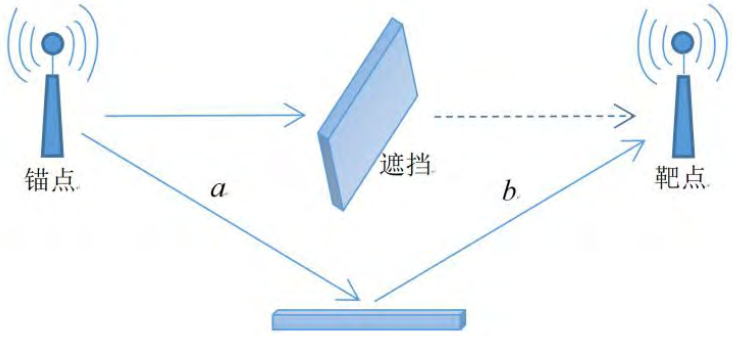


图 5-3 NLOS 误差原理图

5.2.2 模型数据准备

（1）模型训练数据

场景 1 中的锚点位置信息，靶点位置信息作为初始数据输入模型；

任务一处理后的“正常.xls”与“异常.xls”数据将作为训练集的特征值；

附件 1 中的 Tag 坐标信息将作为训练集的目标值。

（2）模型预测数据

附件 2 中数据作为预测集，输入到定位模型中求解。

5.2.3 定位模型建立原理

根据测试空间环境，建立位置坐标的数学模型，以测试场景信息为基础数据，以 4 个锚点到靶点的距离为输入，求解靶点空间坐标。其测距定位的结构原理图如下图所示，其中，各锚点坐标分别为$A_0(x_0,y_0,z_0)$，$A_1(x_1,y_1,z_1)$，$A_2(x_2,y_2,z_2)$，$A_3(x_3,y_3,z_3)$，其靶点坐标为$Tag(x,y,z)$ ，各锚点到靶点的的距离分别为d_0，d_1，d_2，d_3.

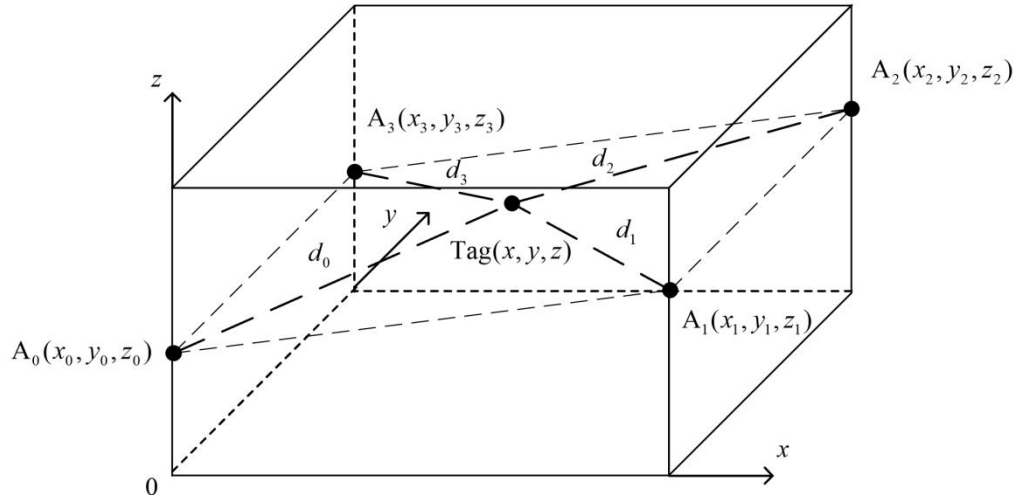


图 5-4 UWB 测距定位的结构原理图

根据靶点与锚点的坐标关系，联立可得方程组如（5-1）式

$$\begin{cases}(x_0-x)^2+(y_0-y)^2+(z_0-z)^2=d_0^{\ 2}\\(x_1-x)^2+(y_1-y)^2+(z_1-z)^2=d_1^{\ 2}\\(x_2-x)^2+(y_2-y)^2+(z_2-z)^2=d_2^{\ 2}\\(x_3-x)^2+(y_3-y)^2+(z_3-z)^2=d_3^{\ 2}\end{cases} \tag{5-1}$$

对（5-1）式进行移项化简可得：

$$\begin{cases}-2(x_0-x_3)x-2(y_0-y_3)y-2(z_0-z_3)z=d_0^{\ 2}-d_3^{\ 2}+x_3^{\ 2}+y_3^{\ 2}+z_3^{\ 2}-x_0^{\ 2}-y_0^{\ 2}-z_0^{\ 2}\\-2(x_1-x_3)x-2(y_1-y_3)y-2(z_1-z_3)z=d_1^{\ 2}-d_3^{\ 2}+x_3^{\ 2}+y_3^{\ 2}+z_3^{\ 2}-x_1^{\ 2}-y_1^{\ 2}-z_1^{\ 2}\\-2(x_2-x_3)x-2(y_2-y_3)y-2(z_2-z_3)z=d_2^{\ 2}-d_3^{\ 2}+x_3^{\ 2}+y_3^{\ 2}+z_3^{\ 2}-x_2^{\ 2}-y_2^{\ 2}-z_2^{\ 2}\\-2(x_3-x_3)x-2(y_3-y_3)y-2(z_3-z_3)z=d_3^{\ 2}-d_3^{\ 2}+x_3^{\ 2}+y_3^{\ 2}+z_3^{\ 2}-x_3^{\ 2}-y_3^{\ 2}-z_3^{\ 2}\end{cases} \tag{5-2}$$

最终可化简为$AX=b$的形式，如下式

$$-2\begin{bmatrix}x_0-x_3 & y_0-y_3 & z_0-z_3\\x_1-x_3 & y_1-y_3 & z_1-z_3\\x_2-x_3 & y_2-y_3 & z_2-z_3\end{bmatrix}\begin{bmatrix}x\\y\\z\end{bmatrix}=\begin{bmatrix}d_0^{\ 2}-d_3^{\ 2}+x_3^{\ 2}+y_3^{\ 2}+z_3^{\ 2}-x_0^{\ 2}-y_0^{\ 2}-z_0^{\ 2}\\d_1^{\ 2}-d_3^{\ 2}+x_3^{\ 2}+y_3^{\ 2}+z_3^{\ 2}-x_1^{\ 2}-y_1^{\ 2}-z_1^{\ 2}\\d_2^{\ 2}-d_3^{\ 2}+x_3^{\ 2}+y_3^{\ 2}+z_3^{\ 2}-x_2^{\ 2}-y_2^{\ 2}-z_2^{\ 2}\end{bmatrix} \tag{5-3}$$

求$AX=b$最佳最小二乘解为 $X=(A^TA)^{-1}A^Tb.$

由最佳最小二乘解得到的 X 为靶点位置的初始预测值，考虑到硬件误差、人为操作误差以及其他原因，需要对 X 进行修正，修正方法采用 BP 神经网络。

将场景 1 中各锚点A_0,A_1,A_2,A_3作为输入，结合任务一处理后的d_0,d_1,d_2,d_3数据，使用最小二乘法求解出初始的 Tag 坐标(x_1,y_1,z_1)，将(x_1,y_1,z_1)作为输入，真实的 Tag 坐标(x,y,z)作为输出，输入到 BP 网络模型中进行训练，最后得出完整定位模型。由于数据分

为信号无干扰数据和有干扰数据，所以会得出 2 个定位模型，分别记作无干扰定位模型、有干扰定位模型。

保存训练好的模型，只需要输入锚点的位置信息，靶点返回的初始测量距离即可得到高精度的靶点精确定位。

5.2.4 任务求解与模型评估分析

（1）结果展示

使用无干扰定位模型对附件 2 中的无干扰信号求解，最终的定位结果如表 5-5 所示：

表 5-5 附件 2 无干扰信号定位结果

序号	X/mm	Y/mm	Z/mm
1	1162	721	1158
2	3200	1734	1022
3	2767	1251	1092
4	2394	1081	1862
5	1514	2535	1854

使用有干扰定位模型对附件 2 中的有干扰信号求解，最终的定位结果如下表所示：

表 5-6 附件 2 有干扰信号定位结果

序号	X/mm	Y/mm	Z/mm
1	2037	827	1499
2	4206	1736	1452
3	1769	1254	1532
4	3499	2058	1485
5	4741	2159	1543

在获得定位模型后，使用定位模型对任务一中处理后的数据进行预测，得到预测的靶点坐标信息，将靶点坐标信息与对应的真实靶点坐标信息带入欧式距离公式获得距离损失，做为定位模型的 3 维，2 维以及 1 维精度准则，损失越小精度越高。

表 5-7 距离损失表

维度	距离损失	数值（单位：mm）
1 维	$\Delta x_{\max}$	210.94
	$\Delta y_{\max}$	204.20
	$\Delta z_{\max}$	123.21
2 维	$\max\left(\sqrt{(x_i-\bar{x})^2+(y_i-\bar{y})^2}\right)$	207.60
	$\max\left(\sqrt{(x_i-\bar{x})^2+(z_i-\bar{z})^2}\right)$	172.74
	$\max\left(\sqrt{(y_i-\bar{y})^2+(z_i-\bar{z})^2}\right)$	168.64
3 维	$\max\left(\sqrt{(x_i-\bar{x})^2+(y_i-\bar{y})^2+(z_i-\bar{z})^2}\right)$	183.83

（2）模型评估

无干扰定位模型损失情况如图 5-5 所示：

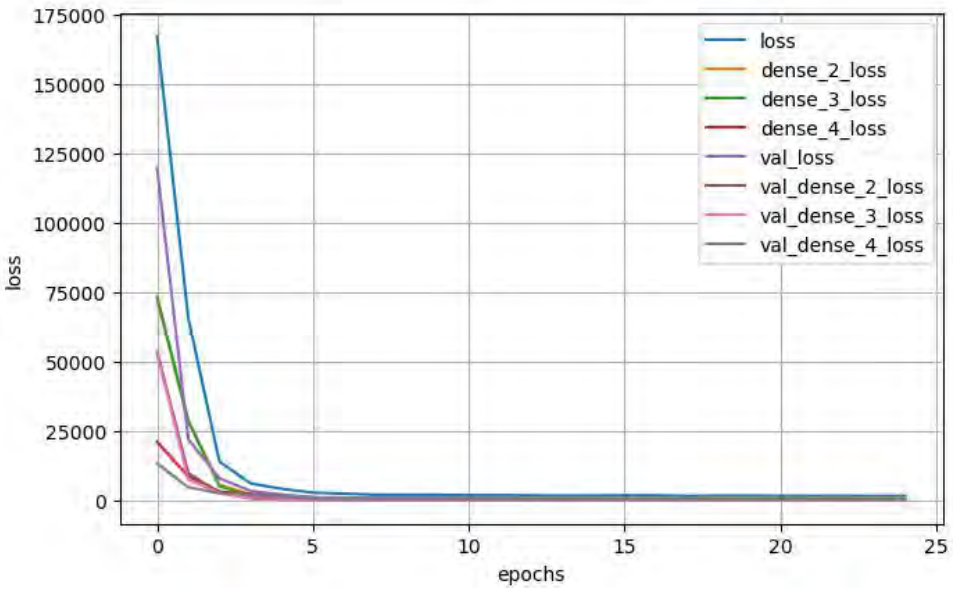


图 5-5 无干扰定位模型损失图

有干扰定位模型损失情况如图 5-6 所示：

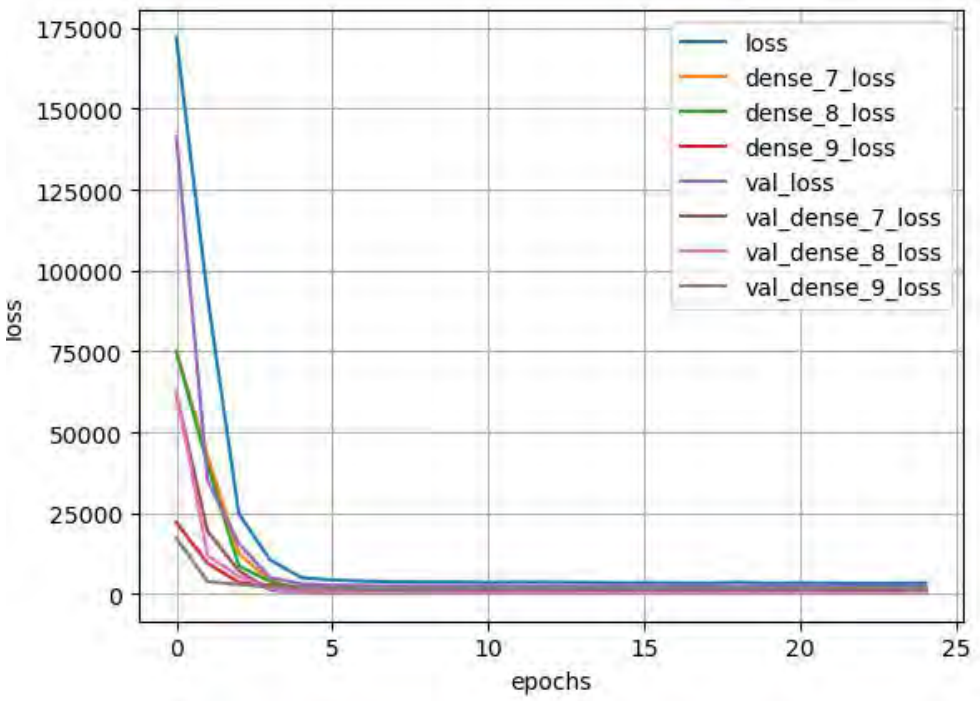


图 5-6 无干扰定位模型损失图

loss：模型训练过程中的训练集总的损失值

dense_n_loss：模型训练过程中训练集的各个输出通道的损失值

val_loss：模型训练过程中的测试集总的损失值

val_n_loss：模型训练过程中的测试集各个输出通道的损失值

采用 MSE 得到的 loss 值，loss 越小，模型拟合效果越好。根据上图分析得知：两个模型均在第 5 轮迭代时损失值降到很小，到 20 轮迭代时 loss 基本降到了 100 左右。

5.3 任务三模型建立与求解

5.3.1 任务三问题分析

不同于之前的问题，任务三中的场景做了改变，对定位模型应用场景适用性的考验，锚点的位置和靶点的范围均发生了变化，所给数据分为有干扰与无干扰数据。该问题的解决首先需要改变初始数据的输入，然后再将数据输入到对应的定位模型即可预测出靶点的位置。可以在建立任务二模型时直接考虑到实验场景信息，以降低工作量。

5.3.2 模型数据准备

（1）附件 3 的数据

包含正常数据与异常数据两个文件，每个文件中含有 20 条数据。

（2）场景 2 的数据

表 5-8 场景 2 数据表

靶点（Tag）范围	5000mm×3000mm×3000mm	
锚点（anchor）位置（单位：mm）	A_0（0，0，1200）	A_1（5000，0，1600）
	A_2（0，3000，1600）	A_3（5000，3000，1200）

（3）定位模型

任务二得到的无干扰定位模型与有干扰定位模型。

5.3.3 模型求解过程

（1）数据预处理

将附件 3 中的数据按照任务一的处理方式处理为“时间戳 d_0 d_1 d_2 d_3”的形式。

（2）最小二乘得到初始位置

将场景 2 的锚点坐标与（1）中的数据输入最小二乘模型中得到初始的低精度位置信息，由于数据分为正常数据与异常数据，所以得到两组位置数据。

（3）定位模型求取高精度定位

分别使用无干扰与有干扰定位模型分别对（2）中求取的两组位置数据求解，获得两组高精度靶点位置信息数据。

5.3.4 模型的求解

使用无干扰定位模型对附件 3 中的无干扰信号求解，最终的定位结果如表 5-9 所示：

表 5-9 附加 3 无干扰信号求解结果

序号	X/mm	Y/mm	Z/mm
1	3699	2196	1110
2	4247	1771	1156
3	3192	1733	1086
4	2566	1889	1540
5	743	394	1396

使用有干扰定位模型对附件 3 中的有干扰信号求解，最终的定位结果如表 5-10 所示：

表 5-10 附件 3 有干扰信号求解结果

序号	X/mm	Y/mm	Z/mm
1	4647	2114	1529
2	2456	1622	1490
3	1729	1358	1535
4	2001	922	1515
5	1212	869	1553

5.4 任务四模型建立与求解

5.4.1 任务四问题分析

该问题要求将无干扰数据与有干扰数据进行区分，不同于之前给出的数据，附件 4 中的数据并没有告知数据类型，但是可以先使用任务一中已知分类的数据进行学习训练分类模型，然后再利用训练好的分类模型将附件 4 中的数据进行分类。使用单个分类模型，如 KNN（K-NearestNeighbor）或决策回归树模型时，可能存在人为误差与计算误差，单一的分类模型可能不能很好的适用于本研究，故需要综合性考虑，建立模型验证指标，进行模型的准确性分析，最终改进或综合模型以找到最适应本研究的模型。

因此本文采用集成学习的思想，采用多个分类模型进行训练，最后将它们的分类结果投票选取，得出最优的分类结果。

5.4.2 模型数据准备

（1）任务一处理得到的数据

分为多个“正常.xls”文件与多个“异常.xls”数据文件。

（2）附件 4 提供的数据

包含 40 条原始数据，不区分有无信号干扰。

5.4.3 分类模型建立

（1）数据预处理：

把附件 4 中的数据处理为同任务一结果数据相同的格式，方便下面调用预测结果；

（2）创建训练数据集：

获取任务一处理后的所有文件，将它们整合为一个数据文件，取d_0-d_3列的数据作为训练集的特征值。由于已经知道它们是否有无信号干扰，所以手动创建每条数据对应的标签值。无信号干扰用 0 表示，有信号干扰用 1 表示。创建好的数据集部分如表 5-11 所示：

表 5-11 分类模型训练集

数据编号	特征值（d_0，d_1，d_2，d_3）	标签值
1	[760，4550，4550，6300]	0
2	[770，4550，4550，6300]	0
.....		
n	[930，4100，4530，6530]	1

（3）分类模型训练：

常见的分类模型有很多，如 KNN，SVM，RF 等，本文采用多种分类模型对数据进行分类训练。由于每个模型都有自己的优缺点，单一分类模型预测的分类结果可能并不令人信服，故采用集成学习的思想，使用多个分类模型进行分类，最后对多个结果采取多数投票原则，选取最“可靠”的结果做为最终的分类结果。

表 5-12 常见分类模型优缺点

模型名称	优点	缺点
KNN	简单有效，重新训练代价低	不适合小样本数据集，样本不平衡时分类结果较差
BP 神经网络	准确度高，并行处理能力强	需要大量参数，过拟合可能严重，输出结果难以解释
RandomForest（随机森林）[9]	泛化能力强，可以平衡误差	数据噪声太大时容易过拟合，数据属性取值划分较多的对模型影响较大

（4）分类模型预测：

分类模型训练好后，将附件 4 中预处理好的数据直接导入分类模型得出结果。

5.4.4 模型的求解与评估分析

（1）结果展示

用分类模型对附件 4 中的 10 条数据进行分类预测，得到的分类结果部分如表 5-13 所示：

表 5-13 附件 4 数据分类预测表

数据编号	时间戳(ms)	d_0(mm)	d_1(mm)	d_2(mm)	d_3(mm)	信号受干扰情况
数据 1	055943544	2940	4290	2840	4190	无干扰
数据 2	060341974	5240	5360	2040	2940	有干扰
数据 3	060548859	4800	2610	4750	2550	无干扰
数据 4	060452358	5010	4120	3810	2020	有干扰
数据 5	053906604	2840	4490	2860	4190	无干扰
数据 6	053627102	5010	5320	1990	2930	有干扰
数据 7	053524347	5050	3740	3710	2070	有干扰
数据 8	055501490	5050	4110	3710	2110	有干扰
数据 9	054658330	4840	2600	4960	2700	有干扰
数据 10	055551320	2740	2720	4670	4790	有干扰

（2）模型评估

分类模型预测分类评估报告（classification_report）如表 5-14 所示：

表 5-14 分类模型评估报告表

Classification	Precision	Recall	F1-score	Support
0	0.97	1.00	0.99	2421
1	1.00	0.97	0.98	2439
Accuracy	-	-	0.98	4860
Macro avg	0.99	0.99	0.98	4860
Weighted avg	0.99	0.98	0.98	4860

说明：

无信号干扰用 0 表示，有信号干扰用 1 表示;

Support 中 2421 表示测试集中类别为 0 的数据量，2439 是类别为 1 的数据量；

Macro avg 表示宏平均，是所有类的 F1 值的算术平均。

5.5 任务五模型建立与求解

5.5.1 任务五问题分析

任务五利用静态点的定位模型和靶点运动规律，来建立动态轨迹预测模型。附件 5 的数据为动态靶点在实验场景 1 下的采集数据，首先要分析数据的特点。其数据按时间戳排列，每个样本间的时间差、数据差，可以大致判断靶点运动的速度情况接近匀速。假设靶点为匀速运动。根据计算时间戳之差，发现两条记录之间的时间差$\Delta T \approx 200\text{ms}$，故可将采样时间确定为$T = 0.2\text{s}$，但有三处为不同的情况，如表 5-15 所示：

表 5-15 时间戳之差特殊情况表

序号	时间戳	d_0	d_1	d_2	d_3	数据编号	时间戳之差ΔT
1	110959934	2420	4630	2950	4970	23203	40210
	111000144	2440	4620	2930	4970	63413	
2	111059813	3640	1550	5630	4570	123082	40210
	111100023	3630	1560	5620	4570	163292	
3	111127612	6360	4450	4500	870	190881	419
	111128031	6380	4460	4500	850	191300	

可以发现序号 1、2 的两处锚点到靶点所测量出的距离相近，各出现了 40s 的停留，将 1、2 两处靶点移动的时间定为采样时间 T，为方便起见将数据以 1、2 处为界分为三段。在序号 3 处两条记录时间戳之差为 419ms，需在两条记录间线性差值补充一条记录。如表 5-16 所示：

表 5-16 记录插值表

序号	时间戳	d_0	d_1	d_2	d_3	数据编号	时间戳之差ΔT
3	111127612	6360	4450	4500	870	190881	-
	111127821	6370	4455	4500	860	-	209
	111128031	6380	4460	4500	850	191300	210

该任务给出了在场景 1 中，靶点不断运动采集到的数据，这些数据有的是无干扰信号，有的是有干扰信号数据，所以需要先使用分类模型判断出数据的类别，然后再使用定位模型定位靶点位置，最后根据定位画出运动轨迹。

卡尔曼滤波具有良好的动态跟踪定位的效果，扩展卡尔曼滤波可以将非线性系统近似线性化，对于该任务扩展卡尔曼滤波较为适用。

5.5.2 模型数据准备

（1）附件 5 动态轨迹数据

需要附件 5 原始数据处理为同任务一格式的数据。

（2）任务二得到的定位模型

信号无干扰定位模型和有干扰定位模型。

（3）任务四得到的分类模型

5.5.3 模型建立

（1）运用 EKF 建立模型

靶点在$k-1$时刻的位置设为$(x_{k-1}，\ y_{k-1}，\ z_{k-1})$，由采样时间 T，则靶点在k时刻的位置可以联立方程组得到（5-4）式，状态变量由位置坐标与速度组成，其中$x_{k,k-1}$，$y_{k,k-1}$，$z_{k,k-1}$，$\dot{x}_{k,k-1}$，$\dot{y}_{k,k-1}$，$\dot{z}_{k,k-1}$分别为k时刻的位置坐标及速度，w_k为服从$N(0，\ Q_k)$多元正态分布的过程噪声。

$$\begin{pmatrix} x_{k,k-1} \\ y_{k,k-1} \\ z_{k,k-1} \\ \dot{x}_{k,k-1} \\ \dot{y}_{k,k-1} \\ \dot{z}_{k,k-1} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & T & 0 & 0 \\ 0 & 1 & 0 & 0 & T & 0 \\ 0 & 0 & 1 & 0 & 0 & T \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_{k-1,k-1} \\ y_{k-1,k-1} \\ z_{k-1,k-1} \\ \dot{x}_{k-1,k-1} \\ \dot{y}_{k-1,k-1} \\ \dot{z}_{k-1,k-1} \end{pmatrix} + w_k \qquad (5\text{-}4)$$

根据锚点到靶点的距离方程可建立观测方程为：

$$\begin{pmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{pmatrix} = \begin{pmatrix} \sqrt{(x-x_0)^2+(y-y_0)^2+(z-z_0)^2} \\ \sqrt{(x-x_1)^2+(y-y_1)^2+(z-z_1)^2} \\ \sqrt{(x-x_2)^2+(y-y_2)^2+(z-z_2)^2} \\ \sqrt{(x-x_3)^2+(y-y_3)^2+(z-z_3)^2} \end{pmatrix} + v_k \qquad (5\text{-}5)$$

其中v_{k}为服从 $\mathrm{N}(0，\ R_{\mathrm{k}})$高斯白噪声，$R_{\mathrm{k}}$为观测量的方差，即各测量距离的方差。

为将非线性方差线性化，需要求得相应的雅可比矩阵：

$$H_k=\begin{pmatrix}\frac{x_{k,k-1}-x_0}{d_{k,k-1,0}} & \frac{y_{k,k-1}-y_0}{d_{k,k-1,0}} & \frac{z_{k,k-1}-z_0}{d_{k,k-1,0}} & 0 & 0 & 0\\ \frac{x_{k,k-1}-x_1}{d_{k,k-1,1}} & \frac{y_{k,k-1}-y_1}{d_{k,k-1,1}} & \frac{z_{k,k-1}-z_1}{d_{k,k-1,1}} & 0 & 0 & 0\\ \frac{x_{k,k-1}-x_2}{d_{k,k-1,2}} & \frac{y_{k,k-1}-y_2}{d_{k,k-1,2}} & \frac{z_{k,k-1}-z_2}{d_{k,k-1,2}} & 0 & 0 & 0\\ \frac{x_{k,k-1}-x_3}{d_{k,k-1,3}} & y\frac{x_{k,k-1}-y_3}{d_{k,k-1,3}} & \frac{z_{k,k-1}-z_3}{d_{k,k-1,3}} & 0 & 0 & 0\end{pmatrix} \quad (5\text{-}6)$$

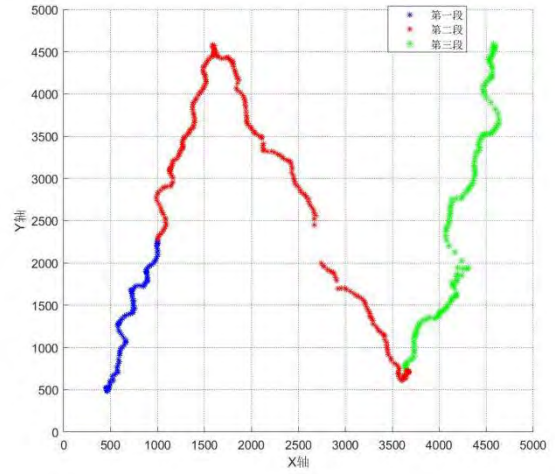
对状态变量X_0、协方差矩阵$P_{0,0}$进行初始化，输入观测值与锚点坐标于扩展卡尔曼滤波器进行迭代计算，可以求得每个时刻的靶点坐标$(x，y，z)$.

（2）BP 定位模型与分类模型

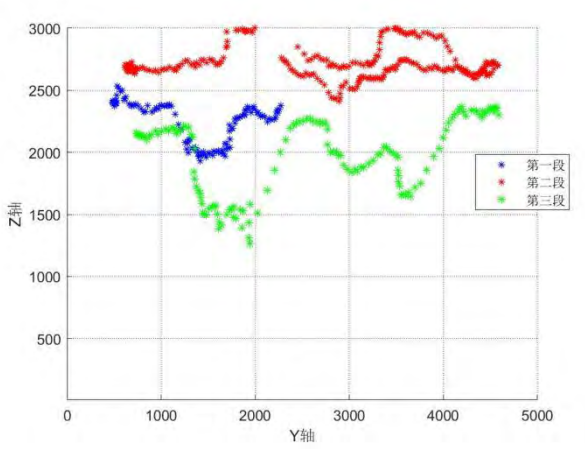
在 EKF 预测得到运动轨迹后，结果仍不理想，为了提高精度，在此基础上继续接入 BP 定位模型与定位模型进行运动轨迹预测，三者结合精度得到提高。

5.5.4 模型的求解

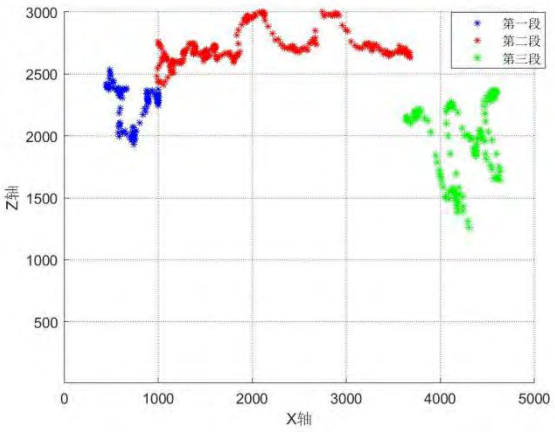
（1）扩展卡尔曼滤波预测结果



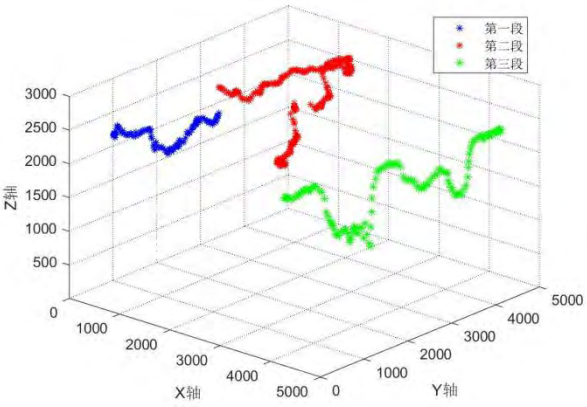
（a）运动轨迹 XOY 平面截图



（b）运动轨迹 YOZ 平面截图



（c）运动轨迹 XOZ 平面截图

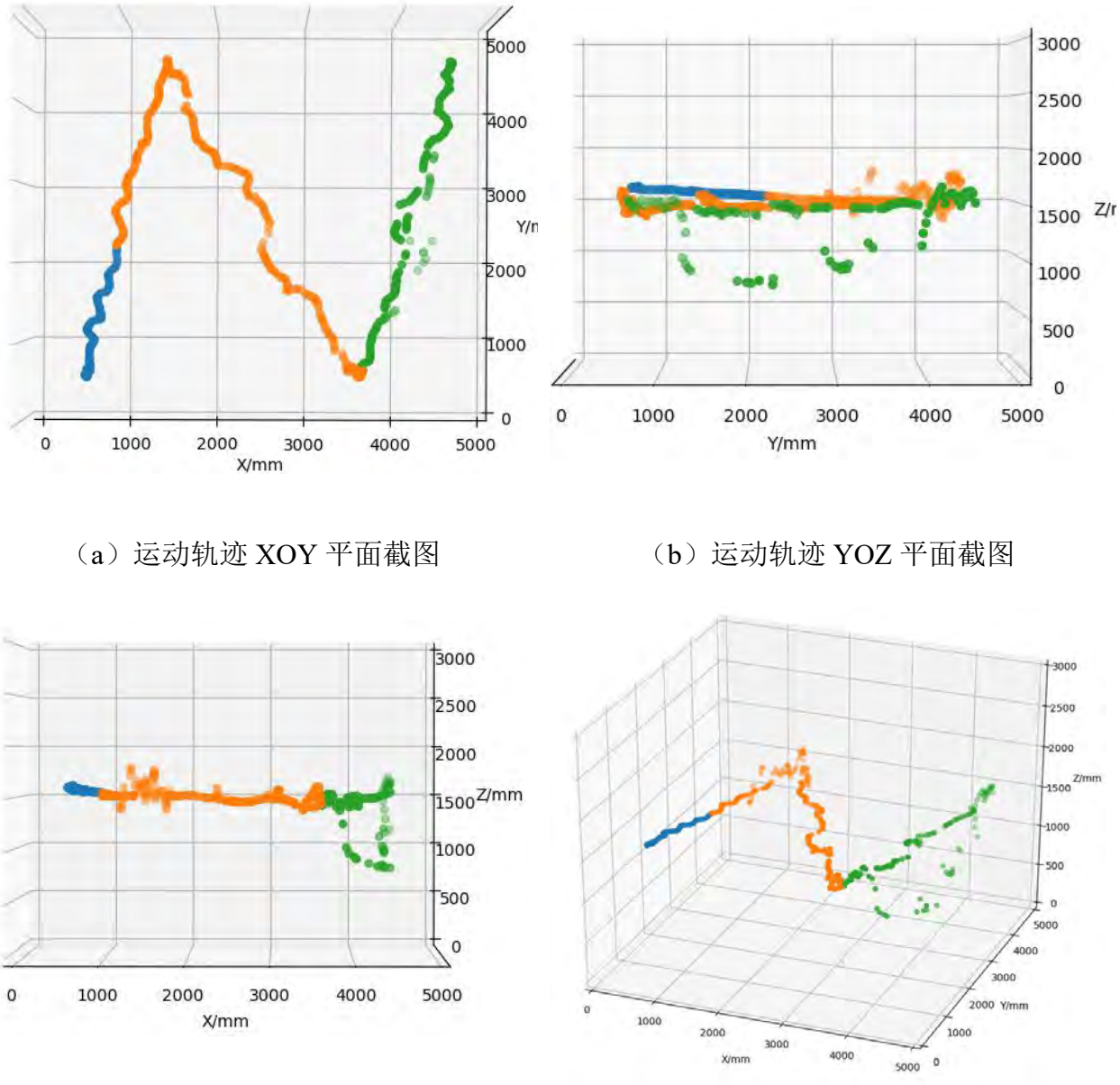


（d）运动轨迹立体图

图 5-7　EKF 运动轨迹预测结果

基于扩展卡尔曼滤波的轨迹预测模型，在 XOY 平面较为稳定，Z 轴数据波动较大，效果较差。

（2）BP 神经网络优化结果



（a）运动轨迹 XOY 平面截图　　（b）运动轨迹 YOZ 平面截图

（c）运动轨迹 XOZ 平面截图　　（d）运动轨迹立体图

图 5-8　BP 模型优化运动轨迹预测结果

根据上述 4 个图，抛去少数异常点分析得知，靶点运动轨迹大致呈现一个‘Z’字型，运动高度基本为 1500mm 左右，蓝色为起始段，绿色为停止段。

假设每一段运动为匀速运动，结合数据的时间戳分析得知，蓝色时间段的运动速率为 70mm/s，黄色时间段的运动速率为 51mm/s，绿色时间段的运动速率为 138mm/s，在蓝黄交界处与黄绿交界处，靶点都大约停留了 40s 左右。绿色时间段系统受干扰影响最为强烈。

根据上述 4 个图，抛去少数异常点分析得知，靶点运动轨迹大致呈现一个‘Z’字型，运动高度基本为 1500mm 左右，蓝色为起始段，起点为（632mm，622mm，1613mm），绿色为停止段，终点为（4521mm，4514mm，1529mm）。

假设每一段运动为匀速运动，结合数据的时间戳分析得知，蓝色时间段的运动速率为 70mm/s，黄色时间段的运动速率为 51mm/s，绿色时间段的运动速率为 138mm/s，在蓝黄交界处（950mm，220mm，1540mm）与黄绿交界处（3600mm，668mm，1541mm），靶点都大约停留了 40s 左右。

6. 总结

6.1 模型优点

（1）本文综合使用基本定位模型与 BP 神经网络，结合多个模型，相互优化，能够较为精确预测静态靶点坐标。

（2）本文在建立分类模型时，综合考虑各模型的优缺点，采用集成学习的方式将基模型合理的结合集成综合性模型，大大提高了分类的准确性。

（3）本文在建立动态轨迹定位时在选用经典的扩展卡尔曼滤波模型的基础上，使用 BP 神经网络优化，综合模型的优点，能较为准确获得动态定位轨迹。

6.2 模型缺点

由于时间限制，本文未能试用更多经典的 UWB 室内高精度定位抗 NLOS 的模型，比如 Chan 定位法、Taylor 级数展开法、基于 LSTM 网络模型等，当遇到靶点复杂运动时，模型求解难度比较大。

6.3 总结与推广

UWB 室内定位最主要是解决 NLOS 误差以及环境噪声，本文建立的模型方法，旨在综合考虑问题，其主要思想可以为该研究提供一些启发。

参考文献

[1] 王佩，高凯.UWB 定位技术分析[J]. 中国科技信息.2020,(17)：67-68.

[2] Mazraani R， Saez M， Govoni L，et al. Experimental results of a combined TDOA/TOF technique for UWB based localization systems[C]// 2017 IEEE International Conference on Communications Workshops (ICC Workshops). IEEE, 2017.

[3] 柏宇轩.Kmeans 应用与特征选择[J]. 电子制作.2017,(23)：55-56，63.[1]陈伽洛，陈龙然.决策树与随机森林[J]. 信息与电脑(理论版).2019,31(17)：43-45.

[4] 王赟.基于 BP 神经网络的数字识别探究[J]. 电脑编程技巧与维护.2021,(6)：127-128.

[5] 丁滔.基于 BP 神经网络的车距测量方法[J]. 装备制造技术.2021,(6)：37-39，59.

[6] 闫保芳,毛庆洲.一种基于卡尔曼滤波的超宽带室内定位算法[J].传感器与微系统,2017,36(10):137-140+143.

[7] Daley, Roger. Estimating the Wind Field from Chemical Constituent Observations: Experiments with a One-Dimensional Extended Kalman Filter[J]. Monthly Weather Review, 1995, 123(1):181.

[8] 郑飞，郑继禹. 基于 TDOA 的 CHAN 算法在 UWB 系统 LOS 和 NLOS 环境中的应用研究[J]. 电子技术应用，2007(11):110-113.

[9] 孙明喆，毕瑶家，孙驰.改进随机森林算法综述[J]. 现代信息科技.2019,3(20)：28-30.

附　录

附录一：数据预处理列举文件

注：其他数据及程序源码文件详见上传附件

（1）“24.正常.txt”处理后数据

时间戳(ms)	d_0(mm)	d_1(mm)	d_2(mm)	d_3(mm)
094843811	3280	4660	2600	3910
094844019	3280	4670	2600	3920
094844643	3280	4660	2600	3900
094844851	3280	4660	2610	3920
094845267	3280	4660	2610	3910
094845891	3290	4660	2600	3920
094846499	3300	4650	2590	3920
094846706	3290	4660	2590	3910
094847122	3280	4660	2590	3890
094847330	3280	4650	2590	3880
094847953	3280	4660	2590	3920
094848162	3280	4660	2590	3910
094848371	3270	4670	2600	3900
094848578	3270	4660	2600	3910
094849394	3290	4660	2610	3910
094851473	3280	4670	2610	3930
094853953	3270	4660	2590	3880
094854161	3270	4650	2590	3890
094854784	3270	4670	2590	3900
094854993	3270	4680	2590	3900
094856224	3280	4650	2590	3900
094856848	3280	4640	2600	3900
094857056	3270	4660	2600	3890

094902655	3270	4670	2590	3920
094903071	3270	4660	2590	3940
094908446	3270	4670	2600	3910
094908861	3280	4670	2600	3890
094910301	3290	4660	2610	3900
094910925	3300	4650	2610	3900
094912796	3300	4660	2610	3920

（2）“109.正常.txt”处理后数据

时间戳(ms)	d_0(mm)	d_1(mm)	d_2(mm)	d_3(mm)
144156086	4910	5310	2030	2870
144156294	4920	5310	2020	2870
144156710	4920	5310	2020	2880
144157750	4890	5310	2020	2880
144157958	4880	5310	2020	2880
144158166	4890	5310	2020	2870
144159829	4880	5320	2020	2860
144200245	4860	5320	2020	2870
144201077	4860	5320	2030	2870
144203157	4890	5330	2030	2880
144203572	4910	5320	2020	2880
144204196	4910	5320	2040	2880
144208979	4890	5320	2020	2880

144209811	4890	5320	2010	2880
144210435	4910	5320	2010	2880
144212099	4910	5330	2030	2880
144213762	4880	5340	2010	2880
144214386	4880	5330	2040	2880
144215009	4870	5330	2050	2870
144217090	4860	5320	2030	2890
144217713	4890	5320	2040	2890
144218753	4880	5330	2040	2870
144222703	4910	5340	2010	2890
144224784	4920	5330	2030	2870
144226863	4910	5340	2040	2880
144227487	4920	5340	2040	2890
144232894	4910	5340	2060	2880
144233726	4890	5340	2040	2890
144235391	4910	5330	2040	2900
144244123	4930	5340	2020	2880

（3）“1.异常.txt”处理后数据

时间戳(ms)	d_0(mm)	d_1(mm)	d_2(mm)	d_3(mm)
090531088	1280	4550	4550	6300
090531296	1260	4550	4550	6300
090531711	1240	4550	4550	6300
090532335	1230	4550	4550	6300
090532751	1290	4550	4550	6300
090533983	1250	4550	4560	6300
090534399	1230	4550	4570	6300
090534814	1280	4550	4560	6300
090539774	1220	4540	4550	6300
090540605	1270	4540	4550	6310
090541229	1210	4540	4560	6300
090541853	1200	4550	4550	6300
090545996	1270	4550	4550	6290
090609799	770	5040	4540	6300
090610007	770	5000	4540	6290
090610215	760	5060	4540	6290
090610422	770	5030	4540	6290
090610822	770	5070	4550	6290
090611030	760	5050	4550	6290

090611862	780	5030	4550	6300
090613942	760	5010	4550	6300
090617045	760	5030	4550	6300
090617653	760	5030	4550	6310
090618069	750	5070	4550	6310
090621588	770	5070	4550	6310
090621796	760	5050	4540	6310
090623875	780	5000	4550	6310
090624083	770	5000	4550	6300
090626563	780	5050	4550	6300
090640431	780	5010	4550	6280

（4）“100.异常.txt”处理后数据

时间戳(ms)	d_0(mm)	d_1(mm)	d_2(mm)	d_3(mm)
115709181	1510	3750	4690	5710
115709389	1520	3720	4680	5720
115710413	1510	3730	4680	5710
115710621	1510	3720	4680	5710
115710829	1520	3750	4680	5710
115711037	1530	3760	4690	5700
115711453	1510	3770	4690	5700

115712493	1510	3790	4680	5700
115712701	1510	3820	4680	5700
115714348	1520	3860	4680	5710
115718492	1490	3780	4670	5700
115726362	1510	3700	4670	5700
115750422	1510	3560	5290	5710
115750630	1510	3560	5230	5710
115750839	1510	3560	5190	5700
115751046	1510	3560	5110	5700
115751254	1510	3560	5060	5710
115751461	1510	3560	5010	5710
115751670	1510	3560	4970	5710
115752086	1510	3550	4950	5710
115752710	1510	3560	4930	5700
115752918	1510	3560	4910	5700
115753126	1520	3560	4890	5700
115753333	1520	3560	4870	5700
115753542	1520	3560	4860	5710
115753750	1520	3560	4850	5710
115754789	1510	3560	5030	5710
115754981	1510	3560	5140	5710

115758725	1520	3560	4910	5690
115806212	1510	3560	4810	5700

附录二：数据处理程序

```
import pandas as pd
import numpy as np
import os.path
from sklearn.cluster import KMeans
# 遍历读取文件
path_yc = './datas/附件 1：UWB 数据集/异常数据/'
path_zc = './datas/附件 1：UWB 数据集/正常数据/'
# 所有正常异常文件的名字
zcs = os.listdir(path_zc)
ycs = os.listdir(path_yc)
def normfun(x, mu, sigma):
    return np.exp(-((x - mu) ** 2) / (2 * (sigma ** 2))) / (sigma * np.sqrt(2 *
np.pi))
def dropEx(a, index_ex):
    mean = a.mean()
    std = a.std()
    low = mean - 3 * std
    high = mean + 3 * std
    index_lex = np.where(a <= low)[0]
    index_hex = np.where(a >= high)[0]
    index_ex.append(index_lex)
    index_ex.append(index_hex)
# 处理每个文件的方法：去重，去相似，处理异常值
def fixdata(path):
    file_num = path.split('/')[-1].split('.')[0]
    file_type = path.split('/')[-1].split('.')[1]
    data = pd.read_table(path)
    # 1.检查有没有缺失异常，有缺失时填充该行
    if data.isnull().any()[0] == True:
        print(path + '=====有缺失')
        data.fillna(axis=0, method='backfill')
    # 2.创建空 df
    hs = data.shape[0]
    lieming = ['时间戳', 'D0', 'D1', 'D2', 'D3']
    df = pd.DataFrame(columns=lieming)
    # 3.将每组数据的 s0,s1,s2,s3 放置到同一行
    mindex = 0
```

```
    for i in range(0, hs, 4):
        RR0 = data.iloc[i, 0]
        RR0S = RR0.split(':')
        RR1 = data.iloc[i + 1, 0]
        RR1S = RR1.split(':')
        RR2 = data.iloc[i + 2, 0]
        RR2S = RR2.split(':')
        RR3 = data.iloc[i + 3, 0]
        RR3S = RR3.split(':')
        # 2.获取每组的四个距离
        T = RR0S[1]
        s0 = int(RR0S[5])
        s1 = int(RR1S[5])
        s2 = int(RR2S[5])
        s3 = int(RR3S[5])
        df.loc[mindex, lieming] = [T, s0, s1, s2, s3]
        mindex = mindex + 1
    # 5.去异常
    a0 = np.array(df['D0'])
    a1 = np.array(df['D1'])
    a2 = np.array(df['D2'])
    a3 = np.array(df['D3'])
    index_ex = []   # 要去除的下标
    dropEx(a0, index_ex)
    dropEx(a1, index_ex)
    dropEx(a2, index_ex)
    dropEx(a3, index_ex)
    # 遍历 index_ex 元素来进行删除异常值，由于可能有重复的
    # 需要二次保存要去除的下标
    index_ex2 = []
    for i in index_ex:
        for rindex in i:
            if rindex not in index_ex2:
                index_ex2.append(rindex)
    # 去重复保留第一条
    index_dup = np.where(df.iloc[:, 1:].duplicated())[0]
    for i in index_dup:
        if i not in index_ex2:
            index_ex2.append(i)
    df = df.drop(index_ex2, axis=0)
    # 重置索引，方便下面取 30 条数据
    df.index = range(len(df))
    # 6.k-means 去相似
    n = 30
    if len(df) < 30:
        n = len(df)
    km = KMeans(n_clusters=n)
    x_data = df.iloc[:, 1:]
```

```
        y = km.fit_predict(x_data)
        # 将 numpy 数组转为普通数组
        yy = []
        for i in y:
            yy.append(i)
        # 保留最后 n 条数据
        index_ex3 = []
        save_index = []
        for i in range(n):
            save_index.append(yy.index(i))
        for i in range(len(df)):
            if i not in save_index:
                index_ex3.append(i)
        df = df.drop(index_ex3, axis=0)
        # 7.输出最后文件
        df.to_excel('./newdatas/' + file_type + '数据/' + file_num + '.' + file_type +
'.xls', index=False)
        return df
# 修改后的正常数据与异常数据的汇总
lieming = ['时间戳', 'D0', 'D1', 'D2', 'D3']
zcdf = pd.DataFrame(columns=lieming)
ycdf = pd.DataFrame(columns=lieming)
for txt in zcs:
        file_path = path_zc + txt
        df = fixdata(file_path)
        zcdf = zcdf.append(df)
for txt in ycs:
        file_path = path_yc + txt
        df = fixdata(file_path)
        ycdf = ycdf.append(df)
zcdf.to_csv('./newdatas/正常数据.csv', index=False)
ycdf.to_csv('./newdatas/异常数据.csv', index=False)
```

附录三：任务二求解

1、无干扰定位模型和有干扰定位模型

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os.path
import tensorflow as tf
from sklearn.model_selection import train_test_split
# 1.将所有正常与异常文件的路径合并成集合
zcspath = './newdatas/正常数据/'
ycspath = './newdatas/异常数据/'
```

```
zcfilesname = os.listdir(zcspath)
ycfilesname = os.listdir(ycspath)
zcfilespath = []
ycfilespath = []
for name in zcfilesname:
    zcfilespath.append(zcspath + name)
for name in ycfilesname:
    ycfilespath.append(ycspath + name)
# 2.使用最小二乘法得到初始(x,y,z)
#  输入 A0,A1,A2,A3，以及正异常 D 的值
#  场景 1 的锚点位置
A0 = (0, 0, 1300)
A1 = (5000, 0, 1700)
A2 = (0, 5000, 1700)
A3 = (5000, 5000, 1300)
x0 = A0[0]
y0 = A0[1]
z0 = A0[2]
x1 = A1[0]
y1 = A1[1]
z1 = A1[2]
x2 = A2[0]
y2 = A2[1]
z2 = A2[2]
x3 = A3[0]
y3 = A3[1]
z3 = A3[2]
#  最小二乘法
A = [[x0 - x3, y0 - y3, z0 - z3],
     [x1 - x3, y1 - y3, z1 - z3],
     [x2 - x3, y2 - y3, z2 - z3]]
A = np.multiply(-2, A)
def getRes(d0, d1, d2, d3):
    b = np.array([[d0 ** 2 - d3 ** 2 + x3 ** 2 + y3 ** 2 + z3 ** 2 - x0 ** 2 - y0 ** 2 - z0 ** 2],
                  [d1 ** 2 - d3 ** 2 + x3 ** 2 + y3 ** 2 + z3 ** 2 - x1 ** 2 - y1 ** 2 - z1
** 2],
                  [d2 ** 2 - d3 ** 2 + x3 ** 2 + y3 ** 2 + z3 ** 2 - x2 ** 2 - y2 ** 2 - z2
** 2]])
    x, y, z = np.dot((np.dot((np.linalg.inv(np.dot(A.T, A))), A.T)), b)
    # x, y, z = np.dot(np.linalg.inv(A), b)
    return x[0], y[0], z[0]
def getLoc(path):
    data = pd.read_excel(path)
```

```
        # 获取每个文件的数据
        counts = data.shape[0]
        d = []
        loc = []
        for i in range(counts):
            [d0, d1, d2, d3] = data.iloc[i, 1:]
            d.append([d0, d1, d2, d3])

        for dlist in d:
            x, y, z = getRes(dlist[0], dlist[1], dlist[2], dlist[3])
            loc.append([x, y, z])
        loc = np.array(loc)
        xx = range(counts)
        x = loc[:, 0]
        y = loc[:, 1]
        z = loc[:, 2]
        return np.array([x, y, z]).T
# 3.获取正常与异常数据根据最小二乘得到的(x,y,z)
tagData = {....}
zcfirstxyz = []
zcrealxyz = []
ycfirstxyz = []
ycrealxyz = []
for path in zcfilespath:
        fileNum = int(path.split('/')[-1].split('.')[0])
        locs = getLoc(path) #  获取初始坐标预测结果
        num = locs.shape[0] #  该 Tag 坐标组内的坐标数量
        # 将获取到的坐标信息添加到总集合
        for loc in locs:
            zcfirstxyz.append(loc)
        # 创建对应的标签值
        for i in range(num):
            zcrealxyz.append(tagData[fileNum])
for path in ycfilespath:
        fileNum = int(path.split('/')[-1].split('.')[0])
        locs = getLoc(path) #  获取初始坐标预测结果
        num = locs.shape[0] #  该 Tag 坐标组内的坐标数量
        # 将获取到的坐标信息添加到总集合
        for loc in locs:
            ycfirstxyz.append(loc)
        # 创建对应的标签值
        for i in range(num):
            ycrealxyz.append(tagData[fileNum])
```

```
# 4.创建 BP 模型
def getModel():
    input = tf.keras.layers.Input(shape=[3])
    hidden1 = tf.keras.layers.Dense(30, activation='relu')(input)
    bn = tf.keras.layers.BatchNormalization()(hidden1)
    hidden2 = tf.keras.layers.Dense(30, activation='relu')(bn)
    output1 = tf.keras.layers.Dense(1)(hidden2)
    output2 = tf.keras.layers.Dense(1)(hidden2)
    output3 = tf.keras.layers.Dense(1)(hidden2)
    model = tf.keras.models.Model(inputs=input, outputs=[output1, output2, output3])
    model.compile(loss='mse', optimizer='adam')
    return model
def lossView(h):
    pd.DataFrame(h.history).plot()
    plt.grid(True)
    plt.xlabel('epochs')
    plt.ylabel('loss')
    plt.show()
# 5 创建正常与异常数据的模型训练集
zcfirstxyz = np.array(zcfirstxyz)
zcrealxyz = np.array(zcrealxyz)
ycfirstxyz = np.array(ycfirstxyz)
ycrealxyz = np.array(ycrealxyz)
zcx_train, zcx_test, zcy_train, zcy_test = train_test_split(zcfirstxyz, zcrealxyz)
ycx_train, ycx_test, ycy_train, ycy_test = train_test_split(ycfirstxyz, ycrealxyz)
zcy_train0 = zcy_train[:,0]
zcy_train1 = zcy_train[:,1]
zcy_train2 = zcy_train[:,2]
zcy_test0 = zcy_test[:,0]
zcy_test1 = zcy_test[:,1]
zcy_test2 = zcy_test[:,2]
ycy_train0 = ycy_train[:,0]
ycy_train1 = ycy_train[:,1]
ycy_train2 = ycy_train[:,2]
ycy_test0 = ycy_test[:,0]
ycy_test1 = ycy_test[:,1]
ycy_test2 = ycy_test[:,2]
zcmodel= getModel()
ycmodel= getModel()
zch                                                                                    =
zcmodel.fit(zcx_train,[zcy_train0,zcy_train1,zcy_train2],epochs=25,verbose=1,validation_data=(
zcx_test,[zcy_test0,zcy_test1,zcy_test2]))
ych                                                                                    =
```

```
ycmodel.fit(ycx_train,[ycy_train0,ycy_train1,ycy_train2],epochs=25,verbose=1,validation_data=
(ycx_test,[ycy_test0,ycy_test1,ycy_test2]))
lossView(zch)
lossView(ych)
print(zcmodel.metrics_names)
print(zcmodel.evaluate(zcx_test, [zcy_test0, zcy_test1, zcy_test2]))
print(ycmodel.evaluate(ycx_test, [ycy_test0, ycy_test1, ycy_test2]))
#
zcmodel.save('./mymodel/task2zc.h5')
ycmodel.save('./mymodel/task2yc.h5')
```

2. 附件二求解

```
import tensorflow as tf
import numpy as np
import pandas as pd
def fixdata(file_type, path):
    data = pd.read_table(path, header=None)
    hs = data.shape[0]
    lieming = ['时间戳', 'D0', 'D1', 'D2', 'D3']
    df = pd.DataFrame(columns=lieming)
    # 3.将每组数据的 s0,s1,s2,s3 放置到同一行
    mindex = 0
    for i in range(0, hs, 4):
        RR0 = data.iloc[i, 0]
        RR0S = RR0.split(':')
        RR1 = data.iloc[i + 1, 0]
        RR1S = RR1.split(':')
        RR2 = data.iloc[i + 2, 0]
        RR2S = RR2.split(':')
        RR3 = data.iloc[i + 3, 0]
        RR3S = RR3.split(':')
        # 2.获取每组的四个距离
        T = RR0S[1]
        s0 = int(RR0S[5])
        s1 = int(RR1S[5])
        s2 = int(RR2S[5])
        s3 = int(RR3S[5])
        df.loc[mindex, lieming] = [T, s0, s1, s2, s3]
        mindex = mindex + 1
    # 7.输出最后文件
    df.to_excel('./附件 2 数据/' + file_type + '数据处理后'      '.xls',
```

```
index=False)
# 场景 1 的锚点位置
A0 = (0, 0, 1300)
A1 = (5000, 0, 1700)
A2 = (0, 5000, 1700)
A3 = (5000, 5000, 1300)
x0 = A0[0]
y0 = A0[1]
z0 = A0[2]
x1 = A1[0]
y1 = A1[1]
z1 = A1[2]
x2 = A2[0]
y2 = A2[1]
z2 = A2[2]
x3 = A3[0]
y3 = A3[1]
z3 = A3[2]
# 最小二乘法
A = [[x0 - x3, y0 - y3, z0 - z3],
     [x1 - x3, y1 - y3, z1 - z3],
     [x2 - x3, y2 - y3, z2 - z3]]
A = np.multiply(-2, A)
def getRes(d0, d1, d2, d3):
    b = np.array([[d0 ** 2 - d3 ** 2 + x3 ** 2 + y3 ** 2 + z3 ** 2 - x0 ** 2 -
y0 ** 2 - z0 ** 2],
                  [d1 ** 2 - d3 ** 2 + x3 ** 2 + y3 ** 2 + z3 ** 2 - x1 ** 2
- y1 ** 2 - z1 ** 2],
                  [d2 ** 2 - d3 ** 2 + x3 ** 2 + y3 ** 2 + z3 ** 2 - x2 ** 2
- y2 ** 2 - z2 ** 2]])
    x, y, z = np.dot((np.dot((np.linalg.inv(np.dot(A.T, A))), A.T)), b)
    # x, y, z = np.dot(np.linalg.inv(A), b)
    return x[0], y[0], z[0]
def getLoc(path):
    data = pd.read_excel(path)
    # 获取每个文件的数据
    counts = data.shape[0]
    d = []
    loc = []
    for i in range(counts):
        [d0, d1, d2, d3] = data.iloc[i, 1:]
        d.append([d0, d1, d2, d3])
    for dlist in d:
```

```
            x, y, z = getRes(dlist[0], dlist[1], dlist[2], dlist[3])
            loc.append([x, y, z])
        loc = np.array(loc)
        x = loc[:, 0]
        y = loc[:, 1]
        z = loc[:, 2]
        return np.array([x, y, z]).T
if __name__ == '__main__':
        # fixdata('正常','附件 2 数据/附件 2 正常数据.txt')
        # fixdata('异常','附件 2 数据/附件 2 异常数据.txt')
        zcpath = '附件 2 数据/正常数据处理后.xls'
        ycpath = '附件 2 数据/异常数据处理后.xls'
        zcLoc = getLoc(zcpath)
        ycLoc = getLoc(ycpath)
        zcmodel = tf.keras.models.load_model('./mymodel/task2zc.h5')
        ycmodel = tf.keras.models.load_model('./mymodel/task2yc.h5')
        zcpre = zcmodel.predict(zcLoc)
        ycpre = ycmodel.predict(ycLoc)
        print(zcpre)
        print(ycpre)
```

3、定位模型精度求解

```
import pandas as pd

import numpy as np

zcpre = pd.read_csv('temp/zcpre.csv')

ycpre = pd.read_csv('temp/ycpre.csv')

real = pd.read_csv('temp/real.csv').iloc[:,1:]

pre = zcpre.append(ycpre).iloc[:,1:]

xypre = pre.iloc[:,0:2]

xzpre = pre.iloc[:,[0,2]]

yzpre = pre.iloc[:,1:]

xpre = pre.iloc[:,0]

ypre = pre.iloc[:,1]

zpre = pre.iloc[:,2]

# 3 维精度
```

```
loss = (pre-real)**2
xyzloss = np.sqrt(np.mean(np.mean(loss)))
print('xyzloss:',xyzloss)
# 2 维精度
xyloss = (xypre-real.iloc[:,0:2])**2
xzloss = (xzpre-real.iloc[:,[0,2]])**2
yzloss = (yzpre-real.iloc[:,1:])**2
xyzloss = np.sqrt(np.mean(np.mean(xyloss)))
xzloss = np.sqrt(np.mean(np.mean(xzloss)))
yzloss = np.sqrt(np.mean(np.mean(yzloss)))
print('xyloss:',xyzloss)
print('xzloss:',xzloss)
print('yzloss:',yzloss)
# 1 维精度
xloss = (xpre-real.iloc[:,0])**2
yloss = (ypre-real.iloc[:,1])**2
zloss = (zpre-real.iloc[:,2])**2
xloss = np.sqrt(np.mean(np.mean(xloss)))
yloss = np.sqrt(np.mean(np.mean(yloss)))
zloss = np.sqrt(np.mean(np.mean(zloss)))
print('xloss:',xloss)
print('yloss:',yloss)
print('zloss:',zloss)
```

附录四：任务三求解

```
import tensorflow as tf
import numpy as np
import pandas as pd
def fixdata(file_type,path):
    data = pd.read_table(path,header=None)
    hs = data.shape[0]
    lieming = ['时间戳', 'D0', 'D1', 'D2', 'D3']
    df = pd.DataFrame(columns=lieming)
    # 3.将每组数据的 s0,s1,s2,s3 放置到同一行
    mindex = 0
    for i in range(0, hs, 4):
        RR0 = data.iloc[i, 0]
        RR0S = RR0.split(':')
        RR1 = data.iloc[i + 1, 0]
        RR1S = RR1.split(':')
        RR2 = data.iloc[i + 2, 0]
        RR2S = RR2.split(':')
        RR3 = data.iloc[i + 3, 0]
        RR3S = RR3.split(':')
        # 2.获取每组的四个距离
        T = RR0S[1]
        s0 = int(RR0S[5])
        s1 = int(RR1S[5])
        s2 = int(RR2S[5])
        s3 = int(RR3S[5])
        df.loc[mindex, lieming] = [T, s0, s1, s2, s3]
        mindex = mindex + 1
    # 7.输出最后文件
    df.to_excel('./附件 3 数据/' + file_type + '数据处理后'      '.xls',
index=False)
# 场景 1 的锚点位置
A0 = (0, 0, 1200)
A1 = (5000, 0, 1600)
A2 = (0, 3000, 1600)
A3 = (5000, 3000, 1200)
x0 = A0[0]
y0 = A0[1]
z0 = A0[2]
x1 = A1[0]
y1 = A1[1]
z1 = A1[2]
x2 = A2[0]
y2 = A2[1]
z2 = A2[2]
x3 = A3[0]
y3 = A3[1]
z3 = A3[2]
```

```
# 最小二乘法
A = [[x0 - x3, y0 - y3, z0 - z3],
     [x1 - x3, y1 - y3, z1 - z3],
     [x2 - x3, y2 - y3, z2 - z3]]
A = np.multiply(-2, A)

def getRes(d0, d1, d2, d3):
    b = np.array([[d0 ** 2 - d3 ** 2 + x3 ** 2 + y3 ** 2 + z3 ** 2 - x0 ** 2 -
y0 ** 2 - z0 ** 2],
                  [d1 ** 2 - d3 ** 2 + x3 ** 2 + y3 ** 2 + z3 ** 2 - x1 ** 2
- y1 ** 2 - z1 ** 2],
                  [d2 ** 2 - d3 ** 2 + x3 ** 2 + y3 ** 2 + z3 ** 2 - x2 ** 2
- y2 ** 2 - z2 ** 2]])
    x, y, z = np.dot((np.dot((np.linalg.inv(np.dot(A.T, A))), A.T)), b)
    # x, y, z = np.dot(np.linalg.inv(A), b)
    return x[0], y[0], z[0]
def getLoc(path):
    data = pd.read_excel(path)
    # 获取每个文件的数据
    counts = data.shape[0]
    d = []
    loc = []
    for i in range(counts):
        [d0, d1, d2, d3] = data.iloc[i, 1:]
        d.append([d0, d1, d2, d3])

    for dlist in d:
        x, y, z = getRes(dlist[0], dlist[1], dlist[2], dlist[3])
        loc.append([x, y, z])
    loc = np.array(loc)
    x = loc[:, 0]
    y = loc[:, 1]
    z = loc[:, 2]
    return np.array([x, y, z]).T
if __name__ == '__main__':
    fixdata('正常','附件 3 数据/附件 3 正常数据.txt')
    fixdata('异常','附件 3 数据/附件 3 异常数据.txt')
    zcpath = '附件 3 数据/正常数据处理后.xls'
    ycpath = '附件 3 数据/异常数据处理后.xls'
    zcLoc = getLoc(zcpath)
    ycLoc = getLoc(ycpath)
    zcmodel = tf.keras.models.load_model('./mymodel/task2zc.h5')
    ycmodel = tf.keras.models.load_model('./mymodel/task2yc.h5')
    zcpre = zcmodel.predict(zcLoc)
    ycpre = ycmodel.predict(ycLoc)
    print(zcpre)
    print(ycpre)
```

附录五：任务四求解

```
import joblib
import pandas as pd
import numpy as np
import os.path
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.preprocessing import StandardScaler
from sklearn import neighbors
from sklearn.ensemble import RandomForestClassifier,BaggingClassifier
from scipy import stats
# 遍历单个文件的数据
def getSingleData(path):
    data = pd.read_excel(path)
    data = data.iloc[:, 1:]
    return data
def getAlldatas():
    allzc = pd.DataFrame()
    allyc = pd.DataFrame()
    # 遍历读取文件
    path_zc = './newdatas/正常数据/'
    path_yc = './newdatas/异常数据/'
    zcs = os.listdir(path_zc)
    ycs = os.listdir(path_yc)
    for zc in zcs:
        zc_path = path_zc + zc
        data = getSingleData(zc_path)
        allzc = allzc.append(data, ignore_index=True)
    for yc in ycs:
        yc_path = path_yc + yc
        data = getSingleData(yc_path)
        allyc = allyc.append(data, ignore_index=True)

    allzc.to_csv('./newdatas/正常数据.csv', index=False)
    allyc.to_csv('./newdatas/异常数据.csv', index=False)
def makeModel():
    x1 = pd.read_csv('./newdatas/正常数据.csv').iloc[:,1:]
    x2 = pd.read_csv('./newdatas/异常数据.csv').iloc[:,1:]
    x = x1.append(x2,ignore_index=True)
    # 0 正常   1 异常
    n1 = x1.shape[0]
    n2 = x2.shape[0]
    y = np.zeros(n1+n2)
    y[n1:]=1
    x = StandardScaler().fit_transform(x)
```

```
    x_train, x_test, y_train, y_test = train_test_split(x, y)
    knn = neighbors.KNeighborsClassifier()
    knn.fit(x_train,y_train)
    print('knn_score====', knn.score(x_test,y_test))
    joblib.dump(knn, './mymodel/knn_task4.pkl')
    print(confusion_matrix(y_test, knn.predict(x_test)))
    print(classification_report(y_test, knn.predict(x_test)))
    print('==================================================')
    RF1 = RandomForestClassifier(n_estimators=10,min_samples_split=2)
    RF1.fit(x_train, y_train)
    print('RF1_score====', RF1.score(x_test, y_test))
    joblib.dump(RF1, './mymodel/RF1_task4.pkl')
    print(confusion_matrix(y_test, RF1.predict(x_test)))
    print(classification_report(y_test, RF1.predict(x_test)))
    print('==================================================')
    RF2 = RandomForestClassifier(n_estimators=100, min_samples_split=4)
    RF2.fit(x_train, y_train)
    print('RF2_score====', RF2.score(x_test, y_test))
    joblib.dump(RF2, './mymodel/RF2_task4.pkl')
    print(confusion_matrix(y_test, RF2.predict(x_test)))
    print(classification_report(y_test, RF2.predict(x_test)))
    print('==================================================')
    bagg = BaggingClassifier(RF2,n_estimators=20)
    bagg.fit(x_train, y_train)
    print('bagg_score====', bagg.score(x_test, y_test))
    joblib.dump(bagg, './mymodel/bagg_task4.pkl')
    print(confusion_matrix(y_test, bagg.predict(x_test)))
    print(classification_report(y_test, bagg.predict(x_test)))
    print('==================================================')
    mlp = MLPClassifier(hidden_layer_sizes=(20, 10), max_iter=1000)
    mlp.fit(x_train, y_train)
    print('mlp_score====', mlp.score(x_test, y_test))
    joblib.dump(mlp, './mymodel/mlp_task4.pkl')
    print(confusion_matrix(y_test, mlp.predict(x_test)))
    print(classification_report(y_test, mlp.predict(x_test)))
def fixdata4():
    # 1.先处理附件 4 的数据，将它们提取为一个数据文件
    filepath = './附件 4 数据/附件 4 数据.txt'
    deskpath = './附件 4 数据/附件 4 处理后数据.xls'
    data = pd.read_table(filepath,header=None)
    #  创建空 df
    hs = data.shape[0]
    lieming = ['时间戳', 'A0', 'A1', 'A2', 'A3']
    df = pd.DataFrame(columns=lieming)
    # 将每组数据的 s0,s1,s2,s3 放置到同一行
    mindex = 0
    for i in range(0, hs, 4):
        RR0 = data.iloc[i, 0]
```

```
            RR0S = RR0.split(':')
            RR1 = data.iloc[i + 1, 0]
            RR1S = RR1.split(':')
            RR2 = data.iloc[i + 2, 0]
            RR2S = RR2.split(':')
            RR3 = data.iloc[i + 3, 0]
            RR3S = RR3.split(':')
            # 2.获取每组的四个距离
            T = RR0S[1]
            s0 = int(RR0S[5])
            s1 = int(RR1S[5])
            s2 = int(RR2S[5])
            s3 = int(RR3S[5])
            df.loc[mindex, lieming] = [T, s0, s1, s2, s3]
            mindex = mindex + 1
    df.to_excel(deskpath, index=False) #为了方便存了一个文本，注释即可
def getClass():
    df = pd.read_excel('附件 4 数据/附件 4 处理后数据.xls')
    modelpath1 = './mymodel/bagg_task4.pkl'
    modelpath2 = './mymodel/knn_task4.pkl'
    modelpath3 = './mymodel/mlp_task4.pkl'
    modelpath4 = './mymodel/RF1_task4.pkl'
    modelpath5 = './mymodel/RF2_task4.pkl'
    # 2.将数据传入模型预测
    x = df.iloc[:,1:]
    x = StandardScaler().fit_transform(x)
    model1 = joblib.load(modelpath1)
    model2 = joblib.load(modelpath2)
    model3 = joblib.load(modelpath3)
    model4 = joblib.load(modelpath4)
    model5 = joblib.load(modelpath5)
    y1 = model1.predict(x)
    y2 = model2.predict(x)
    y3 = model3.predict(x)
    y4 = model4.predict(x)
    y5 = model5.predict(x)
    res = [y1,y2,y3,y4,y5]
    res = np.array(res).T
    return stats.mode(res,axis=1)[0].flatten()
if __name__ == '__main__':
    # 运行时分开运行，不能一块
    getAlldatas()
    makeModel()
    fixdata4()
    res = getClass()
    print(res)
```

附录六：任务五求解

```
import tensorflow as tf
import numpy as np
import pandas as pd
from scipy import stats
from sklearn.preprocessing import StandardScaler
import joblib
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
# 场景 1 的锚点位置
A0 = (0, 0, 1300)
A1 = (5000, 0, 1700)
A2 = (0, 5000, 1700)
A3 = (5000, 5000, 1300)
x0 = A0[0]
y0 = A0[1]
z0 = A0[2]
x1 = A1[0]
y1 = A1[1]
z1 = A1[2]
x2 = A2[0]
y2 = A2[1]
z2 = A2[2]
x3 = A3[0]
y3 = A3[1]
z3 = A3[2]
# 最小二乘法
A = [[x0 - x3, y0 - y3, z0 - z3],
     [x1 - x3, y1 - y3, z1 - z3],
     [x2 - x3, y2 - y3, z2 - z3]]
A = np.multiply(-2, A)

def getRes(d0, d1, d2, d3):
    b = np.array([[d0 ** 2 - d3 ** 2 + x3 ** 2 + y3 ** 2 + z3 ** 2 - x0 ** 2 -
y0 ** 2 - z0 ** 2],
                  [d1 ** 2 - d3 ** 2 + x3 ** 2 + y3 ** 2 + z3 ** 2 - x1 ** 2
- y1 ** 2 - z1 ** 2],
                  [d2 ** 2 - d3 ** 2 + x3 ** 2 + y3 ** 2 + z3 ** 2 - x2 ** 2
- y2 ** 2 - z2 ** 2]])
    x, y, z = np.dot((np.dot((np.linalg.inv(np.dot(A.T, A))), A.T)), b)
    # x, y, z = np.dot(np.linalg.inv(A), b)
    return x[0], y[0], z[0]
def getLoc(path):
    data = pd.read_excel(path)
    # 获取每个文件的数据
    counts = data.shape[0]
    d = []
    loc = []
```

```
        for i in range(counts):
            [d0, d1, d2, d3] = data.iloc[i, 1:]
            d.append([d0, d1, d2, d3])
        for dlist in d:
            x, y, z = getRes(dlist[0], dlist[1], dlist[2], dlist[3])
            loc.append([x, y, z])
        loc = np.array(loc)
        x = loc[:, 0]
        y = loc[:, 1]
        z = loc[:, 2]
        return np.array([x, y, z]).T
def getClass(path):
        df = pd.read_excel(path)
        modelpath1 = './mymodel/bagg_task4.pkl'
        modelpath2 = './mymodel/knn_task4.pkl'
        modelpath3 = './mymodel/mlp_task4.pkl'
        modelpath4 = './mymodel/RF1_task4.pkl'
        modelpath5 = './mymodel/RF2_task4.pkl'

        # 2.将数据传入模型预测
        x = df.iloc[:,1:]
        x = StandardScaler().fit_transform(x)
        model1 = joblib.load(modelpath1)
        model2 = joblib.load(modelpath2)
        model3 = joblib.load(modelpath3)
        model4 = joblib.load(modelpath4)
        model5 = joblib.load(modelpath5)
        y1 = model1.predict(x)
        y2 = model2.predict(x)
        y3 = model3.predict(x)
        y4 = model4.predict(x)
        y5 = model5.predict(x)
        res = [y1,y2,y3,y4,y5]
        res = np.array(res).T
        return stats.mode(res,axis=1)[0].flatten()
def xyz3d(xyz):
        fig = plt.figure()
        ax = Axes3D(fig)
        x1 = xyz[:113,0]
        y1 = xyz[:113,1]
        z1 = xyz[:113,2]
        x2 = xyz[113:400, 0]
        y2 = xyz[113:400, 1]
        z2 = xyz[113:400, 2]
        x3 = xyz[400:, 0]
        y3 = xyz[400:, 1]
        z3 = xyz[400:, 2]
        ax.scatter(x1,y1,z1)
        ax.scatter(x2,y2,z2)
```

```
    ax.scatter(x3,y3,z3)
    ax.set_xlabel('X/mm')
    ax.set_ylabel('Y/mm')
    ax.set_zlabel('Z/mm')
    ax.set_xlim(0,5000)
    ax.set_ylim(0,5000)
    ax.set_zlim(0,3000)
    plt.show()
if __name__ == '__main__':
    '''
    zcmodel = tf.keras.models.load_model('./mymodel/task2zc.h5')
    ycmodel = tf.keras.models.load_model('./mymodel/task2yc.h5')

    filepath = 'datas/附件 5 新数据.xls'
    cls = getClass(filepath)
    loc = getLoc(filepath)
    xyz_res = []
    num = 0
    indexList=[]
    for index,(cls,loc) in enumerate(zip(cls,loc)):
        loc = np.array([loc])
        if cls==0:# 无干扰
            yp = zcmodel.predict(loc)
            yp = np.array(yp).flatten()
            if not (np.min(yp,axis=0)<0 or np.max(yp,axis=0)>500 or
yp[2]>300):
                xyz_res.append(yp)
            else:
                num+=1
                indexList.append(index)
        else:# 有干扰
            yp = ycmodel.predict(loc)
            yp = np.array(yp).flatten()
            if not (np.min(yp, axis=0) < 0 or np.max(yp, axis=0) > 500 or
yp[2]>300):
                xyz_res.append(yp)
            else:
                num+=1
                indexList.append(index)
    print(num)
    print(np.array(indexList))
    print(len(xyz_res))
    xyz_res = np.array(np.dot(10,xyz_res))
    xyz3d(xyz_res)
```