|  |  |  |
| --- | --- | --- |
| 所在组别 | **2022年第二届中国高校大数据挑战赛** | 参赛编号 |
| 本科组 | bdc220586 |

基于树模型的设备故障预测与分析

摘 要

本文根据题目给出的机械设备多个指标的相关数据、故障相关的类型信息，基于树模型如决策树、随机森林模型、GBDT模型、XGBoost模型、LightGBM模型等建立了一系列判别机械设备是否发生故障及故障类型的模型，以期帮助相关人员更好开展相关工作。

针对任务一：首先对机器质量等级进行编码；删除是否发生故障指标为1同时具体故障类型为Normal的两条异常数据；经分析得出机器编号、统一规范代码两个指标对后期的模型训练没有帮助，予以删除；然后对剩余的六个指标的相关性利用**perason相关系数**进行刻画，分析出所处厂房的温度与工作时的机器温度、转速和扭矩两对变量之间存在着强相关性，在后期如果模型对变量的相关性敏感则考虑予以删除，否则予以保留；针对数据集中正常运转设备的数据量与发生故障设备的数据量严重失衡的问题，使用**SMOTE算法**对数据集进行过采样，最终两者的比例降低到4：1，有效缓解了数据量失衡的问题。

针对任务二：本文采取**集成学习**的思想，首先针对任务一处理后的数据集训练出四个基分类器，分别为**随机森林模型、GBDT模型、XGBoost模型、LightGBM模型**，使用**在测试集上的准确率和混淆矩阵**两个指标对四个基分类器的模型表现进行评估。在进行多次**调参**之后，四个基分类器在测试集上的准确率分别为98.40%、98.44%、98.44%、98.06%，混淆矩阵见图11~图14。利用**投票法**对四个基分类器进行集成，最后得到的二分类器在测试集上的准确率达到98.61%，混淆矩阵见图15。该分类器接近于完美分类器，具有不易过拟合、预测结果准确等优点。

针对任务三：鉴于任务二训练出的二分类器效果显著，且四个基分类器在**多分类**问题上均能胜任，所以继续训练集成学习模型，采用与任务二相同的评估指标。其中，四个基分类器在测试集上的准确率均达到了100%，在依据投票法进行集成之后，集成学习的模型在测试集上的准确率达到了100%，混淆矩阵见图16，可以认为是**完美分类器**。

针对任务四：首先使用任务二训练出来的二分类器对给定的数据集进行预测，结果表明有967条数据为正常运转类型，33条数据为发生故障的类型。然后利用任务三训练出来的多分类器对发生故障的数据预测具体的故障类别，预测结果中TWF、HDF、PWF、OSF、RNF类故障数据分别有7、10、7、9、0条，详见**forecast.xlsx**。

针对任务五：为了初步探索数据集里五类故障的数据的规律，首先对数据集进行侧写，绘制发生五类故障的设备与正常运转的设备在六个指标上的**直方图**，得出初步结论：发生TWF故障的机械设备，其平均使用时间显著高于非故障的设备；HDF类故障设备的室温、扭矩与转速均明显区别于非故障类设备；PWF类故障设备的扭矩明显高于或明显低于非故障类的平均水平；OSF类过载故障普遍机器质量等级较低；由于RNF类故障样本太少，且各项指标均未明显偏离非故障类的均值，所以没有观测到明显特征。接着进一步构建**决策树模型**，依据模型结果分别给出五类故障在某些指标上可能处于的范围，具体结果可见**表8~表11**。

关键词：SMOTE算法 二分类与多分类 集成学习 决策树

# 问题重述

## 问题背景

近年来，我国制造业在庞大的市场规模、充足的劳动力的加持下，不断攀上新高度。而机械设备作为工业生产过程中的核心，在长时间的高负荷运转下，产生故障也是在所难免，如磨损(图1所示)、老化(图2所示)等。

|  |  |
| --- | --- |
|  |  |
| 图1 机械磨损 | 图2 机械老化 |

在这种情况下，根据机械设备的使用情况探究不同故障的主要成因，提前预测潜在的故障风险，可以保证设备的安全可靠运行，同时获取更大的经济效益，具有深远的意义。

## 问题重述

本题提供了企业设备的使用情况和故障发生情况的相关数据，要求据此完成以下五个任务：

* **任务一：**对数据“train data.xlsx”进行数据预处理，挑选合适的指标进行机械设备故障的预测，并给出合理的解释。
* **任务二：**设计模型判断机械设备是否发生故障，并构造评价方式、评价指标对模型的表现进行评估。
* **任务三：**设计模型判断机械设备发生故障的具体类别，并构造评价方式、评价指标对模型的表现进行评估。
* **任务四：**结合任务二、三构造的模型对给定的数据集“forecast.xlsx”进行预测，判断是否发生故障，如发生故障进一步判断具体故障类别。
* **任务五：**探究不同类别的故障的主要成因，寻找出相关的特征属性，进行量化分析，挖掘其背后的模式或规则。

# 问题分析

## 任务一的分析

针对任务一，在数据预处理步骤，本文首先对机器质量等级进行编码，为与是否发生故障这一指标相呼应，将L(低)定义为2，M(中)定义为1，H(高)定义为0；并且删除样本数据集里面同时发生故障同时具体故障类别为“normal”的两条异常数据。

在特征筛选步骤，首先分析出机器编号、统一规范代码对后期的模型训练没有帮助，予以剔除，同时对剩余的六个指标利用perason相关系数对其相关性进行刻画，分析出两对相关性较高的变量，分别为室温1(所处厂房的温度)、室温2(工作时的机器温度)和转速、扭矩，在后期模型训练时可以适当予以注意。

同时针对原始数据集发生故障和正常使用的数据量严重失调的问题，使用SMOTE算法进行过采样，有效地缓解了类不平衡的问题。

## 任务二的分析

对于任务二，本文采用集成学习的方法进行预测。将数据集按比例随机切割出训练集和测试集之后，首先训练出四个基分类器：随机森林、GBDT模型、XGBoost模型、LightGBM模型，分别观察它们在测试集上的表现效果。

然后依据少数服从多数的原则，对四个基分类器利用投票的方法进行集成，进一步提高模型的泛化能力和预测的精度。

## 任务三的分析

首先将数据集中发生故障的数据切割出来，并且按比例随机切割出训练集和测试集。

鉴于四个基分类器在任务二上的表现都十分的优越，并且四个基分类器均适用于多分类问题，所以在任务三的预测中继续训练这四个基分类器，观察它们在测试集上的表现。

然后依据少数服从多数的原则，对四个基分类器利用投票的方法进行集成，使得多分类器的预测结果精度更高。

## 任务四的分析

任务四针对设备是否发生故障，首先利用任务二训练出来的二分类器进行预测。筛选出二分类预测结果中发生故障的数据，利用任务三训练出来的多分类器进行预测，得出具体的故障类型。

## 任务五的分析

为了探究不同类别故障的主要成因，首先进行数据集的侧写，绘制出五类故障的设备与正常无故障的设备的数据集在机器质量等级、所处厂房的室温、工作时的机器温度、转速、扭矩、机器运转时长六个指标的直方图，观察它们数据的相关分布规律，得出初步的规律。

然后构建决策树模型，利用程序求解出的决策树模型结合数据集侧写过程时总结的规律，分析出五类故障的设备一些关键的指标所处的范围，进行量化分析，挖掘出存在的规则。

图3给出了整道题的思维流程：



图3：全题思维流程图

# 任务一的求解

## 数据预处理

* **数据编码**

在机器学习中，特征数据一般分为数值特征和对象特征，对象特征一般为离散变量，对此本文采用数据编码的方式转换为可进行处理的数值。而在问题一中，机器质量等级就是一个对象特征，故对其进行编码。鉴于在是否发生故障这一指标中，1代表发生故障，0代表不发生故障，为了与其对应，所以将L(低)定义为2，M(中)定义为1，H(高)定义为0。

* **筛除异常值**

在对数据进行初步观察时，发现统一规范代码为L55686、L56195的两个样本数据发生故障，同时具体故障类别为normal,互相矛盾，影响后期的模型训练，故将其删除。

## 特征筛选

特征选择是从特征集合中筛选出一个子集，是为了构建模型而选择相关特征子集的过程。

**A.特征剔除**

观察数据集，可以推断机器编号这一特征与机器是否发生故障没有关系，对模型训练没有帮助，予以剔除。

同时统一规范代码这一特征只有第一个字母反映了机器质量等级，剩余的编号与机器是否发生故障无相关性。但机器质量等级已经有指标反映，所以可以认为该指标对模型训练没有帮助，予以剔除。

**B.特征相关性刻画**

本文对机器质量等级、室温1(所处厂房的温度)、室温2(工作时的机器温度)、转速(rpm)、扭矩（Nm）、使用时长（min）六个指标，利用皮尔逊相关系数对它们之间的相关性进行刻画，得到的热力图如图4所示：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 机器质量等级 | 室温1 | 室温2 | 转速（rpm） | 扭矩（Nm） | 使用时长（min） |
| 机器质量等级 | 1 | 0.015 | 0.013 | 0.003 | 0.006 | 0.003 |
| 室温1 | 0.015 | 1 | 0.86 | 0.024 | -0.018 | 0.02 |
| 室温2 | 0.013 | 0.86 | 1 | 0.019 | -0.016 | 0.016 |
| 转速（rpm） | 0.003 | 0.024 | 0.019 | 1 | -0.875 | 0.004 |
| 扭矩（Nm） | 0.006 | -0.018 | -0.016 | -0.875 | 1 | -0.007 |
| 使用时长（min） | 0.003 | 0.02 | 0.016 | 0.004 | -0.007 | 1 |

图4：指标间的相关系数热力图

从图4可以看出，室温1与室温2的相关系数达到0.860，而转速和扭矩的相关系数达到-0.875,可以认为两对变量之间存在着一定的相关性。在后期进行机器学习模型的训练过程中，如果遇到选用的机器学习模型对指标的相关性比较敏感，可以考虑予以筛除；但是如果选用的机器学习模型对指标的相关性不敏感，则不做考虑，予以保留。事实上，本文选用的机器学习模型对指标之间的相关性均不做要求，所以基本对这几个指标予以保留。

## 数据过采样

观察数据可以发现，在train data.xlsx中，不发生故障的数据有8697条，而发生故障的数据仅仅有301条，比例严重失衡。从模型训练的角度来看，如果某类的样本数量很少，这个类别可以提供的信息就很少，模型在追求错误率最小化的过程中可能会直接忽视该类别，从而导致没有学习到如何去判别该类。

而过采样可以对训练集里面样本数量较少的类别（少数类）进行过采样，合成新的样本来缓解类不平衡。本文采用SMOTE过采样算法，主要分为以下三个步骤：

A.遍历少数类中每一个样本x，计算其到与少数类样本集中其余样本的距离，得到其k近邻。

B.据样本不平衡比例设置采样倍率N，对每个少数类的样本x，从其k近邻中随机选取N个样本，设选中的为。

C.对每一个，结合原样本x构造新样本如下：

|  |  |
| --- | --- |
|  | () |

算法流程如图5所示(以k=3,采样倍率N=3为例)：



图5：SMOTE算法示意图

train data.xlsx原始的数据集和过采样后的数据集如表1、表2所示，Normal类型和其他数据集基本满足4:1的比例，一定程度上缓解了比例失调的问题。

表1：采样前数据集 表2：采样后数据集

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类别** | **数量** |  | **类别** | **数量** |
| Normal | 8697 |  | Normal | 8697 |
| HDF | 95 |  | HDF | 686 |
| OSF | 85 |  | OSF | 614 |
| PWF | 74 |  | PWF | 535 |
| RNF | 6 |  | RNF | 43 |
| TWF | 41 |  | TWF | 296 |

## 总结

对于任务一：

Ⅰ剔除了两条异常值

Ⅱ舍弃机器编号、统一规范代码两个特征值，并且在后期训练模型时如果模型对指标间的相关性敏感则在室温1与室温2、转速和扭矩两对变量之间各舍弃一个指标，如果不敏感则予以保留。

Ⅲ针对数据不平衡的问题，利用SMOTE算法进行过采样，缓解了比例失调的问题。

# 任务二的求解

## 模型的建立

### 基分类器的构建

* **随机森林模型**

随机森林是专门为决策树分类器设计的集成算法。它在以决策树为基学习器构建bagging集成的基础上，在训练过程中不再单纯参照传统决策树使用信息增益等指标作为根节点，而是引进了随机属性选择：即对基决策树的每个结点，先从该结点的属性集合中随机选择包含k个属性的子集，再从该子集中选择最优属性用于划分。在追求“多样性”方面，随机森林同时对样本、属性均施与一定的扰动，使得个体学习器之间的差异度进一步增加，有效增强了模型的泛化性能。[1]

随机森林的模型流程图如图6所示。它在很多分类任务中均展示出优越的性能，因此本文选用随机森林模型作为集成学习的一个基学习器进行训练。



图6：随机森林模型图

* **GBDT模型**

GBDT算法是boosting算法的一种，依据梯度提升的思想，它在算法的每一步用一棵决策树去拟合当前学习器的残差，获得一个新的弱学习器。将每一步的决策树进行组合，得到一个强学习器。[2]

类似于逻辑回归、FM模型用于分类问题，本质上是在用一个线性模型或包含交叉项的非线性模型，去拟合对数几率。而GBDT模型用于分类问题，是用一系列的梯度提升树去拟合这个对数几率，最终得到一系列CART回归树。用代表学习到的决策树，分类模型可以表达为：

|  |  |
| --- | --- |
|  | () |

单个样本的损失函数可以表达为：

|  |  |
| --- | --- |
|  | () |

设第k步迭代之后当前学习器为：

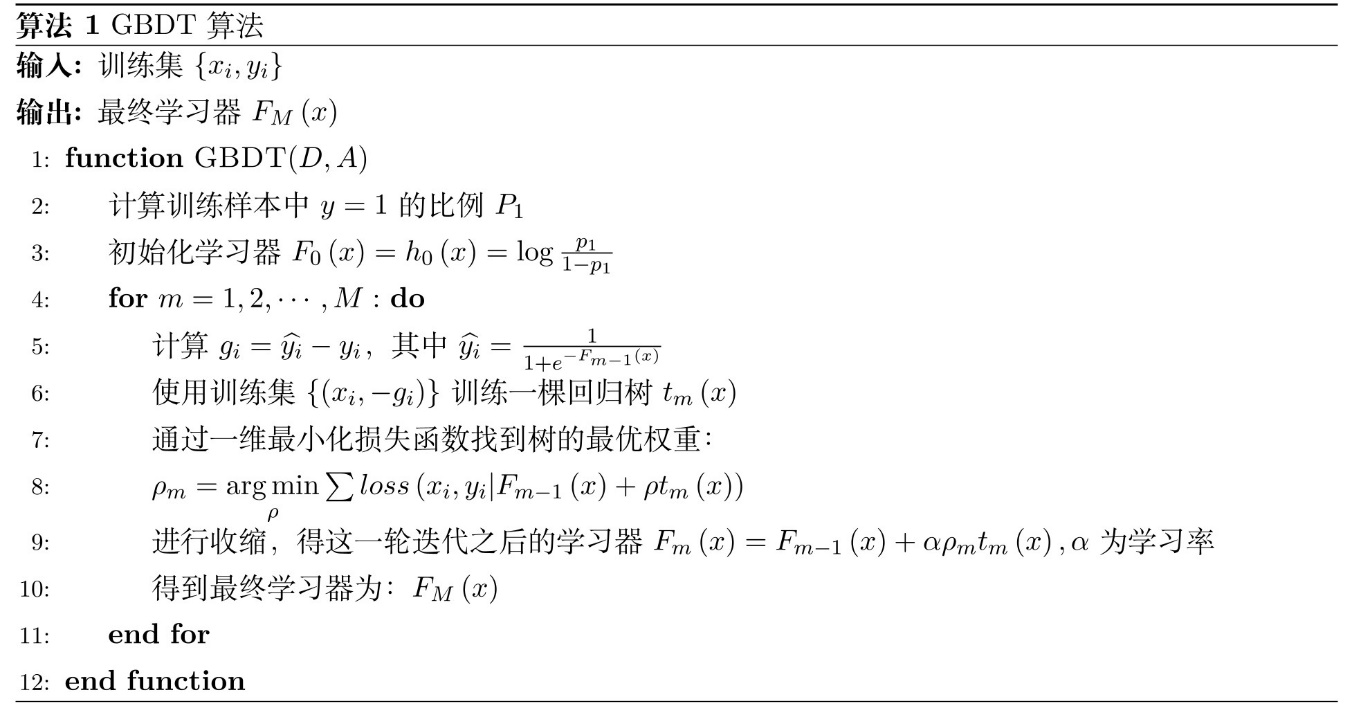
|  |  |
| --- | --- |
|  | () |

代入(3)，得到损失函数为：

|  |  |
| --- | --- |
|  | () |

然后可以求得损失函数相对于当前学习器的负梯度，继续用于训练下一棵决策树。

算法的流程伪代码如下所示(二分类)：



* **XGBoost模型**

XGBoost，又称为极致梯度提升，是在GBDT算法的基础上的进一步优化。在 GBDT的基础上，XGBoost通过最小化损失函数，结合控制模型复杂程度的正则化项，优化目标函数。其步骤如图7所示：



图7：XGBoost流程图

* **LightGBM模型**：

LightGBM模型可以看作是XGBoost的改进版，在获得与XGBoost近似精度的同时，又提供了更快的训练速度与更少的内存消耗。为了解决XGBoost模型空间消耗大、时间开销大、对cache优化不友好的缺点，LightGBM做了以下改进：

1. 基于Histogram的决策树算法：

基本思想是：先把连续的浮点特征值离散化处理成K个整数，构造一个宽度为K的直方图。遍历数据时，直接以离散化后的值作为index在直方图中累计统计量，然后直接根据直方图的离散值，寻找最优分割点,如图8所示:



图8：基于Histogram的决策树算法示意图

1. 直方图做差加速：

为更快提高速度，一个叶子的直方图可以直接由其父亲节点的直方图和兄弟的直方图做差得到。在实际建树过程中，LightGBM先计算直方图小的叶子结点，利用做差获得直方图大的叶子结点，即可以很小的代价得到兄弟结点的直方图。

1. 带深度限制的 Leaf-wise 算法：

LightGBM直接舍弃了GBDT的层生长决策树生长策略，而是采用Leaf-wise的增长策略[3]：每次从当前所有叶子中，找到分裂增益最大的一个叶子，进行分裂，循环下去，如图9所示：



图9：Leaf-wise示意图(白点代表本轮循环选中的叶子)

1. 互斥特征捆绑算法

高维度的数据往往是稀疏的，这时减少特征维度就显得十分重要。互斥特征捆绑算法将一些特征进行融合绑定，设bundle为特征绑定融合后的个数，则构建直方图的时间复杂度从降低为。

### 集成模型

投票是广泛使用的继承算法之一，主要分为软投票和硬投票两种。硬分类器各基分类器权值相同，进行投票，如果基分类器的某一分类结果超过半数，则集成算法选择该结果，若无半数结果则无输出；软投票原理也是少数服从多数，但基分类器可以赋予不同的权值。当基分类器之间差距比较大，一般倾向于使用软投票，赋予分类性能更优的基分类器更大的权值。

基于上面训练出的四个基分类器，按投票算法构造的集成学习模型如图10所示：



图10：集成学习模型图

## 模型的求解

### 基分类器的表现

* 随机森林：

对于随机森林，本文在决策树划分标准(criterion)、决策树个数(n\_estimators)、树的最大深度(max\_depth)、可以被分割的所含最小样本数(min\_samples\_split)、叶子节点所含最少样本数(min\_samples\_leaf)、每棵树用到的最大特征数(max\_features)六个特征之间进行调参，最后获得的最优参数如表3所示，测试集上得到的混淆矩阵如图11所示：

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **参数名称** | **最优参数** | | criterion | ‘entropy’ | | n\_estimators | 120 | | max\_depth | 5 | | min\_samples\_split | 2 | | min\_samples\_leaf | 1 | | max\_features | 0.5 | | |  |  |  | | --- | --- | --- | |  | 无故障 | 有故障 | | 无故障 | 1727 | 25 | | 有故障 | 2 | 46 | |
| 表3 随机森林最优参数表 | 图11 随机森林模型混淆矩阵图 |

在测试集上的准确率达到了98.40%。

* GBDT模型

对于GBDT模型，本文在树的最大深度、可以被分割的所含最小样本数(max\_depth)、可以被分割的所含最小样本数(min\_samples\_split)、叶子节点所含最少样本数(min\_samples\_leaf)、每棵树用到的最大特征数(max\_features)之间进行调参，最后获得的最优参数如表4所示，在测试集上得到的混淆矩阵如图12所示：

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **参数名称** | **最优参数** | | max\_depth | 20 | | min\_samples\_split | 15 | | min\_samples\_leaf | 6 | | max\_features | 0.4 | | |  |  |  | | --- | --- | --- | |  | 无故障 | 有故障 | | 无故障 | 1726 | 26 | | 有故障 | 2 | 46 | |
| 表4 GBDT最优参数表 | 图12 GBDT模型混淆矩阵图 |

在测试集上的准确率达到了98.44%。

* XGBoost模型

对于XGBoost模型，本文在树的最大深度(max\_depth)、决定最小叶子节点样本权重和(min\_child\_weight)、惩罚项系数(gamma)上进行调参，最后获得的最优参数如表5所示，在测试集上获得的混淆矩阵如图13所示：

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **参数名称** | **最优参数** | | max\_depth | 20 | | min\_child\_weight | 2 | | gamma | 0 | | |  |  |  | | --- | --- | --- | |  | 无故障 | 有故障 | | 无故障 | 1726 | 26 | | 有故障 | 2 | 46 | |
| 表5 XGBoost最优参数表 | 图13 XGBoost模型混淆矩阵图 |

在测试集上的准确率达到了98.44%。

* LightGBM模型

对于LightGBM模型，本文在拟合的树的棵树(n\_estimators)、树的最大深度(max\_depth)、决定最小叶子节点样本权重和(min\_child\_weight)三个参数上进行调参，最后获得的最优参数如表6所示，在测试集上获得的混淆矩阵如图14所示：

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **参数名称** | **最优参数** | | n\_estimators | 80 | | max\_depth | 20 | | min\_child\_weight | 4 | | |  |  |  | | --- | --- | --- | |  | 无故障 | 有故障 | | 无故障 | 1719 | 33 | | 有故障 | 2 | 46 | |
| 表5 LightGBM最优参数表 | 图14 LightGBM模型混淆矩阵图 |

在测试集上的准确率达到了98.06%。

### 集成模型的表现

在训练完成四个基分类器之后，进行集成，投票器在测试集上的混淆矩阵如图15所示：

|  |  |  |
| --- | --- | --- |
|  | 无故障 | 有故障 |
| 无故障 | 1729 | 23 |
| 有故障 | 2 | 46 |

图15 集成模型混淆矩阵图

在测试集上的准确率达到了98.61%，高于所有的基分类器，接近完美分类器，并且不存在过拟合的问题。

# 任务三的求解

## 模型的建立

鉴于任务二的分类器效果十分优越，并且使用到的四个基分类器均适用于多分类问题，所以任务三继续采用任务二的分类器，但是标签变为具体故障类别，分别为TWF、HDF、PWF、OSF、RNF五类。

## 模型的表现

首先将出现故障的数据单独切割出来，并利用sklearn库的train\_test\_split函数随机切割数据集，使得训练集占比80%，测试集占比20%。

在随机森林、GBDT模型、XGBoost模型、LightGBM模型上，输入训练集进行训练，在测试集验证模型的泛化能力，调参细节跟任务二的过程类似，不再赘述。最终结果表现，在四个模型上，测试集的准确率均达到了1，效果十分的优越。

在进行投票集成之后，在测试集上的准确率也达到了1。混淆矩阵如图16所示：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | HDF | OSF | PWF | TWF | RNF |
| HDF | 66 | 0 | 0 | 0 | 0 |
| OSF | 0 | 71 |  | 0 | 0 |
| PWF | 0 | 0 | 43 | 0 | 0 |
| TWF | 0 | 0 | 0 | 34 | 0 |
| RNF | 0 | 0 | 0 | 0 | 2 |

图16：多分类器的混淆矩阵

# 任务四的求解

## 有无故障的预测

首先利用任务二训练出来的二分类器对机器是否有故障进行预测，最终预测出来的结果中显示有967个机器没有故障，即为Normal，剩余的33个机器均存在着故障。

## 具体故障类型的预测

接下来对存在着故障的33条数据利用任务三训练出来的多分类模型进行预测，最终预测出来的结果中的详细情况如表7所示：

表7：多分类预测结果表

|  |  |
| --- | --- |
| **故障类型** | **数量** |
| TWF磨损类故障 | 7 |
| HDF散热故障 | 10 |
| PWF电力故障 | 7 |
| OSF过载故障 | 9 |
| RNF其他故障 | 0 |

# 任务五的求解

## 数据集侧写

### TWF类磨损故障

对TWF类的各个指标的数据与Normal类的六个指标放置在一起绘制直方图，显示如下：

|  |  |
| --- | --- |
|  |  |
| 图17 TWF+NORMAL：机器质量等级 | 图18 TWF+NORMAL：室温1 |
|  |  |
| 图19 TWF+NORMAL：室温2 | 图20 TWF+NORMAL：转速 |
|  |  |
| 图21 TWF+NORMAL：扭矩 | 图22 TWF+NORMAL：具体使用时长 |

通过直方图，可以看出，发生TWF故障的机械设备，其平均使用时间显著高于非故障的设备，其余的指标与非故障类的分布大致相同。因此，可以认为，随着使用时间的增加，机械设备更容易发生TWF类故障。

### HDF散热故障

从故障名称即可推测，HDF类故障温度有关。通过直方图进一步观测，可以看出，HDF类故障设备的室温、扭矩与转速均明显区别于非故障类设备。

|  |  |
| --- | --- |
|  |  |
| 图23 HDF+NORMAL：室温 | 图24 HDF+NORMAL：扭矩 |

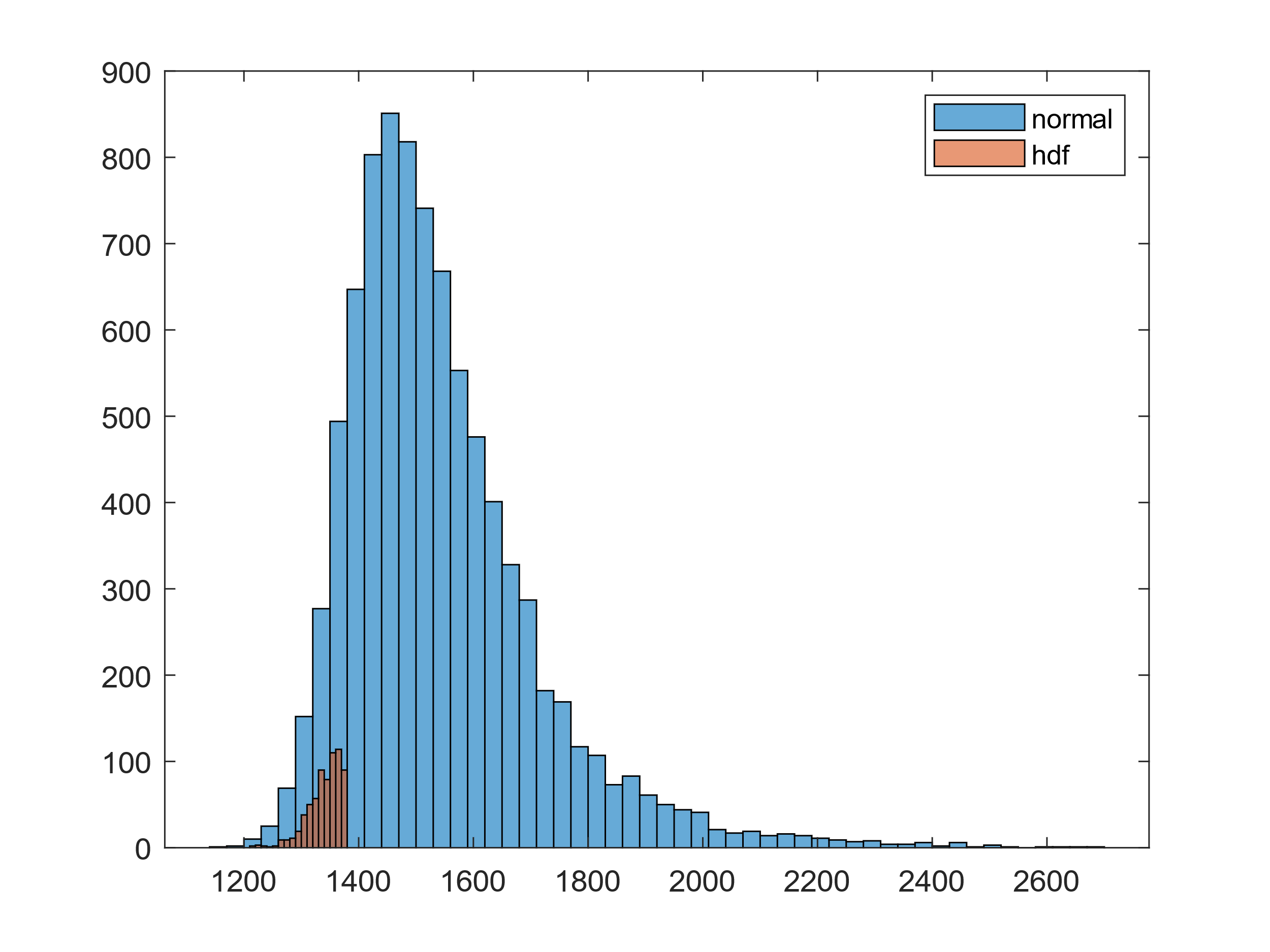


图25 HDF+NORMAL：转速

### PWF电力故障

出现PWF类故障的机械设备，其扭矩与转速的分布呈现两级分化的趋势。在直方图中可以清楚看到，PWF类故障设备的扭矩明显高于或低于非故障类的平均水平。

|  |  |
| --- | --- |
|  |  |
| 图26 PWF+NORMAL：扭矩 | 图27 HDF+NORMAL：转速 |

### OSF过载故障

通过直方图，可以看出，发生OSF故障的机械设备其扭矩、转速与使用时长的分布，均不同于非故障类。值得注意的是，其他类的故障对于设备质量等级的分布是均匀的，而发生OSF类故障的设备，L级占比很高。由此可以推测，低质量等级的机械设备，往往更容易发生OSF类过载故障。

|  |  |
| --- | --- |
|  |  |
| 图28 OSF+NORMAL：扭矩 | 图29 OSF+NORMAL：转速 |

|  |  |
| --- | --- |
|  |  |
| 图30 OSF+NORMAL：使用时长 | 图31 OSF+NORMAL：机器质量等级 |

### RNF其他故障

RNF类故障样本太少，且各项指标均未明显偏离非故障类的均值，直方图也没有观测到明显特征。因此可以推测，出现RNF类故障的原因可能与质量等级、温度、扭矩、转速以及使用时长均无关，可能是某些数据集之外的特征造成了RNF类故障。

## 模型的构建

### 决策树模型

决策树模型是一类常见的机器学习方法，主要思想是基于树模型进行决策，在确定最终的分类类别时，往往会进行一系列的子决策：属性1在何种类别？在属性1的前提下，属性2或属性3在何种类别？最后得出最终决策，思想如图32示例所示：

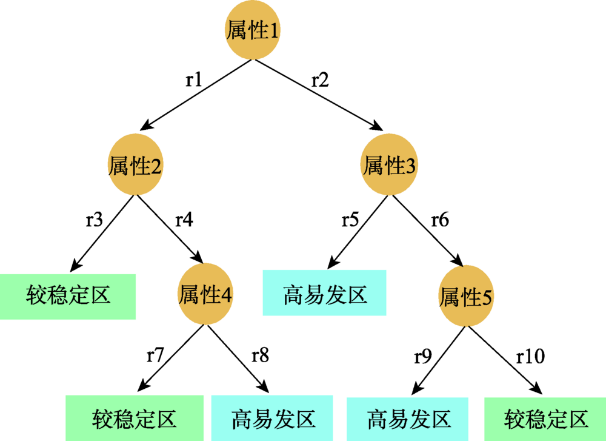


图32：决策树示意图

决策树学习的关键是如何选择最优划分属性。在划分过程中，希望决策树的分支节点所包含的样本尽可能属于同一类别，即所谓的不断提高“纯度”。这里本文使用到的模型是基于基尼指数的属性划分方法，流程如下：

A.计算比例：

计算当前样本集合D中第k类样本所占的比例

B.用基尼值衡量数据集D的纯度：

|  |  |
| --- | --- |
|  | () |

C.计算数据集D中，候选属性集合A中a属性的基尼指数：

|  |  |
| --- | --- |
|  | () |

D.筛选划分属性：

在A中选择使得划分后基尼系数最小的属性作为最优划分属性，即：

|  |  |
| --- | --- |
|  | () |

E.循环下去，直至每个分支节点所包含的样本均属于同一类别或者某个类别达到一个相对高的比例。

## 模型的结论

由于绘制出来的决策树可视化结果过于庞大冗余，正文部分不放入，只做决策树部分的结果阐释：

* TWF磨损故障

综合数据集侧写部分对TWF磨损故障的分析，结合决策树的绘制结果，可以大致给出判断机械设备是否会发生TWF故障的指标范围：

表8：TWF故障指标范围

|  |  |
| --- | --- |
| **指标** | **范围** |
| 室温（K） | [299.2, 302.0] |
| 转速（rpm） | [1614, 2120] |
| 扭矩（Nm） | [13.2, 52.6] |
| 使用时长（min） | [224.5,237.5] |

如果机械设备同时满足以上条件，则很有可能出现TWF类故障。其中使用时长为判断是否会出现TWF类故障的最关键指标。

* HDF散热故障

综合数据集侧写部分对HDF散热故障的分析，结合决策树的绘制结果，可以大致给出判断机械设备是否会发生HDF故障的指标范围：

表9：HDF故障指标范围

|  |  |
| --- | --- |
| **指标** | **范围** |
| 室温（K） | ≥301.5 |
| 转速（rpm） | ≤1380.5 |

* PWF电力故障

综合数据集侧写部分对PWF电力故障的分析，结合决策树的绘制结果，可以大致给出判断机械设备是否会发生PWF故障的指标范围：

表10：PWF故障指标范围

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **指标** | **范围** |  | **指标** | **范围** |
| 扭矩（Nm） | ≥65 | 或 | 扭矩（Nm） | ≤12.5 |
| 转速（rpm） | ≥1231.6 |  |  |  |
| 使用时长（min） | ≤166.5 |  |  |  |

* OSF过载故障

综合数据集侧写部分对OSF过载故障的分析，结合决策树的绘制结果，可以大致给出判断机械设备是否会发生OSF过载故障的指标范围：

表11：OSF过载故障指标范围

|  |  |
| --- | --- |
| **指标** | **范围** |
| 机器质量等级 | L、M |
| 扭矩（Nm） | [50.5, 65.0] |
| 转速（rpm） | 1380.5 |
| 使用时长（min） | 206.5 |

* RNF其他故障

由于RNF类故障样本太少，且各项指标均未明显偏离非故障类的均值，分布十分的随机，决策树没有观测到比较明显的特征。

# 参考文献

[1]周志华，机器学习，清华大学出版社，2016，179页~180页

[2]Eureka，梯度提升树（GBDT），https://zhuanlan.zhihu.com/p/40096769，2022.10.30

[3]浩波的笔记，LightGBM（lgb）介绍，https://zhuanlan.zhihu.com/p/165627712，2022.10.30

附录

|  |  |  |
| --- | --- | --- |
| 代码名称 | 代码解决的问题 | 代码电子版所在的位置 |
| Smote.py | 对原始数据集过采样，构造训练集 | /A220586附件.zip |
| test\_sample.py | 随机选取原始数据作为测试集 | /A220586附件.zip |
| rf\_parameter\_adjust.py | RF模型参数调优 | /A220586附件.zip |
| gbdt\_parameter\_adjust.py | GBDT模型参数调优 | /A220586附件.zip |
| xgboost\_parameter\_adjust.py | XGBoost模型参数调优 | /A220586附件.zip |
| lgbm\_parameter\_adjust.py | LightGBM模型参数调优 | /A220586附件.zip |
| Ensemble\_task2.py | 构建任务二的集成模型并测试精度 | /A220586附件.zip |
| Ensemble\_task3.py | 构建任务三的集成模型并测试精度 | /A220586附件.zip |
| forecast.py | 预测任务四的数据 | /A220586附件.zip |
| DTree.py | 构建任务五的决策树模型并可视化 | /A220586附件.zip |

|  |
| --- |
| 附录1 |
| Smote.py：对原始数据集过采样，构造训练集 |
| import numpy as np  import pandas as pd  from imblearn.over\_sampling import SMOTE  data = pd.read\_excel("train data.xlsx", engine='openpyxl')  data = data.drop(index=data[(data["具体故障类别"] == "Normal") & (data["是否发生故障"] == 1)].index.tolist())  data = data.drop(["机器编号"], axis=1)  data = data.drop(["统一规范代码"], axis=1)  def class\_dict(m):  class\_dictionary = {'H': 0, 'M': 1, 'L': 2}  return class\_dictionary[m]  data["机器质量等级"] = data["机器质量等级"].map(lambda x : list(map(class\_dict, x))[0])  def resample(data, sort, number, k\_neighbors):  data\_pos = data.loc[data["具体故障类别"] == "Normal"].sample(number)  data\_neg = data.loc[data["具体故障类别"] == sort]  data\_concat = pd.concat([data\_pos, data\_neg], axis=0, ignore\_index=False)  X = data\_concat[["机器质量等级", "室温（K）", "室温（K）.1", "转速（rpm）", "扭矩（Nm）", "使用时长（min）"]].values  y = data\_concat["具体故障类别"].values  sm = SMOTE(k\_neighbors=k\_neighbors)  X\_resample, y\_resample = sm.fit\_resample(X, y)  X\_resample[:, 0] = np.round(X\_resample[:, 0], 0)  X\_resample[:, 1] = np.round(X\_resample[:, 1], 1)  X\_resample[:, 2] = np.round(X\_resample[:, 2], 1)  X\_resample[:, 3] = np.round(X\_resample[:, 3], 0)  X\_resample[:, 4] = np.round(X\_resample[:, 4], 1)  X\_resample[:, 5] = np.round(X\_resample[:, 5], 0)  data\_resample = np.append(X\_resample, np.ones((len(y\_resample), 1)), axis=1)  data\_resample = np.append(data\_resample, y\_resample.reshape(-1, 1), axis=1)  data\_resample = pd.DataFrame(data\_resample, columns=data\_neg.columns)  return data\_resample.loc[data\_resample["具体故障类别"] == sort]  Normal = data.loc[data["具体故障类别"] == "Normal"]  HDF\_resample = resample(data, sort="HDF", number=686, k\_neighbors=5)  OSF\_resample = resample(data, sort="OSF", number=614, k\_neighbors=5)  PWF\_resample = resample(data, sort="PWF", number=535, k\_neighbors=5)  RNF\_resample = resample(data, sort="RNF", number=43, k\_neighbors=3)  TWF\_resample = resample(data, sort="TWF", number=296, k\_neighbors=5)  data\_resample = pd.concat([Normal, HDF\_resample, OSF\_resample, PWF\_resample, RNF\_resample, TWF\_resample], axis=0)  data\_resample["机器质量等级"] = data\_resample["机器质量等级"].astype(int)  data\_resample["转速（rpm）"] = data\_resample["转速（rpm）"].astype(int)  data\_resample["使用时长（min）"] = data\_resample["使用时长（min）"].astype(int)  data\_resample["是否发生故障"] = data\_resample["是否发生故障"].astype(int)  # data\_resample.to\_csv("train\_data\_resample.csv", index=False, encoding='utf-8-sig') |

|  |
| --- |
| 附录2 |
| test\_sample.py：随机选取原始数据作为测试集 |
| import pandas as pd  data = pd.read\_excel("train data.xlsx", engine='openpyxl')  data\_resample = pd.read\_csv("train\_data\_resample.csv")  data = data.drop(index=data[(data["具体故障类别"] == "Normal") & (data["是否发生故障"] == 1)].index.tolist())  data = data.drop(["机器编号"], axis=1)  data = data.drop(["统一规范代码"], axis=1)  def class\_dict(m):  class\_dictionary = {'H': 0, 'M': 1, 'L': 2}  return class\_dictionary[m]  data["机器质量等级"] = data["机器质量等级"].map(lambda x : list(map(class\_dict, x))[0])  data\_test = data.sample(random\_state=0, frac=0.2)  for \_, row in data\_test.iterrows():  for i in data\_resample.index:  if i in data\_resample.index:  if (row == data\_resample.loc[i]).all():  data\_resample = data\_resample.drop(index=i)  # data\_train.to\_csv("train\_data.csv", index=False, encoding='utf-8-sig')  # data\_test.to\_csv("test\_data.csv", index=False, encoding='utf-8-sig') |

|  |
| --- |
| 附录3 |
| rf\_parameter\_adjust.py：RF模型参数调优 |
| # import numpy as np  import pandas as pd  from sklearn.model\_selection import GridSearchCV  from sklearn.metrics import classification\_report  from sklearn.metrics import confusion\_matrix  from sklearn.ensemble import RandomForestClassifier  # 读取数据  data\_train = pd.read\_csv("train\_data.csv").sample(frac=1)  data\_test = pd.read\_csv("test\_data.csv")  X\_train, y\_train = data\_train.values[:, :-2], data\_train.values[:, -2].astype(int)  X\_test, y\_test = data\_test.values[:, :-2], data\_test.values[:, -2].astype(int)  # 设置网格搜索范围  criterion = ['gini', 'entropy'] # 决策树划分标准  n\_estimators = [80, 100, 120] # 决策树个数  max\_depth = [30, 40, 50] # 树的最大深度  min\_samples\_split = [1, 2, 3] # 可以被分割的所含最小样本数  min\_samples\_leaf = [1, 2, 3] # 叶子节点所含最少样本数  max\_features = [0.4, 0.5, 0.6] # 每棵树用到的最大特征数  param\_grid = {'criterion': criterion, 'n\_estimators': n\_estimators, 'max\_depth': max\_depth,  'min\_samples\_split': min\_samples\_split, 'min\_samples\_leaf': min\_samples\_leaf,  'max\_features': max\_features}  # 进行网格搜索与交叉验证  rf = RandomForestClassifier(random\_state=0)  grid = GridSearchCV(estimator=rf, param\_grid=param\_grid, cv=5, scoring='accuracy', verbose=2)  grid.fit(X\_train, y\_train)  print('网格搜索-度量记录：', grid.cv\_results\_) # 包含每次训练的相关信息  print('网格搜索-最佳度量值:', grid.best\_score\_) # 获取最佳度量值  print('网格搜索-最佳参数：', grid.best\_params\_) # 获取最佳度量值时的代定参数的值。是一个字典  print('网格搜索-最佳模型：', grid.best\_estimator\_) # 获取最佳度量时的分类器模型  # 使用获取的最佳参数生成模型预测数据  rf\_best = grid.best\_estimator\_  y\_rf\_predict = rf\_best.fit(X\_train, y\_train).predict(X\_test)  print("混淆矩阵：")  print(confusion\_matrix(y\_test, y\_rf\_predict))  print("RF得分：")  print(classification\_report(y\_test, y\_rf\_predict, digits=5)) |

|  |
| --- |
| 附录4 |
| gbdt\_parameter\_adjust.py：GBDT模型参数调优 |
| # import numpy as np  import pandas as pd  from sklearn.model\_selection import GridSearchCV  from sklearn.metrics import classification\_report  from sklearn.metrics import confusion\_matrix  from sklearn.ensemble import GradientBoostingClassifier  # 读取数据  data\_train = pd.read\_csv("train\_data.csv").sample(frac=1)  data\_test = pd.read\_csv("test\_data.csv")  X\_train, y\_train = data\_train.values[:, :-2], data\_train.values[:, -2].astype(int)  X\_test, y\_test = data\_test.values[:, :-2], data\_test.values[:, -2].astype(int)  # 设置网格搜索范围  max\_depth = [10, 15, 20] # 树的最大深度  min\_samples\_split = [15, 30, 50] # 可以被分割的所含最小样本数  min\_samples\_leaf = [4, 5, 6] # 叶子节点所含最少样本数  max\_features = [0.4, 0.5, 0.6] # 每棵树用到的最大特征数  param\_grid = {'max\_depth': max\_depth, 'min\_samples\_split': min\_samples\_split,  'min\_samples\_leaf': min\_samples\_leaf, 'max\_features': max\_features}  # 进行网格搜索与交叉验证  gbdt = GradientBoostingClassifier(random\_state=0)  grid = GridSearchCV(estimator=gbdt, param\_grid=param\_grid, cv=5, scoring='accuracy', verbose=2)  grid.fit(X\_train, y\_train)  print('网格搜索-度量记录：', grid.cv\_results\_) # 包含每次训练的相关信息  print('网格搜索-最佳度量值:', grid.best\_score\_) # 获取最佳度量值  print('网格搜索-最佳参数：', grid.best\_params\_) # 获取最佳度量值时的代定参数的值。是一个字典  print('网格搜索-最佳模型：', grid.best\_estimator\_) # 获取最佳度量时的分类器模型  # 使用获取的最佳参数生成模型预测数据  gbdt\_best = grid.best\_estimator\_  y\_gbdt\_predict = gbdt\_best.fit(X\_train, y\_train).predict(X\_test)  print("混淆矩阵：")  print(confusion\_matrix(y\_test, y\_gbdt\_predict))  print("GBDT得分：")  print(classification\_report(y\_test, y\_gbdt\_predict, digits=5)) |

|  |
| --- |
| 附录5 |
| xgboost\_parameter\_adjust.py：XGBoost模型参数调优 |
| # import numpy as np  import pandas as pd  from sklearn.model\_selection import GridSearchCV  from sklearn.metrics import classification\_report  from sklearn.metrics import confusion\_matrix  from xgboost import XGBClassifier  import warnings  warnings.filterwarnings("ignore")  # 读取数据  data\_train = pd.read\_csv("train\_data.csv").sample(frac=1)  data\_test = pd.read\_csv("test\_data.csv")  X\_train, y\_train = data\_train.values[:, :-2], data\_train.values[:, -2].astype(int)  X\_test, y\_test = data\_test.values[:, :-2], data\_test.values[:, -2].astype(int)  # 设置网格搜索范围  max\_depth = [20, 30, 50] # 树的最大深度  min\_child\_weight = [1, 2] # 决定最小叶子节点样本权重和  gamma = [0] # 惩罚项系数  param\_grid = {'max\_depth': max\_depth, 'min\_child\_weight': min\_child\_weight, 'gamma': gamma}  # 进行网格搜索与交叉验证  xgboost = XGBClassifier(random\_state=0)  grid = GridSearchCV(estimator=xgboost, param\_grid=param\_grid, cv=5, scoring='accuracy', verbose=2)  grid.fit(X\_train, y\_train)  print('网格搜索-度量记录：', grid.cv\_results\_) # 包含每次训练的相关信息  print('网格搜索-最佳度量值:', grid.best\_score\_) # 获取最佳度量值  print('网格搜索-最佳参数：', grid.best\_params\_) # 获取最佳度量值时的代定参数的值。是一个字典  print('网格搜索-最佳模型：', grid.best\_estimator\_) # 获取最佳度量时的分类器模型  # 使用获取的最佳参数生成模型预测数据  xgboost\_best = grid.best\_estimator\_  y\_xgboost\_predict = xgboost\_best.fit(X\_train, y\_train).predict(X\_test)  print("混淆矩阵：")  print(confusion\_matrix(y\_test, y\_xgboost\_predict))  print("XGBoost得分：")  print(classification\_report(y\_test, y\_xgboost\_predict, digits=5)) |

|  |
| --- |
| 附录6 |
| lgbm\_parameter\_adjust.py：LightGBM模型参数调优 |
| # import numpy as np  import pandas as pd  from sklearn.model\_selection import GridSearchCV  from sklearn.metrics import classification\_report  from sklearn.metrics import confusion\_matrix  from lightgbm import LGBMClassifier  import warnings  warnings.filterwarnings("ignore")  # 读取数据  data\_train = pd.read\_csv("train\_data.csv").sample(frac=1)  data\_test = pd.read\_csv("test\_data.csv")  X\_train, y\_train = data\_train.values[:, :-2], data\_train.values[:, -2].astype(int)  X\_test, y\_test = data\_test.values[:, :-2], data\_test.values[:, -2].astype(int)  # 设置网格搜索范围  n\_estimators = [30, 50, 80, 100] # 拟合的树的棵树  max\_depth = [5, 10, 15, 20] # 树的最大深度  min\_child\_weight = [4, 5, 6, 8, 10] # 决定最小叶子节点样本权重和  param\_grid = {'n\_estimators': n\_estimators, 'max\_depth': max\_depth, 'min\_child\_weight': min\_child\_weight}  # 进行网格搜索与交叉验证  lgbm = LGBMClassifier(random\_state=0)  grid = GridSearchCV(estimator=lgbm, param\_grid=param\_grid, cv=5, scoring='accuracy', verbose=2)  grid.fit(X\_train, y\_train)  print('网格搜索-度量记录：', grid.cv\_results\_) # 包含每次训练的相关信息  print('网格搜索-最佳度量值:', grid.best\_score\_) # 获取最佳度量值  print('网格搜索-最佳参数：', grid.best\_params\_) # 获取最佳度量值时的代定参数的值。是一个字典  print('网格搜索-最佳模型：', grid.best\_estimator\_) # 获取最佳度量时的分类器模型  # 使用获取的最佳参数生成模型预测数据  lgbm\_best = grid.best\_estimator\_  y\_lgbm\_predict = lgbm\_best.fit(X\_train, y\_train).predict(X\_test)  print("混淆矩阵：")  print(confusion\_matrix(y\_test, y\_lgbm\_predict))  print("LightGBM得分：")  print(classification\_report(y\_test, y\_lgbm\_predict, digits=5)) |

|  |
| --- |
| 附录7 |
| Ensemble\_task2.py：构建任务二的集成模型并测试精度 |
| import numpy as np  import pandas as pd  from sklearn.ensemble import VotingClassifier  from sklearn.ensemble import RandomForestClassifier  from sklearn.ensemble import GradientBoostingClassifier  from lightgbm import LGBMClassifier  from xgboost import XGBClassifier  from sklearn.metrics import classification\_report  from sklearn.metrics import confusion\_matrix  from sklearn.metrics import accuracy\_score  # 读取数据  data\_train = pd.read\_csv("train\_data.csv").sample(frac=1)  data\_test = pd.read\_csv("test\_data.csv")  X\_train, y\_train = data\_train.values[:, :-2], data\_train.values[:, -2].astype(int)  X\_test, y\_test = data\_test.values[:, :-2], data\_test.values[:, -2].astype(int)  # 构建模型  rf = RandomForestClassifier(random\_state=0, criterion='entropy', n\_estimators=120, max\_depth=30,  min\_samples\_split=2, min\_samples\_leaf=1, max\_features=0.5)  gbdt = GradientBoostingClassifier(random\_state=0, max\_depth=20, min\_samples\_split=15,  min\_samples\_leaf=6, max\_features=0.4)  xgbt = XGBClassifier(random\_state=0, max\_depth=20, min\_child\_weight=2, gamma=0)  lgbm = LGBMClassifier(random\_state=0, n\_estimators=80, max\_depth=20, min\_child\_weight=4)  # 模型的训练与预测  y\_rf\_predict = rf.fit(X\_train, y\_train).predict(X\_test)  y\_gbdt\_predict = gbdt.fit(X\_train, y\_train).predict(X\_test)  y\_lgbm\_predict = lgbm.fit(X\_train, y\_train).predict(X\_test)  y\_xgbt\_predict = xgbt.fit(X\_train, y\_train).predict(X\_test)  # 模型的集成  voting = VotingClassifier(estimators=[('rf', rf), ('gbdt', gbdt), ('lgbm', lgbm), ('xgbt', xgbt)],  voting='soft', weights=[accuracy\_score(y\_test, y\_rf\_predict),  accuracy\_score(y\_test, y\_gbdt\_predict),  accuracy\_score(y\_test, y\_lgbm\_predict),  accuracy\_score(y\_test, y\_xgbt\_predict)])  y\_voting\_predict = voting.fit(X\_train, y\_train).predict(X\_test)  print("RF得分：")  print(classification\_report(y\_test, y\_rf\_predict, digits=5))  print("GBDT得分：")  print(classification\_report(y\_test, y\_gbdt\_predict, digits=5))  print("LightGBM得分：")  print(classification\_report(y\_test, y\_lgbm\_predict, digits=5))  print("XGBoost得分：")  print(classification\_report(y\_test, y\_xgbt\_predict, digits=5))  print("集成模型得分：")  print(classification\_report(y\_test, y\_voting\_predict, digits=5))  print("混淆矩阵：")  print(confusion\_matrix(y\_test, y\_voting\_predict)) |

|  |
| --- |
| 附录8 |
| Ensemble\_task3.py：构建任务三的集成模型并测试精度 |
| import numpy as np  import pandas as pd  from sklearn.ensemble import VotingClassifier  from sklearn.ensemble import RandomForestClassifier  from sklearn.ensemble import GradientBoostingClassifier  from lightgbm import LGBMClassifier  from xgboost import XGBClassifier  from sklearn.metrics import classification\_report  from sklearn.metrics import accuracy\_score  # 读取数据  data\_train = pd.read\_csv("train\_data.csv").sample(frac=1)  data\_test = pd.read\_csv("test\_data.csv")  data\_train = data\_train.loc[data\_train["具体故障类别"] != "Normal"]  data\_test = data\_test.loc[data\_test["具体故障类别"] != "Normal"]  X\_train, y\_train = data\_train.values[:, :-2], data\_train.values[:, -1]  X\_test, y\_test = data\_test.values[:, :-2], data\_test.values[:, -1]  # 构建模型  rf = RandomForestClassifier(random\_state=0)  gbdt = GradientBoostingClassifier(random\_state=0)  lgbm = LGBMClassifier(random\_state=0)  xgbt = XGBClassifier(random\_state=0)  # 模型的训练与预测  y\_rf\_predict = rf.fit(X\_train, y\_train).predict(X\_test)  y\_gbdt\_predict = gbdt.fit(X\_train, y\_train).predict(X\_test)  y\_lgbm\_predict = lgbm.fit(X\_train, y\_train).predict(X\_test)  y\_xgbt\_predict = xgbt.fit(X\_train, y\_train).predict(X\_test)  # 模型的集成  voting = VotingClassifier(estimators=[('rf', rf), ('gbdt', gbdt), ('lgbm', lgbm), ('xgbt', xgbt)],  voting='soft', weights=[accuracy\_score(y\_test, y\_rf\_predict),  accuracy\_score(y\_test, y\_gbdt\_predict),  accuracy\_score(y\_test, y\_lgbm\_predict),  accuracy\_score(y\_test, y\_xgbt\_predict)])  y\_voting\_predict = voting.fit(X\_train, y\_train).predict(X\_test)  print("RF得分：")  print(classification\_report(y\_test, y\_rf\_predict, digits=5))  print("GBDT得分：")  print(classification\_report(y\_test, y\_gbdt\_predict, digits=5))  print("LightGBM得分：")  print(classification\_report(y\_test, y\_lgbm\_predict, digits=5))  print("XGBoost得分：")  print(classification\_report(y\_test, y\_xgbt\_predict, digits=5))  print("集成模型得分：")  print(classification\_report(y\_test, y\_voting\_predict, digits=5)) |

|  |
| --- |
| 附录9 |
| forecast.py：预测任务四的数据 |
| import numpy as np  import pandas as pd  from sklearn.ensemble import VotingClassifier  from sklearn.ensemble import RandomForestClassifier  from sklearn.ensemble import GradientBoostingClassifier  from lightgbm import LGBMClassifier  from xgboost import XGBClassifier  # 读取数据  data\_train = pd.read\_csv("train\_data.csv").sample(frac=1)  data\_test = pd.read\_csv("test\_data.csv")  X\_train, y\_train = data\_train.values[:, :-2], data\_train.values[:, -2].astype(int)  data = pd.read\_excel("forecast.xlsx", engine='openpyxl')  X\_data = data.copy()  X\_data = X\_data.drop(["机器编号"], axis=1)  X\_data = X\_data.drop(["统一规范代码"], axis=1)  def class\_dict(m):  class\_dictionary = {'H': 0, 'M': 1, 'L': 2}  return class\_dictionary[m]  X\_data["机器质量等级"] = X\_data["机器质量等级"].map(lambda x : list(map(class\_dict, x))[0])  X\_data = X\_data.values  # 构建模型  rf1 = RandomForestClassifier(random\_state=0, criterion='entropy', n\_estimators=120, max\_depth=30,  min\_samples\_split=2, min\_samples\_leaf=1, max\_features=0.5)  gbdt1 = GradientBoostingClassifier(random\_state=0, max\_depth=20, min\_samples\_split=15,  min\_samples\_leaf=6, max\_features=0.4)  xgbt1 = XGBClassifier(random\_state=0, max\_depth=20, min\_child\_weight=2, gamma=0)  lgbm1 = LGBMClassifier(random\_state=0, n\_estimators=80, max\_depth=20, min\_child\_weight=4)  # 模型的训练与预测  y\_rf\_predict = rf1.fit(X\_train, y\_train).predict(X\_data)  y\_gbdt\_predict = gbdt1.fit(X\_train, y\_train).predict(X\_data)  y\_lgbm\_predict = lgbm1.fit(X\_train, y\_train).predict(X\_data)  y\_xgbt\_predict = xgbt1.fit(X\_train, y\_train).predict(X\_data)  # 模型的集成  voting1 = VotingClassifier(estimators=[('rf', rf1), ('gbdt', gbdt1), ('xgbt', xgbt1), ('lgbm', lgbm1)],  voting='soft', weights=[0.9850, 0.9867, 0.9844, 0.9806])  y\_voting\_predict1 = voting1.fit(X\_train, y\_train).predict(X\_data)  data["是否发生故障"] = y\_voting\_predict1  # 构建模型  rf2 = RandomForestClassifier(random\_state=0)  gbdt2 = GradientBoostingClassifier(random\_state=0)  lgbm2 = LGBMClassifier(random\_state=0)  xgbt2 = XGBClassifier(random\_state=0)  # 模型的训练与预测  data\_train = data\_train.loc[data\_train["具体故障类别"] != "Normal"]  X\_train, y\_train = data\_train.values[:, :-2], data\_train.values[:, -1]  y\_rf\_predict2 = rf2.fit(X\_train, y\_train).predict(X\_data)  y\_gbdt\_predict2 = gbdt2.fit(X\_train, y\_train).predict(X\_data)  y\_lgbm\_predict2 = lgbm2.fit(X\_train, y\_train).predict(X\_data)  y\_xgbt\_predict2 = xgbt2.fit(X\_train, y\_train).predict(X\_data)  # 模型的集成  voting2 = VotingClassifier(estimators=[('rf', rf2), ('gbdt', gbdt2), ('lgbm', lgbm2), ('xgbt', xgbt2)], voting='soft')  y\_voting\_predict2 = voting2.fit(X\_train, y\_train).predict(X\_data)  for i in range(len(X\_data)):  if y\_voting\_predict1[i] == 0:  y\_voting\_predict2[i] = "Normal"  data["具体故障类别"] = y\_voting\_predict2  data.to\_csv("forecast.csv", encoding='utf-8-sig') |

|  |
| --- |
| 附录10 |
| DTree.py：构建任务五的决策树模型并可视化 |
| import pandas as pd  import pydotplus  from sklearn import tree  from IPython.display import Image  from sklearn.preprocessing import LabelEncoder  # 读取文件并进行预处理  data = pd.read\_excel("train data.xlsx", engine='openpyxl')  data = data.drop(index=data[(data["具体故障类别"] == "Normal") & (data["是否发生故障"] == 1)].index.tolist())  data = data.drop(['机器编号'], axis=1)  data = data.drop(['统一规范代码'], axis=1)  data = data.drop(['是否发生故障'], axis=1)  def class\_dict(m):  class\_dictionary = {'H': 0, 'M': 1, 'L': 2}  return class\_dictionary[m]  data["机器质量等级"] = data["机器质量等级"].map(lambda x : list(map(class\_dict, x))[0])  data['具体故障类别'] = LabelEncoder().fit\_transform(data['具体故障类别'])  # # 分箱处理  # labels = [1, 2, 3, 4, 5]  # data["室温（K）"] = pd.qcut(data["室温（K）"], 5, labels=labels)  # data["室温（K）.1"] = pd.qcut(data["室温（K）.1"], 5, labels=labels)  # data["转速（rpm）"] = pd.qcut(data["转速（rpm）"], 5, labels=labels)  # data["扭矩（Nm）"] = pd.qcut(data["扭矩（Nm）"], 5, labels=labels)  # data["使用时长（min）"] = pd.qcut(data["使用时长（min）"], 5, labels=labels)  # 划分特征与标签  X\_train, y\_train = data.values[:, :-1], data.values[:, -1].astype(int)  # 构建决策树模型  clf = tree.DecisionTreeClassifier(random\_state=0)  clf.fit(X\_train, y\_train)  # 绘制决策树  dot\_data = tree.export\_graphviz(clf,  feature\_names=['quality', 'temperature1', 'temperature2', 'rotate', 'torque', 'time'],  class\_names=['0', '1', '2', '3', '4', '5'],  out\_file=None,  filled=True,  rounded=True,  special\_characters=True)  graph = pydotplus.graph\_from\_dot\_data(dot\_data)  graph.write\_png("决策树2.png")  Image(graph.create\_png()) |