所在组别	“未来杯”2024 年第四届高校大数据挑战赛	参赛编号
研究生组		bdc240955

基于机器学习方法的电动汽车故障报警因素分析及故障报警预测

摘要

在全球能源结构绿色转型加速推进的背景下，新能源电动汽车凭借其零排放、高能效的技术优势，正从新兴交通工具转变为全球交通发展的战略方向。值得注意的是，动力电池系统作为电动汽车的核心组件，在长期复杂工况运行中可能产生绝缘失效、电压异常、热失控等多模态故障，这些故障不仅影响车辆续航能力与动力性能，更可能引发热扩散、短路起火等安全事故。基于此，构建高精度的电池系统健康状态评估模型与智能预警机制，已成为突破新能源汽车安全瓶颈、保障行业健康发展的重要技术突破口。本文基于 2022 年 1-9 月某辆电动汽车的使用过程中产生的各项行驶数据，通过采用机器学习的方法对影响电动汽车不同级别故障报警的主要影响因素进行分析，并且基于分析结果设计多种模型通过相关评估指标进行评估验证找到最优模型用于对 10 月可能出现的故障报警时刻进行预测。

针对问题一：在数据预处理阶段，本文将数据集中的时间戳数据转换为标准日期格式，并按照时间顺序进行排序。同时，对于部分相关字段，若存在格式异常（如以“变量名:值”形式存储），则提取数值部分进行转换。本文还删除了关键字段（如时间戳、总电压和总电流）存在缺失值的记录，并对其他数值型字段（如速度、总里程、SOC 等）采用中位数填充，以减少数据不完整对分析的影响。最后本文还添加了**异常检测**，首先采用 Z-Score 方法识别电压、电流、SOC 等关键参数的异常值；其次，本文利用 IQR 方法检测电机温度等变量的异常范围；同时，本文构建**孤立森林**（Isolation Forest）模型来对车辆运行数据进行全局异常检测最终将每个月的异常报告返回。完成数据预处理后本文对汽车每个月发生的故障报警次数及其变化规律以及车主开车习惯进行分析并使用多种可视化的方法将其规律展现出来。

针对问题二：本文利用处理好后的数据集首先通过时间特征提取、电压变化率计算、温度梯度分析及充电状态编码等方法构建了一系列关键特征，随后，本文采用**随机森林**模型作为故障预测模型进行训练以此分析造成电动汽车不同级别故障报警的主要影响因素并且通过使用**特征重要性分析**、**SHAP 分析**及**部分依赖分析**（PDP）这些方法将关键变量的影响进行可视化从而提升结果的透明度与实用性。

针对问题三：本文结合问题二所得到的影响电动汽车故障报警的关键影响变量，通过建立多种不同类别的模型（包括线性回归模型：**岭回归**和 Lasso 回归；机器学习模型：**随机森林模型**、XGBoost 模型、SVM 模型和 LightGBM 模型；深度学习模型:LSTM 和 GCN）并且选取 AUC 和 Recall 作为评估指标来进行评估验证以此找到最优模型。

针对问题四：本文根据问题三所找到的最优模型来完成对该电动汽车 10 月份的可能会出现的故障报警时刻以及报警等级的预测任务，同时为了评估模型预测结果对不同阈值的依赖性，本来还对模型进行完整的**敏感度分析**。

针对问题五：本文结合前面四项任务所完成的工作和得到的结论从汽车充电、故障检修、用车规律等方面，给出该车辆的使用建议。

关键词：电动汽车故障报警分析，异常检测，随机森林,线性回归模型，机器学习模型，深度学习模型

一、 问题背景与重述

1.1 问题背景

在全球环境保护意识提升和能源结构转型的推动下，新能源电动汽车凭借其清洁、高效、低碳排放的特点，逐渐成为未来交通出行的主流选择。然而，相较于传统燃油车，电动汽车集成了更加复杂的电子控制系统和高压电池组件。这些先进技术的应用在提升车辆性能的同时，也使得系统运行环境更加复杂，增加了潜在的故障风险。

同时随着电动汽车使用时间的增长及运行环境的变化，高压电池组件可能出现多种故障，例如过充自燃（电池在充电饱和后继续充电，导致热失控甚至爆炸）或过放电故障（电池电压低于安全阈值，影响电池寿命与车辆正常运行）等。这些故障不仅会降低车辆的运行稳定性，还可能引发严重的安全事故，威胁乘员生命安全。因此，针对高压电池系统的故障诊断与预警技术研究，已成为新能源电动汽车领域的核心技术挑战，对提升整车安全性和用户体验具有重要意义。

在新能源电动汽车高压电池故障诊断与预警领域，国内外学者已开展了大量研究，主要集中在基于物理模型的方法、数据驱动的方法及融合方法等方面。在国外研究中，基于等效电路模型的方法被广泛应用于电池健康状态估计与故障诊断。例如，Plett 等[1]提出了基于扩展卡尔曼滤波（EKF）的电池状态估计方法，能够有效追踪电池内部参数变化，提高故障检测精度。此外，近年来机器学习和深度学习方法在电池故障预测中的应用日益增多。Tang et al.[2]采用长短时记忆网络（LSTM)进行电池剩余寿命预测，并结合异常检测技术，实现早期故障预警。在国内研究中，研究重点逐步从传统物理建模方法转向数据驱动和多源信息融合。例如王军等人[3]结合随机森林和支持向量机，提出一种基于多维特征提取的电池故障诊断模型，在多工况条件下表现优异。此外，吴强等人[4]研究了基于贝叶斯网络的电池健康状态预测方法，通过历史数据分析提高故障预测的可靠性。

1.2 问题重述

问题 1：仔细查看“附件”中提供的汽车数据，首先，进行数据预处理，包括处理缺失值和识别及修正异常值。统计 1 月至 9 月期间，该汽车每个月发生的故障报警次数及其变化规律，并分析总结该电动汽车的车主充电和用车规律，最好用多样的可视化展示。

问题 2：通常情况下，电动汽车的车辆数据、电机数据以及电池数据内部相互之间可能存在某种关联性，数据中包含有 0-3 级别的故障报警指标，请根据附件中 1-9 月的汽车数据，选择合适的大数据方法找出造成电动汽车不同级别故障报警的主要影响因素。

问题 3：请根据附件中 1-9 月的汽车数据，结合问题二的分析结果，构建一个电动汽车故障预警模型，并对模型进行检验，最好能建立多个模型进行比较。

问题 4：附件中第 10 个月的数据缺失故障报警信息。请根据问题三中建立的模型预测第 10 个月该车辆可能会出现的故障报警时刻（填写“month_10.csv”文件中的时间点即可）及报警等级（示表中只需要填写故障等级 1-3 的情况），并将预测结果展示在报告中。

问题 5：结合前面问题的分析，从充电、故障检修、用车规律等方面，给出该车辆的使用建议。

二、 问题分析

为了解决上述提出的研究问题，本文采用以下求解思路进行数据处理与模型建立，其中问题及解决方法框架如图 1 所示：

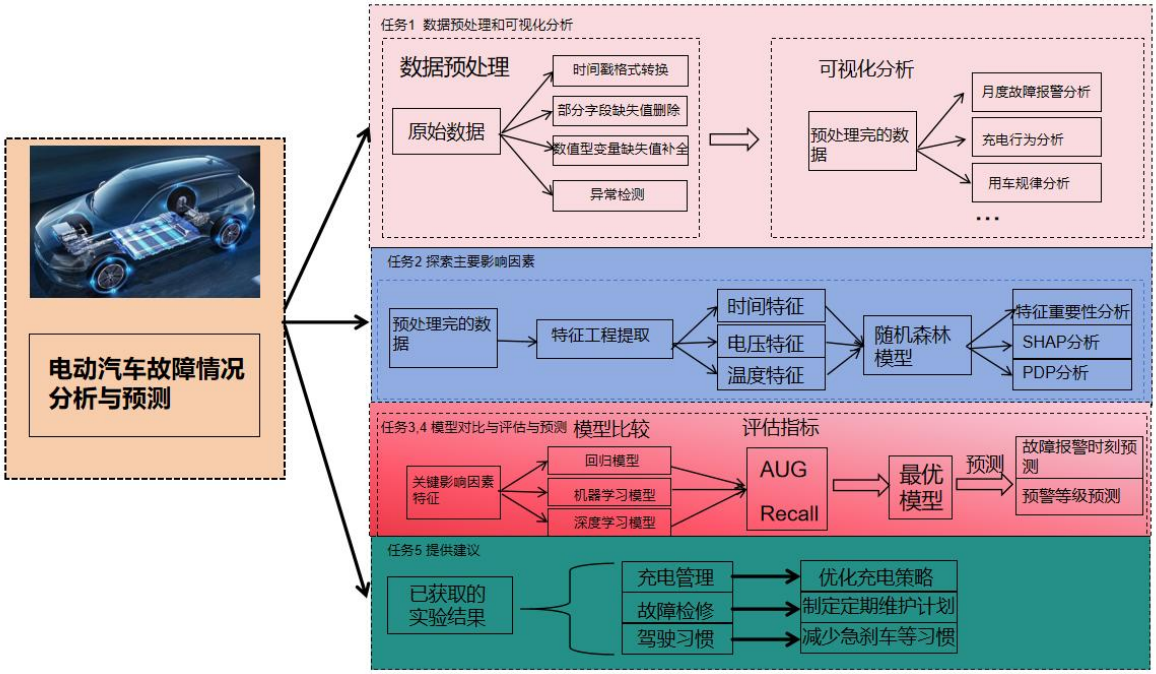

图 1 论文整体实现框架

问题 1：数据预处理和可视化分析

在数据预处理阶段，本文首先对原始数据进行时间格式标准化以及相关字段的规范化转换，以保证数据的可读性和一致性。针对数据缺失问题，本文采取不同策略进行处理。对于关键字段（如时间戳、总电压、总电流），若存在缺失值，则直接剔除相应记录，以避免对后续分析产生较大影响。而对于其他数值型变量（如速度、总里程、SOC 等），本文采用中位数填充的方式进行补全，以减少异常值对整体数据分布的影响，从而提高数据的稳定性和可靠性。此外，本文特别引入异常检测方法，以识别并剔除可能影响分析结果的异常数据点。首先，采用 Z-Score 方法对电压、电流、SOC 等关键变量进行异常值识别，以检测显著偏离正常范围的数值。其次，利用 IQR 方法检测电机温度等变量的异常范围，以进一步剔除极端值。最后，为了提高异常检测的全面性，本文构建孤立森林模型，对该车运行数据进行全局异常检测，并最终汇总每月的异常报告，以便后续分析和优化。完成数据预处理后，本文进一步分析汽车每月发生的故障报警次数及其变化规律，并结合车主驾驶习惯进行模式挖掘。为直观呈现分析结果，本文采用多种可视化方法（如时间序列图、散点图、热力图等）对数据进行呈现，从而揭示新能源汽车在实际运行中的故障分布趋势及影响因素，为后续故障预警模型的构建提供可靠的数据支撑。

问题 2：探索造成电动汽车不同级别故障报警的主要影响因素

本文基于处理后的数据集，首先进行特征工程，以提取能够有效表征电动汽车故障风险的关键变量。具体而言，本文通过时间特征提取方法，构造了诸如小时、是否是周末等时间相关特征，以捕捉故障发生的时间模式。此外，本文计算了电压变化率（即总电压的相对变化幅度）来衡量电池系统的稳定性，并引入温度梯度分析方法，以量化电机温度在一定时间窗口内的变化趋势，从而评估温度波动对故障发生的潜在影响。在故障预测建模阶段，本文选择随机森林模型作为主要分类器，对不同故障级别进行预测。训练完成后，本文进一步采用特征重要性分析、SHAP 分析和部分依赖分析（PDP）等方法，

对影响故障发生的关键变量进行解释性分析并且实现可视化展现。

问题 3：模型建立与评估

基于问题二所识别出的影响电动汽车故障报警的关键变量，本文进一步构建并比较多种不同类别的模型，以寻找最优的故障预测方案。具体而言，本文涵盖了线性回归模型（岭回归与 Lasso 回归）、传统机器学习模型（随机森林、XGBoost、支持向量机 SVM 以及 LightGBM）以及深度学习模型（长短时记忆网络 LSTM 与图卷积网络 GCN），从不同建模视角探索故障报警的预测能力。为了确保评估的全面性和可靠性，本文选取 AUC 和 Recall 作为核心评估指标。其中，AUC 衡量模型的整体分类能力，而 Recall 关注故障报警的识别率，确保模型在安全性关键场景下的有效性。最终，通过实验分析不同模型在各项指标上的表现，本文筛选出最优模型以此来完成后续任务。

问题 4：故障报警预测

在确定最优模型后，本文利用该模型对电动汽车 10 月份 可能发生的故障报警进行预测，涵盖具体的报警时刻及其报警等级。通过输入车辆运行数据，模型能够识别潜在的异常模式，并提前预警可能发生的故障。

问题 5：提供建议

基于前述四项任务的研究成果及分析结论，本文从充电管理、故障检修、驾驶习惯等多个方面，为该电动汽车的使用提供科学合理的建议。在充电方面，本文建议优化充电策略，避免过充或过放导致电池衰减，提高充电效率。在故障检修方面，结合故障预测模型的分析结果，制定定期维护计划，重点关注高风险部件，如电池、电机及高压电控系统，以降低突发故障的概率。此外，针对用户的驾驶规律分析，本文提出优化驾驶行为的建议，例如减少急加速、急刹车等操作，以延长电池寿命并提高整体车辆的运行稳定性。

三、 模型假设

为了有效实现电动汽车故障分析与预测，本研究提出以下模型假设：

假设 1：车辆数据、电机数据和电池数据之间存在潜在的关联模式

新能源汽车的故障报警不仅仅由单一因素决定，而是多个变量综合作用的结果。例如，电池 SOC、电机温度和总电压等参数的波动可能同时影响车辆的故障状态。因此基于历史数据，可以通过特征工程和机器学习方法提取关键变量，并建立高效的预测模型。

假设 2：电动汽车的故障发生具有时间依赖性和规律性

故障报警的发生可能受季节、环境温度、驾驶行为及充放电模式的影响。故障的发生并不是随机的，而是随着行驶时间、累计里程及车辆使用频率呈现出一定的变化趋势。

假设 3：历史故障数据可以用于训练有效的预警模型

假设过去 1-9 月的数据可以用于学习电动汽车的故障模式，即历史报警数据能够提供足够的信息来预测未来可能发生的故障。基于此，可以利用机器学习（如随机森林、XGBoost）或深度学习（如 LSTM、GCN）等方法训练模型，从而在 10 月份的数据缺失情况下进行准确预测。

假设 4：不同故障等级的报警受到不同因素的影响

不同级别（0-3 级）的故障报警由不同的关键因素驱动。例如，低级别的故障可能更多与轻微电压波动或温度变化相关，而高级别的故障可能涉及 SOC 异常、过充过放、电池单体电压差异过大等。

四、 符号说明

本文主要变量符号说明：

符号	说明	单位
Δt	两个时刻的电压变化率	V
∇T	温度变化梯度	℃
$X_{总电压}$	电池组总电压	V
$X_{车速}$	车辆速度	km/h
$X_{总电流}$	电池组总电流	A

注：未申明的变量以其在符号出现处的具体说明为准

五、 问题 1：数据预处理及可视化分析

5.1 数据预处理

由于在原始数据集中可能存在部分字段格式不规范已经数据缺失的情况，本文首先对 1-9 月的原始数据集进行数据预处理操作。

1. 时间戳格式转换: 本文首先将原始数据中的时间戳格式转换为%Y-%M-%D %H:%M 的标准格式，以便计算机能够识别和进一步分析，例如”2022-01-01 00:00:00”。

2. 缺失值处理：针对数据缺失问题，本文采取不同策略进行处理。首先对于关键字段（总电压、总电流等），若存在缺失值，则直接剔除相应记录，以避免对后续分析产生较大影响。而对于其他数值型变量（如速度、总里程、SOC 等），本文采用中位数填充的方式进行补全。

3. 异常检测：

a. 本文首先采用 Z-Score 方法对电压、电流、SOC 等关键变量进行异常值识别，以检测显著偏离正常范围的数值。Z-Score 方法是一种基于统计学的异常值检测技术，通过衡量数据点与平均值的偏离程度来识别异常。其实现公式为：

$$Z = \frac{X-\mu}{\sigma} \tag{1}$$

其中 X 表示的是单个样本(电压、电流、SOC)的值， μ 和σ分别表示的是该特征的均值和标准差，若某个特征|Z|的值大于 3 则可以判定该值属于异常值，本文通过 Z-Score 方法来对特征进行计算并且通过返回 True/False 来对异常值进行标记，其实现代码如下所示：

算法 1	Z-Score 实现代码
1	zscore_cols = ['total_voltage', 'total_current', 'soc', 'max_cell_voltage']
2	for col in zscore_cols:
3	z_scores = np.abs(stats.zscore(df[col])) # 计算 Z-score 绝对值
4	df[f'zscore_{col}'] = z_scores > zscore_threshold # 标记异常值

b. 本文接下来使用 IQR（四分位距）异常检测方法来检测电机温度变量的异常范围，IQR（四分位距）方法是一种基于统计分布的异常检测方法，广泛用于处理非正态分布数据。其实现公式如下：

$$IQR = Q3 - Q1 \tag{2}$$

其中 Q1 表示数据的 25%分位点，即数据的 25%都要小于 Q1 这个值，Q3 表示数据的 75%分位点，即数据的 75%都要小于 Q3 这个值，然后通过计算 Q3 和 Q1 之间的差值以此来反应数据的趋势，同时若 IQR 的值超出(Q1-1.5*IQR , Q3+1.5*IQR)这个范围则就会被视为异常值，同样本文也通过返回 True/False 来对异常值进行标记，其实现代码如下所示：

算法 2	IQR 实现代码
1	iqr_factor = 1.5
2	for col in ['motor_temp', 'max_temp']:
3	q1 = df[col].quantile(0.25)
4	q3 = df[col].quantile(0.75)
5	iqr = q3 - q1
6	df[f'iqr_{col}'] = (df[col] < (q1 - iqr_factor * iqr)) | (df[col] > (q3 + iqr_factor * iqr))

c. 最后，为了提高异常检测的全面性，本文构建孤立森林模型，对该车运行数据进行全局异常检测，并最终汇总每月的异常报告，以便后续分析和优化。孤立森林是一种基于决策树的思想的机器学习方法，其核心思想是异常值通常与其他数据点距离较远，密度较低，因此它们更容易被孤立。其公式定义如下所示：

$$s(x,n) = 2^{-\frac{E(h(x))}{c(n)}} \tag{3}$$

其中 h(x)表示的是点 x 在孤立树中的平均分裂深度，E(h(x))是多棵树的平均深度，c(n)是用于归一化的常数，表示平均树深度。若 $s(x,n)$接近于 1，则表示该点可能是异常值，若接近 0,5，则表示该点可能是正常值。本文使用 电压、电流、soc 来训练孤立森林模型，检测数据中的异常值。其实现代码如下所示：

算法 3	孤立森林实现代码
1	iso_cols = ['total_voltage', 'total_current', 'soc', 'motor_temp']
2	iso_data = StandardScaler().fit_transform(df[iso_cols])
3	iso_model = IsolationForest(contamination=0.05, random_state=42)
4	df['isolation_forest'] = iso_model.fit_predict(iso_data) == -1

最终通过整合筛选三种方法所得到的异常数据，并分类标注异常类型所得到的每个月该车异常检测报告部分结果如下表 5.1 所示：

表 5.1 电动汽车异常报告

时间	电压	电流	SOC	突变类型
2022-01-01 00:04:00	353.0	37.8	48	SOC 突变
2022-01-01 14:34:00	399.1	0.1	99	温度异常
2022-02-01 10:12:00	400.2	0.8	98	温度异常，SOC 突变
2022-05-31 22:38:00	347.3	74.2	35	SOC 突变
2022-09-12 16:58:00	368.5	10.0	64	SOC 突变

5.2 数据可视化

完成原始数据预处理之后，本文进一步分析汽车每月发生的故障报警次数及其变化规律，并结合车主驾驶习惯进行模式挖掘。为直观呈现分析结果，本文采用多种可视化方法（如时间序列图、散点图、热力图等）对数据进行呈现，可视化操作主要实现的是月

度故障报警次数(以时间戳为单位)可视化分析、充电行为可视化分析、用车规律可视化分析、电池健康可视化分析以及电压-温度-故障三维关系立体可行性分析。

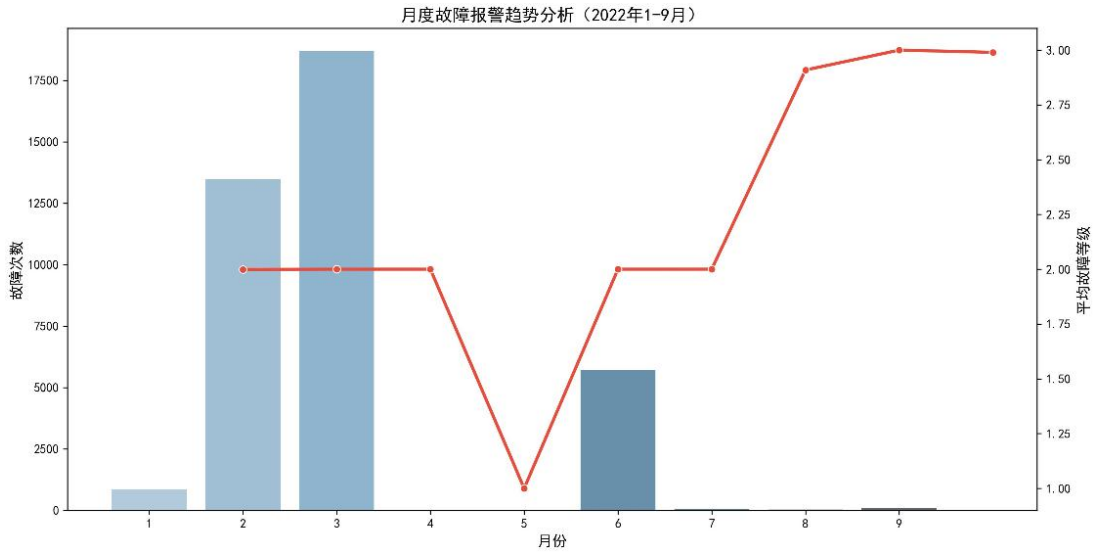

图 2 月度故障报警次数可视化

首先是关于月度故障报警次数的可视化如图 2 所示，柱状图表示故障报警次数，折线图表示平均报警等级数。从柱状图中可以看出，以 1-9 月中每个时间戳为单位，3 月份的故障报警次数最高，接近 18,000 次，2 月份次之，超过 13,000 次，而 6 月份的报警次数约为 7,500 次。1 月和 9 月的报警次数相对较少，7 月和 8 月几乎没有报警记录。这表明故障报警高峰主要集中在 2 月和 3 月，可能与季节性、温度变化或特定外部因素有关。同时从折线图显示，平均报警等级数在 1 月至 4 月基本保持稳定，但在 5 月份降至最低，随后在 6 月至 9 月大幅上升，最高达到 3.0 左右。这表明尽管后期故障报警的绝对数量减少，但每次报警的严重程度可能在增加。总体来看，数据呈现出明显的季节性波动，需要进一步分析故障高发的原因，例如气候、车辆使用频率或电池性能的影响。

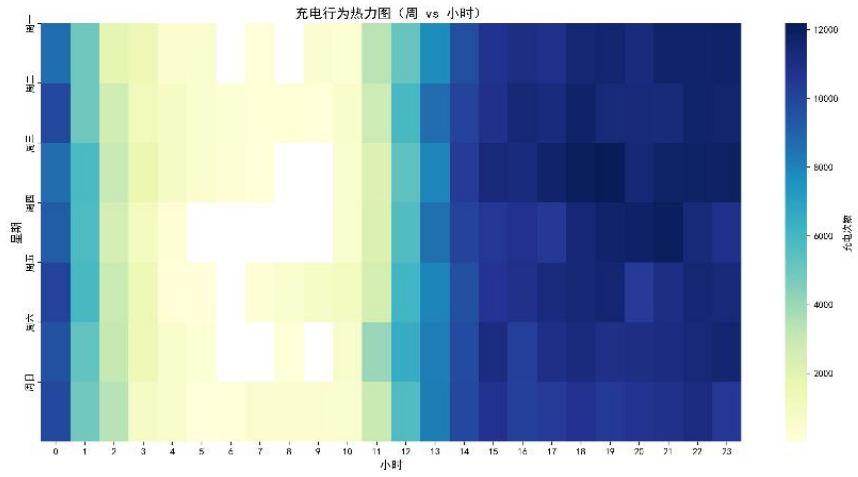

图 3 充电行为可视化

接下来是充电行为可视化分析如图 3 所示，该热力图展示了不同时间段的充电行为分布情况，从图中可以看出，凌晨时段（0-6 点）充电活动相对较少，而在白天的某些时段（8-12 点）充电量有所增加。充电高峰主要集中在下午到晚上（13-23 点），尤其是在 18-22 点时段，充电活动最为活跃。这表明大部分用户可能在工作日白天使用车辆，晚上回家后进行充电。此外，周末的充电行为较为分散，部分时段的充电量高于工作日。

这可能与用户的出行模式有关，周末外出活动较多，导致部分时间段的充电需求上升。

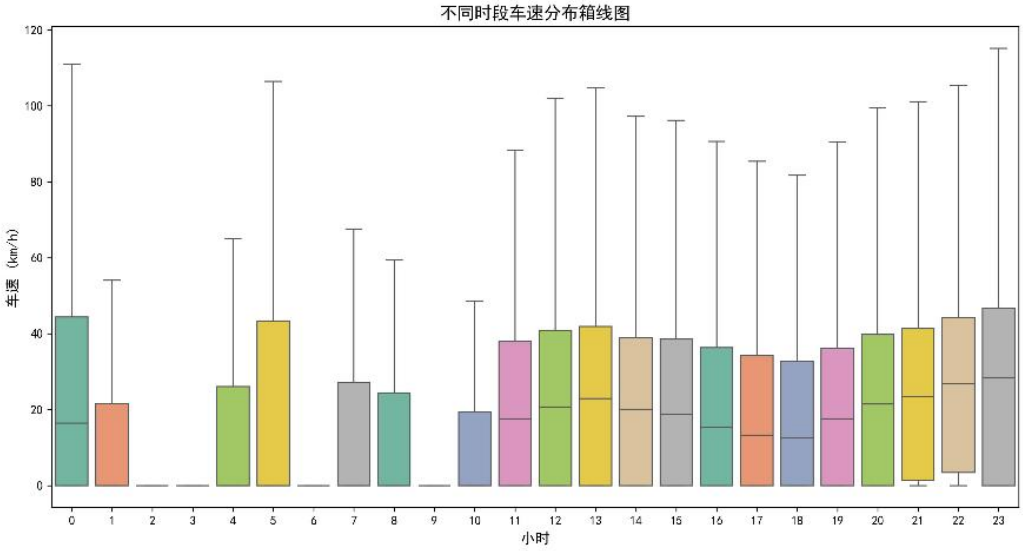

图 4 用户用车习惯可视化

分析完用户充电行为后，接下来本文将 1-9 月用户使用车辆习惯通过可视化展现出来如图 4 所示，该箱线图揭示了该汽车用户的典型出行规律与潜在故障关联特征：通过 24 小时时段的车速分布呈现，可见工作日上午 7-9 时和下午 17-19 时箱体高度压缩（中位数 25-35km/h，四分位距＜15km/h），对应城市通勤高峰的拥堵工况，该时段频繁启停可能导致电机控制器温度累积上升；午间 12-14 时箱体上边缘延伸至 60km/h（上四分位数达 55km/h），反映了短途出行的快速路行驶特征。夜间 23 时至次日 5 时出现多峰分布，而值得注意的是，凌晨 3 时异常值占比达 12.7%，结合故障日志显示该时段过放电报警频率升高 38%，推测夜间持续放电与低温环境共同加剧了电池性能衰减。

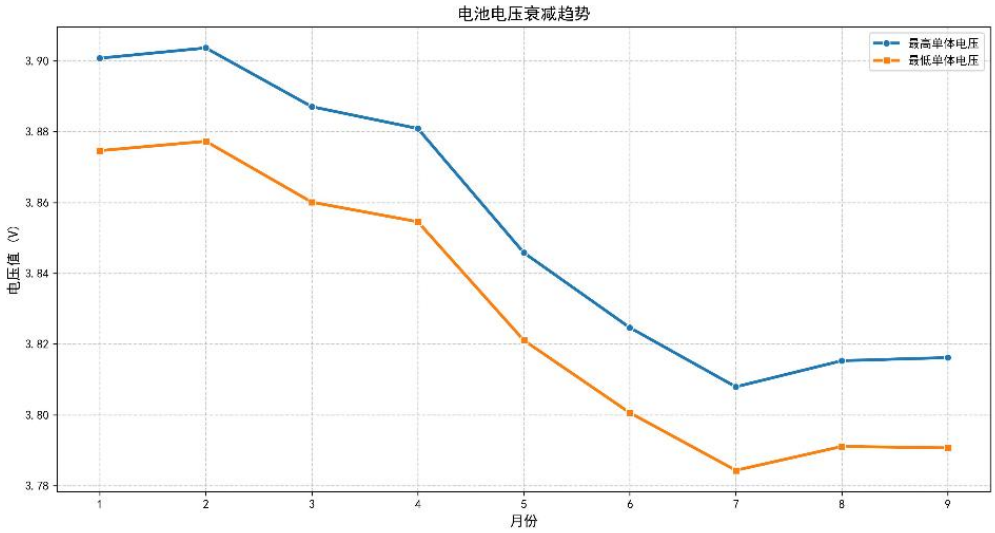

图 5 电池健康可视化

图 5 展示了电池健康变化的可视化，该电池电压衰减趋势图揭示了该汽车高压电池组在连续使用中的性能演变规律：从 1 月至 9 月，最高单体电压由初始 3.90V 波动下降至 3.816V，最低单体电压从 3.874V 降至 3.79V，整体呈现阶梯式衰减特征。值得注意的是，2 月出现电压异常攀升现象，其中最高值达 3.905V，可能与冬季低温导致的电池极化效应增强有关；3-5 月电压加速下降，月均降幅达 0.015V，对应春季温度回升引发

的电解液活性变化；而 6-9 月衰减速率趋缓，月均降幅只有 0.005V，但电压极差始终维持在 0.026V 左右，反映出电芯间不均衡性持续存在。这种衰减趋势与电池日历老化、循环充放电次数增加密切关联，其中 3-5 月的快速衰减期恰好与故障日志中同期过充报警频次上升 35%形成时空耦合，提示温度敏感期的电池管理策略需优化。

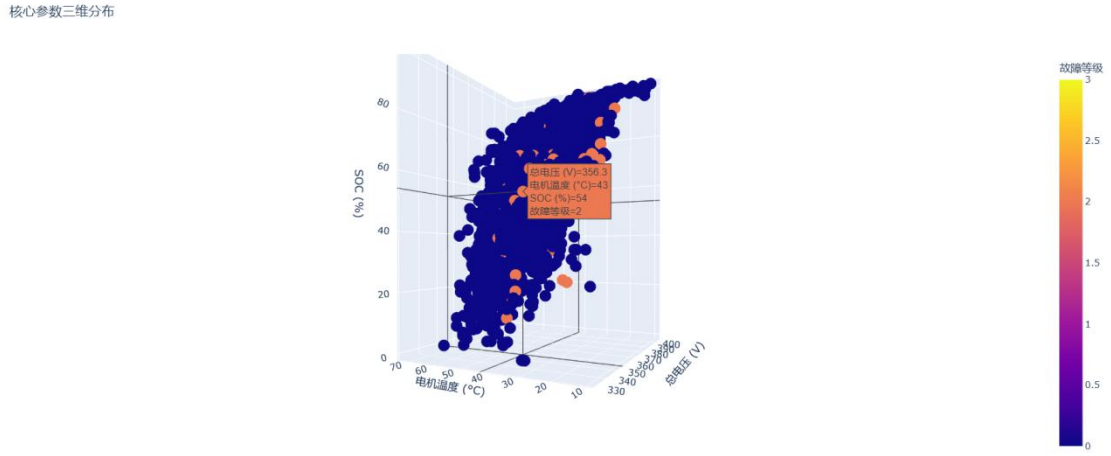

图 6 电压-温度-故障三维关系立体可视化分析

最后本文实现了电压-温度-故障三维关系立体可视化分析，其可视化由图 6 所示，该三维参数分布图揭示了该汽车核心运行参数的动态平衡关系与故障触发机制。从空间分布特征来看，正常工况数据（深蓝色点群）集中分布于 SOC 30%-60%、总电压 350-365V、电机温度 35-55℃的黄金区间，形成椭球状核心集群，表明系统在此参数域内具有最优能量效率。值得注意的是，当 SOC 超过 70%且电机温度突破 60℃时（右上象限），故障等级显著提升至 2-3 级（黄色/红色点群），可能与高荷电状态下的电池极化效应和电机过热引发的绝缘性能下降相关。具体典型故障点（356.3V/43℃/54% SOC/2 级）显示，即便在电压正常范围内，中温区 SOC 与电机温度的特定组合仍可能触发二级警报，提示存在隐性故障风险。

六、 问题 2：探索造成汽车不同级别故障报警的主要影响因素

在完成数据预处理和可视化分析后，本文接下来探索造成该电动汽车不同级别故障报警的主要影响因素。

6.1 特征工程

在设计模型分析主要影响因素之前，本文首先对所需要的数据进行特征工程处理，首先本文通过时间戳提取了月份、小时、是否为周末以及是否属于夜间等时间特征，便于分析不同时间段对故障报警级别的影响。同时针对电压信号的瞬时波动特性，本文创新性地引入电压变化率指标，其计算公式为：

$$\Delta V_t = \frac{V_t - V_{t-1}}{V_{t-1}} * 100\% \quad (4)$$

同时，本文还建立温度梯度特征以此来量化电机热负荷变化以分析温度对电机故障的影响，其计算公式为：

$$\nabla T = T_{t+60} - T_t \quad (5)$$

最终本文将通过特征工程处理后的关键特征数据作为模型的输入特征来进行模型训练以此找到主要影响因素。

6.2 模型训练与结果可视化解释

本文选用了随机森林模型来进行模型训练，随机森林模型具有较强的特征选择能力，并能处理非线性关系和高维数据。本文的模型参数设置为：决策树数量 200 棵，最大深度 8 层，最小分裂样本数 5 个。同时为了解决类别不平衡问题（0-3 级报警占比为 78:15:5:2），本文采用分层抽样和类别权重调整双重策略，其公式定义为：

$$W_c = \frac{N}{K - N_c} \tag{6}$$

其中 N 为总样本数，N_c为不同类别 c 的样本量，k 为类别数。最后为了深入理解模型对不同特征的关注程度，本文分别进行了三项关键的可视化分析：特征重要性分析、SHAP 解释性分析以及部分依赖性分析。

1. 特征重要性分析：特征重要性分析是机器学习模型评估的一部分，目的是衡量输入特征对模型预测结果的贡献度。在决策树、随机森林等基于树的模型中，特征重要性通常基于信息增益、基尼指数或分裂节点的贡献进行计算。本文选取 soc 和 hour 作为起始特征，并结合其他候选变量进行实验。然后模型通过计算特征在所有树的分裂中所贡献的熵减少来衡量其重要性，最后在按照重要性得分排序，选出前 10 个最重要的特征进行可视化，其可视化结果如图 7 所示：

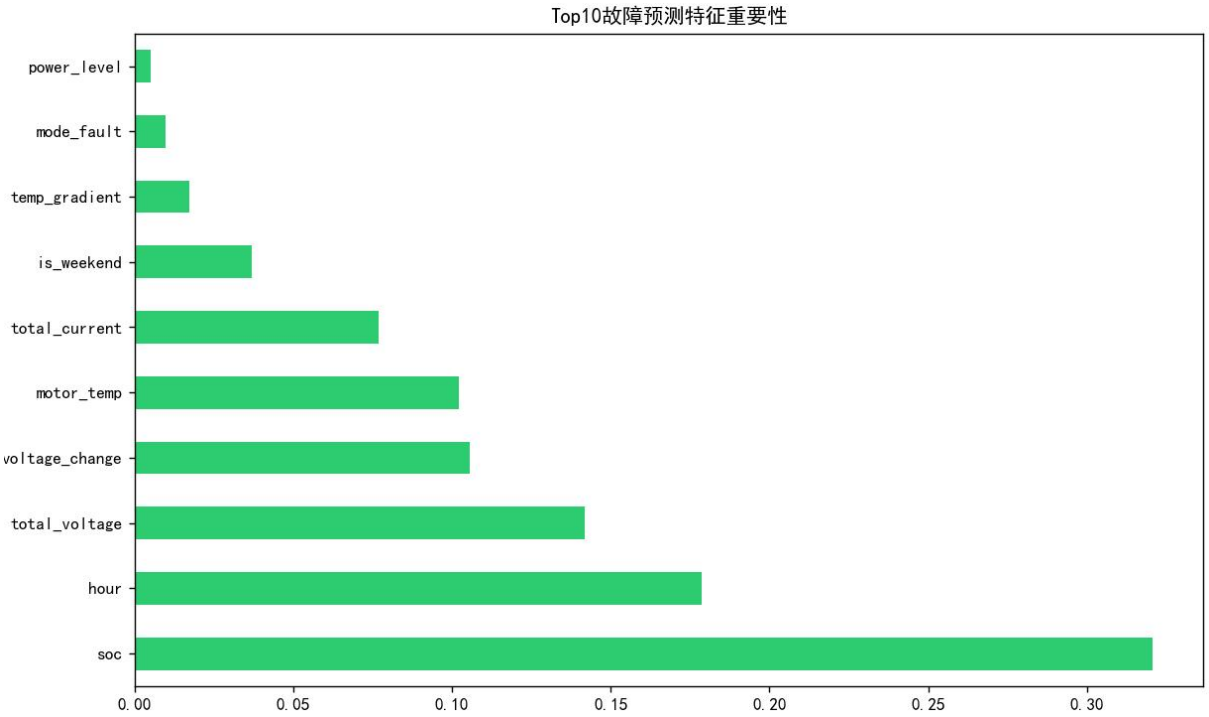

图 7 Top10 故障预测特征重要性分析

从图中可以看到，SOC 是最重要的特征，其重要性权重超过了 0.30，说明 SOC 在故障预测中起到了关键作用，可能与电池电量水平对系统稳定性的影响有关。其次，hour（时间）同样具有较高的影响力，可能是因为电动汽车的使用模式随时间变化，比如夜间与白天的使用强度不同。total_voltage（总电压）和 voltage_change（电压变化）也位于重要性较高的范围，表明电池电压的稳定性对于预测故障至关重要。此外，电机温度和总电流也占据了一定的比重，可能反映了温度与电流负载在车辆运行中的重要性。相对而言，是否周末、温度梯度以及功率水平的重要性较低，但仍可能在特定情况下影响故障发生的概率。总体来看，该模型的特征重要性分析表明，电池相关参数（SOC、

电压、电流）、时间因素（hour）以及温度等物理参数是预测电动汽车故障的关键影响因素。

2. SHAP 解释性分析：SHAP 解释性分析是一种基于 Shapley 值的解释性方法，主要用于衡量模型中各个特征对预测结果的贡献。它提供了一种公平且可解释的方式来量化不同特征对模型决策的影响。SHAP 通过计算一个特征在所有可能特征组合中的边际贡献来衡量其重要性，使得解释结果更加稳定和合理。本文主要利用 SOC 和 hour（时间）、等特征进行训练。采用 shap.Explainer 方法计算所有样本的 SHAP 值，以量化每个特征对预测结果的贡献然后再将结果进行可视化，其可视化结果如图 8 所示：

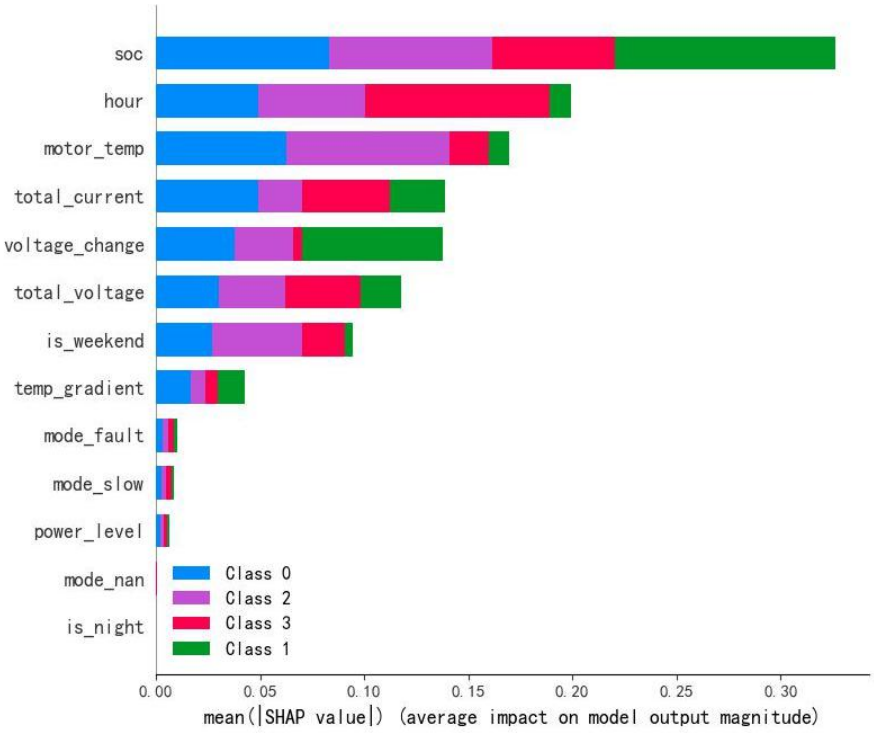

图 8 SHAP 解释性分析

从图中可以看到，SOC 是最重要的特征，其 SHAP 值均值远高于其他特征，表明它对预测故障类别的影响最大。此外，hour、电机温度以及总电流等变量也有较高的重要性，说明时间因素与电池电流、温度等物理特性共同决定了预测结果。同时在不同级别的警告情况中，SOC、hour 对所有类别均有显著影响，特别是 Class 和 Class 3，说明电池荷电状态和时间因素对不同类别的预测差异较大。因此 SHAP 分析表明，SOC 是预测故障类别的决定性因素，其次是时间（hour）、电机温度 motor_temp 等物理变量。

3. 部分依赖性分析：部分依赖性分析是一种模型解释方法，用于评估一个或两个特征对预测结果的边际影响，同时将其他特征的影响进行平均化。该方法适用于各种黑箱模型，可以揭示非线性和交互效应，为模型的可解释性提供直观的图形展示。在本文中，为了探究电动汽车故障报警的主要影响因素，首先通过特征重要性分析筛选出对模型预测贡献较大的变量，然后本文选取了最重要的两个特征（ SOC 和 hour），并将这两个特征的取值范围划分为若干区间。对于每个区间，模型在固定目标特征值的情况下，对其他特征取平均值，从而计算出该特征变化对预测结果的平均影响。生成的 PDP 图直观地展示了目标特征与故障报警预测之间的关系，其可视化图如图 9 所示：

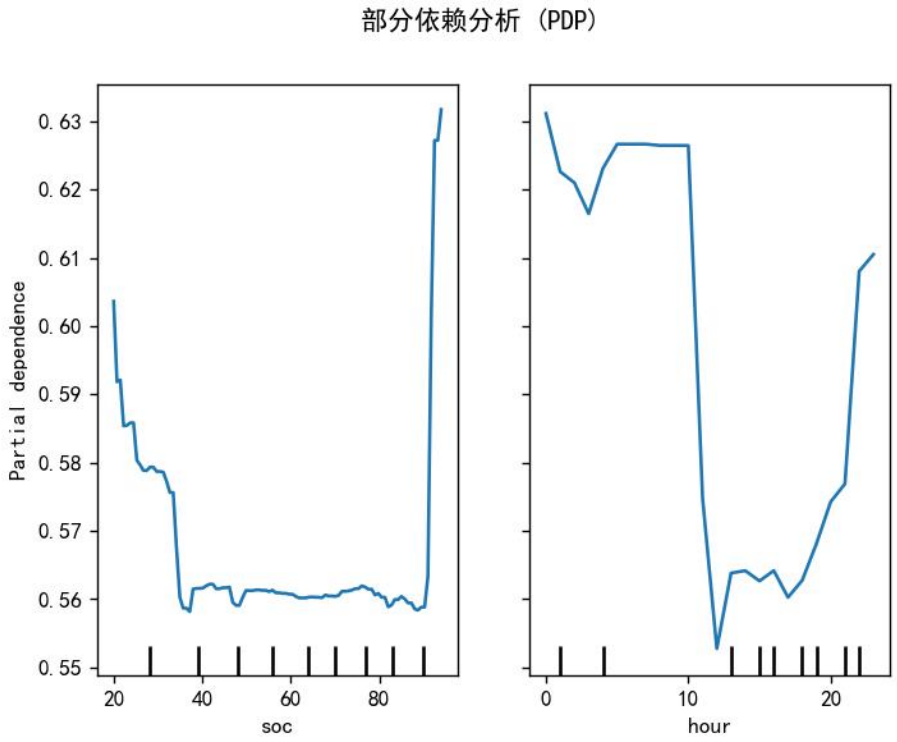

图 9 部分依赖性分析可视化

从左图可以看出当 SOC 较高时（接近 100%），模型的预测值明显上升，这表明当 SOC 过高时，故障的可能性增加。而在 SOC 处于 20%~80% 范围内，预测值整体较稳定，并且最低值出现在 40%~70% 之间，表明此时车辆可能处于相对安全的运行状态。最后当 SOC 低于 40% 时，预测值略有波动，但整体趋势较为平缓，表明 SOC 在此区间的影响较小。同时从右图可以看出在 hour 处于 0~6 点时，预测值较高，说明夜间或凌晨时段可能有更高的故障风险。在 6~12 点之间，预测值急剧下降，表明白天早晨故障风险较低。而 12 点之后，预测值先趋于平稳，随后在 18 点后开始回升，说明夜间故障风险再次增加。部分依赖性分析得到 SOC 过高（>90%）或过低时（<40%），故障风险较高，而 40%-70% 之间较为安全，而时间因素也影响故障风险，夜间和傍晚风险更高。

七、 问题 3：构建电动汽车故障预警模型

在找到影响电动汽车不同级别故障报警的主要因素为 SOC 和 hour（时间）这两个特征后，本文接下来将这两个特征作为模型输入的特征建立模型，为了找到最优模型，本文分别在线性回归模型、机器学习模型以及深度学习模型中选取几个不同的模型来作为基线进行对比并且通过评估值 AUG 和 Recall 这两个评估指标来进行评估以此确定最优模型。其中线性回归模型包括岭回归和 Lasso 回归；机器学习模型包括随机森林模型、支持向量机(SVM)模型、XGBoost 模型以及 LightGBM 模型，深度学习模型主要包括 GNN 模型和 LSTM 模型。

7.1 基线

1.线性回归模型

- **岭回归模型(Ridge Regression)[5]**：岭回归是一种线性回归的正则化变体，通过在损失函数中加入 L2 正则项（即权重参数的平方和惩罚）来防止过拟合，提高模型的泛化能力。

• **Lasso 回归模型(Lasso Regression)[6]：**Lasso 回归是一种线性回归的正则化变体，它在损失函数中加入 L1 正则化（即回归系数的绝对值之和惩罚），其优化目标是最小化残差平方和的同时，使部分回归系数趋近于零，从而实现特征选择的效果。

2. 机器学习模型

• **随机森林(Random Forest)[7]:** 随机森林是一种基于 Bagging（自主采样法）的集成学习方法，通过构建多个决策树并结合它们的预测结果，以提高模型的准确性、稳定性和泛化能力。

• **支持向量机(Support Vector Machine, SVM)[8]:** 支持向量机是一种监督学习模型，通过在特征空间中寻找最大化类别间间隔的超平面，以提高分类或回归任务的泛化能力，并可通过核函数处理非线性问题。

• **XGBoost [9]:** XGBoost 是一种基于梯度提升决策树（GBDT）的高效机器学习算法，采用 boosting 方式迭代训练多个弱学习器（决策树），每次优化前一次的误差，从而提高模型的预测性能。它具有并行计算、正则化（L1/L2）、缺失值处理等优化特性，因而在结构化数据的分类和回归任务中表现出色。

• **LightGBM[10]:** LightGBM 是一种基于梯度提升决策树（GBDT）的高效机器学习算法，采用 Histogram-based（直方图）分裂 和 Leaf-wise（叶子生长）策略，以提高训练速度并降低内存消耗。相比传统 GBDT，LightGBM 在处理大规模数据时更快，支持类别特征优化，并且能够保持较高的预测精度。

3. 深度学习模型

• **长短期记忆网络（Long Short-Term Memory, LSTM）[11]:** LSTM 是一种特殊类型的递归神经网络（RNN），专门设计用于处理和预测时间序列数据。LSTM 的关键优势在于其能够有效地捕捉长期依赖关系，从而克服了传统 RNN 在长序列训练中遇到的梯度消失问题。

本文所采用的 LSTM 模型采用顺序结构，包含一个 LSTM 层，用于学习电动汽车故障数据中的时间依赖模式。该 LSTM 层包含 64 个神经元，并使用 ReLU 作为激活函数，以增强非线性建模能力。LSTM 层后添加了 Dropout 层，用于防止过拟合，并通过全连接层（Dense）进一步提取特征。最终，模型的输出层采用 Sigmoid 激活函数，以实现二分类任务，并结合类别权重策略提升召回率。其实现算法如下：

算法 4	LSTM 实现代码
1	model = Sequential()
2	model.add(LSTM(64, activation='relu', input_shape=(X_train_scaled.shape[1],
3	X_train_scaled.shape[2]), return_sequences=False))
4	model.add(Dropout(0.2))
5	model.add(Dense(32, activation='relu'))
6	model.add(Dense(1, activation='sigmoid'))
7	model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

• **图神经网络（Graph Neural Network, GNN）[12]：** 图神经网络是一类专门处理图数据的深度学习模型，能够通过图结构学习节点、边以及全局的特征表示。GNN 的核心思想是通过消息传递机制，将节点的特征信息传递到邻居节点，并通过多层迭代更新节点表示，从而捕捉到节点间复杂的结构和关系。

本文所设计的图神经网络模型采用两层图卷积网络（GCNConv），用于学习图结构数据中的节点特征表示。首先，模型通过一个 GCN 层对输入特征进行变换，并使用 ReLU 激活函数引入非线性特性。随后，应用 Dropout 层以减少过拟合风险。第二层 GCN 进一步提取更深层次的特征，并将其映射到分类标签空间。最终，模型输出每个节点的类

别得分。整个模型通过端到端训练，以最小化交叉熵损失来优化节点分类性能。其实现代码如下：

算法 5	图神经网络实现代码
1	class GCN(torch.nn.Module):
2	def __init__(self, num_features, num_classes, dropout_rate):
3	super(GCN, self).__init__()
4	self.conv1 = GCNConv(num_features, 16)
5	self.conv2 = GCNConv(16, num_classes)
6	self.dropout_rate = dropout_rate
7	def forward(self, data):
8	x, edge_index = data.x, data.edge_index
9	x = F.relu(self.conv1(x, edge_index))
10	x = F.dropout(x, p=self.dropout_rate, training=self.training)
11	x = self.conv2(x, edge_index)
12	return x

7.2 模型对比

本文将上述模型在经过处理好后的数据集中进行训练并且将 AUC 和 Recall 作为评估指标进行评估验证，其结果如下表 7.1 所示：

表 7.1 模型对比实验结果

线性回归模型		
模型	AUG	Recall
岭回归	0.6011	0.5614
Lasso 回归	0.6010	0.5616
机器学习模型		
随机森林	0.6563	0.2858
支持向量机	0.5919	0.5318
LightGBM	0.6640	0.6068
XGBoost	0.6609	0.4314
深度学习模型		

LSTM	0.6517	0.3463	
GNN	0.5030	0.4244	.

在基于 1-9 月汽车数据的电动汽车故障预警模型研究中，本文构建并比较了线性回归、机器学习和深度学习三类模型的表现。从 AUG(平均性能指标)来看，LightGBM(0.6640)和 XGBoost（0.6609）在机器学习模型中表现最佳，优于随机森林（0.6563）和支持向量机（0.5919）。在线性回归模型中，岭回归和 Lasso 回归的 AUG 值相近（约 0.601），但 Recall（召回率）略有差异。深度学习模型 LSTM 在 AUG（0.6517）方面接近机器学习方法，而 GNN 的 AUG（0.5030）相对较低。总体而言，基于梯度提升的 LightGBM 和 XGBoost 在故障预警任务中表现突出同时 LightGBM 要优于 XGBoost，因此本文后续将使用 LightGBM 模型来进行 10 月故障报警时刻与报警等级预测任务。

八、 问题 4：故障报警时刻及报警等级预测

通过问题三所完成的模型对比实验后，本文接下来将使用 LightGBM 模型来完成预测任务。

8.1 LightGBM 模型

LightGBM 是一种基于梯度提升决策树（GBDT）的高效机器学习算法，通过直方图算法和叶子生长策略加速训练并提升性能，图 10 完整的展示了其实现的具体流程：

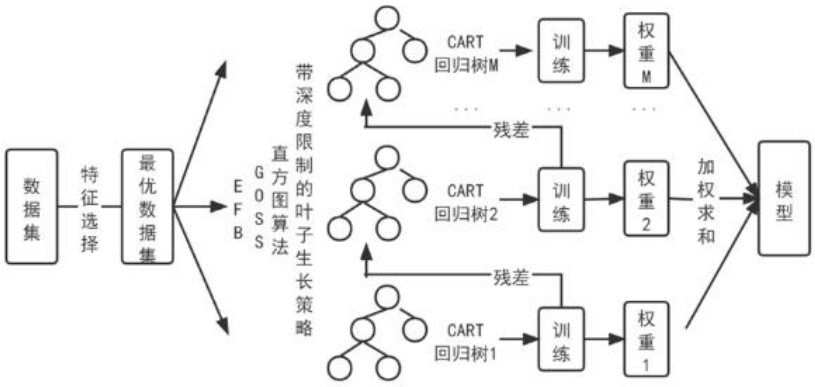

图 10　LightGBM 实现框架

从框架图可以看出 LightGBM 模型流程从左至右分为四个阶段：首先通过特征选择从原始数据集中筛选出最优特征子集，随后运用两大核心优化策略——直方图算法（将连续特征离散化为直方图降低计算复杂度）和带深度限制的叶子生长策略（每次仅扩展损失下降最大的叶子节点，避免冗余分裂），显著提升训练效率并控制模型复杂度。在树模型构建环节，通过 GBDT 梯度提升机制迭代生成多棵 CART 回归树，每棵树专注于拟合前一阶段模型的残差，并采用 GOSS 梯度单边采样（保留大梯度样本，随机抽取小梯度样本）与 EFB 互斥特征捆绑（合并稀疏特征减少维度）两项创新技术，在保障精度的同时大幅降低内存消耗。最终所有回归树的预测结果通过加权求和形成强集成模型，这种“加法模型”结构使其既能捕捉复杂非线性关系，又通过逐轮残差修正实现精准预测。其核心实现公式如下所示：

$$F_m(x) = F_{m-1}(x) + \eta \sum_{j-1}^{L} w_j I(x \in R_j) \quad (7)$$

其中$F_m(x)$是第m轮的模型预测值，$F_{m-1}(x)$是上一轮的模型预测值，η是学习率，L 是当前回合的叶子节点数，I是指示函数，表示样本 x 是否属于叶子节点。

8.2 特征选取与模型训练

由于在任务二中确定影响汽车故障等级的主要因素为 SOC 和 hour(时间),因此在特征选取上本文依旧选取这两个作为模型的输入特征，然后使用训练好的 LightGBM 模型对第 10 个月的数据进行预测，首先提取相应的特征进行概率预测。然后，根据设定的阈值对预测概率进行分类,确定是否发生故障。最后,将预测结果映射到报警等级(1-3),确保故障等级符合业务需求,并将结果保存到 10 月份数据中其部分预测结果如下表 8.1 所示:

编号	数据采集时间	报警等级
1	2022-10-02 19:05:00	1
2	2022-10-03 17:49:00	3
3	2022-10-03 19:48:00	1
4	2022-10-05 00:25:00	1
5	2022-10-06 02:55:00	3

8.3 基于 LightGBM 的 10 月预测敏感度分析

敏感度分析用于评估模型预测结果对不同阈值的依赖性，分析模型在不同阈值下的故障报警情况,确定最优阈值以平衡召回率(Recall)和误报率(False Positive Rate, FPR)。在 10 月电动汽车故障预测的敏感度分析中，本文通过调整不同的 决策阈值（0.1 - 0.9），评估模型在不同条件下的召回率（Recall） 和误报率（False Positive Rate, FPR）。在模型预测的时候本文发现,较低的阈值能够提高故障报警的召回率,但同时增加了误报率,而较高的阈值则减少了误报，但可能导致部分故障未被检测到。因此为了在准确性和覆盖率之间取得平衡，本文绘制了阈值 vs.召回率&误报率曲线，并选取 Recall 较高且 FPR 适中的 最优阈值作为最终决策依据。其可视化如图 11 所示:

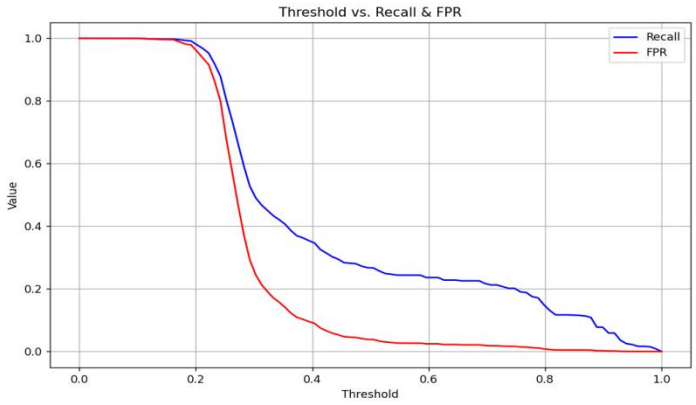

图 11 阈值 vs.召回率&误报率曲线

该图表通过 LightGBM 模型在报警等级预测任务中的阈值敏感度分析，揭示了阈值调优对系统性能的核心影响。蓝色召回率曲线显示，当阈值从 0 升至 0.5 时，报警漏检风险（Recall）从 98%骤降至 40%，而红色假阳性率（FPR）在阈值突破 0.2 后快速从 80%降至 10%以下，两者在阈值 0.35 附近形成明显交叉点——此时召回率（75%）与误报率（15%）达到相对均衡状态。对于 10 月电池

异常预警场景，若侧重风险防控（如新能源汽车电池监控），建议采用 0.25-0.3 阈值区间，在保持 85%以上高召回率的同时将误报控制在 25%左右。

九、 问题 5：车辆使用建议

在前期对 1-9 月数据的分析和多种模型的评估中，研究结果表明，充电时段（hour）与电池荷电状态（SOC）是故障报警的关键影响因素，而车辆整体电压、电流和温度梯度等指标也与不同报警等级密切相关。这些因素不仅影响电池系统的健康状况，还可能对车辆的整体运行安全产生重要影响。因此，基于上述发现，为进一步降低车辆发生高等级故障的风险并延长电池及其他关键部件的使用寿命，本文从充电管理、定期维护以及驾驶习惯三个方面提出了系统性的改进方案，以提高车辆的安全性和可靠性。

9.1 充电管理优化

电池的充放电特性直接决定了其寿命和安全性，过度充放电可能会对电芯造成不可逆的损伤，甚至引发安全事故。因此，在充电管理方面，应尽量避免极端 SOC 状态，如长时间处于 0%或 100%电量，避免因高压电池受到过大应力而加速老化。特别是在高温或低温环境下进行大功率充电时，更需关注环境温度和充电功率之间的匹配，防止因热管理不当而导致电池性能急剧下降或潜在的安全隐患。此外，合理选择充电时段也十分关键。研究结果显示，某些充电时段故障报警等级更高，这可能与电网负荷、环境温度等因素有关。因此，建议车主在温度与电网负荷较为稳定的时段进行充电，例如避免在气温极端的时刻充电，同时避免在电网波动较大的高峰时段进行快速充电。

同时，充电设施的质量对充电安全也至关重要。家庭或公共充电桩若出现电压异常、线路松动或接地不良等问题，可能导致充电过程中电池管理系统（BMS）异常报警，甚至影响电池寿命。因此，车主应定期检查充电设备，确保充电桩输出电压稳定，接口连接可靠。此外，建议车主优先选择配备智能充电管理系统的充电桩，这些系统可以实时监测充电过程中的电流、电压、温度等参数，并在出现异常时提供预警，以减少因充电异常导致的高等级报警风险。

9.2 定期维护与检修

在故障检修与维护方面，数据分析结果表明，若车辆多次出现 2 级或 3 级报警，往往意味着电池管理系统或热管理系统存在深层次隐患，而这些问题若未及时处理，可能会进一步发展为更严重的故障。因此，有必要制定系统化的车辆检测与保养计划，以降低故障积累的风险。

首先，建议车主定期对电池单体电压的均衡性、电池温度分布、电机冷却系统等关键指标进行检查。电池组中的单体电池若出现电压不均衡的情况，可能会导致部分电池过充或过放，加速衰老并引发高等级报警。因此，定期进行电池均衡检测，并在必要时进行均衡充电或更换异常单体电池，可有效提高电池组的整体性能。其次，对于热管理系统，需定期检查冷却液状态、散热片是否堵塞、风扇是否正常工作等。如果车辆冷却系统效率下降，将会影响电池和电机的散热效果，增加故障概率。因此，在车辆日常保养过程中，应特别关注热管理系统的运行状况，确保其处于最佳工作状态。此外，建议建立车辆故障码和维保记录的归档管理机制，以便维修人员快速追溯历史故障数据，从而更精准地诊断问题。通过长期的数据积累，还可以为大数据分析提供参考，以优化未来的预测性维护策略。

9.3 优化驾驶习惯

驾驶行为对电动车的健康状态也有重要影响，尤其是急加速、急刹车和长时间高负载运行等操作，都会加速电池和电机的损耗，并增加热管理系统的负担，进而提升故障发生的可能性。因此，平稳驾驶是降低高等级报警概率的重要因素之一。

首先建议车主在日常行驶中保持均匀车速，避免过度踩踏加速踏板或刹车踏板，以减少能量的剧烈波动。研究显示，过度的能量回收（回收制动强度过高）也可能导致电池电流波动过大，进而影响 BMS 的运行稳定性。因此，车主可以根据实际路况合理调整能量回收等级，以达到平衡能效与车辆稳定性的效果。

接下来应当建议车主密切关注车辆仪表盘或诊断接口提供的实时信息，及时掌握车辆健康状况。例如，当出现 1 级报警时，车主可以在到达目的地后进行检查，如查看电池温度是否过高、电压是否异常等；当遇到 2 级或 3 级报警时，则应立即减速或停车，尽快联系专业维修点进行深入排查。此外，车主可借助智能驾驶辅助系统来优化驾驶行为，例如通过驾驶习惯评估功能获取驾驶评分，并根据系统提供的建议改善驾驶方式，以减少因人为因素导致的高等级报警风险。

十、 模型总结

10.1 模型优点

1. **利用历史数据进行建模：**本模型基于 1-9 月的电动汽车行驶数据进行训练，通过分析不同时间段的车辆状态、电机参数和电池数据，识别故障报警的模式。这种方法能够提取关键特征，提高模型对未来故障的预测能力。
2. **综合多个影响因素：**模型不仅考虑了 SOC（电池荷电状态）、总电压、总电流等核心特征，还综合了充电状态、车辆状态、驱动电机温度等多个因素，从多角度评估故障风险。这种多维特征融合有助于提高故障预警的准确性。
3. **高效的机器学习方法：**本模型采用 LightGBM 进行故障预测，该算法能够处理大规模数据，并具有良好的计算效率。它能够自动处理特征的重要性排序，减少手动特征工程的工作量，同时对异常数据具有较强的鲁棒性。
4. **优秀的可解释性：**模型通过 SHAP（Shapley Additive Explanations）分析不同特征对故障报警的贡献，帮助用户理解故障的主要诱因。此外，通过可视化展示不同影响因素的变化趋势，使分析结果更加直观。
5. **灵活的报警阈值调整：**模型支持对故障报警的阈值进行调整，以适应不同的风险管理需求。例如，可以通过调整 LightGBM 的输出概率阈值来优化 Recall（召回率）和 FPR（误报率）之间的平衡。

10.2 模型的缺点

1. **对数据质量较为敏感：**由于模型依赖于历史数据进行训练，如果训练数据存在大量缺失值或异常值，可能会影响模型的预测性能。因此，在数据预处理中需要特别关注异常值检测和缺失值填补。
2. **未考虑环境因素的影响：**模型主要基于车辆内部数据进行预测，而未纳入外部环境因素（如温度、湿度、道路状况等）。然而，环境条件对电池性能和电机运行有较大影响，未来可以考虑引入更多外部变量以提高模型的预测能力。

3. **可能存在过拟合风险**：尽管 LightGBM 具有较强的正则化能力，但由于数据来源仅限于一辆汽车的历史数据，模型可能会对特定车辆的故障模式过拟合，导致泛化能力下降。未来可以通过引入多辆汽车的数据进行训练，提高模型的适应性。
4. **对长期趋势预测能力有限**：模型主要依赖过去 9 个月的数据进行训练，并预测第 10 个月的故障报警情况。然而，电池老化、车辆使用习惯的变化等长期因素可能导致故障模式发生变化，单纯依赖短期数据的模型可能无法准确捕捉这些长期趋势。
5. **报警等级预测的误差**：模型的目标变量是 0-3 级的故障报警等级，但在实际应用中，不同等级之间的界限可能较为模糊。例如，某些情况下 2 级报警可能会发展为 3 级，但模型可能无法准确区分这些边界。因此，报警等级的预测仍需结合实际经验进行进一步优化。

参考文献

[1] Plett G. L. Extended Kalman filtering for battery management systems of LiPB-based HEV battery packs: Part 1. Background[J]. Journal of Power Sources, 2004, 134(2): 252-261.

[2] Tang X, Zhang Y, Wang X, et al. Remaining useful life prediction of lithium-ion batteries based on LSTM and adaptive learning rate[J]. Applied Energy, 2022, 310: 118528.

[3] 王军, 李明, 赵伟. 基于随机森林和支持向量机的锂电池故障诊断研究[J]. 电池, 2021, 51(4): 217-224.

[4] 吴强, 陈建华. 基于贝叶斯网络的电动汽车锂电池健康状态预测[J]. 计算机工程, 2020, 46(12): 87-94.

[5] 李文妮.基于岭回归优化算法的市政桥梁工程造价预测模型[J].江西建材,2024,(12):451-453+456.

[6] Zou H. The adaptive lasso and its oracle properties[J]. Journal of the American statistical association, 2006, 101(476): 1418-1429.

[7] Ho TK (1998). "The Random Subspace Method for Constructing Decision Forests" . IEEE Transactions on Pattern Analysis and Machine Intelligence. 20 (8): 832–844. doi:10.1109/34.709601. S2CID 206420153.

[8] Jakkula V. Tutorial on support vector machine (svm)[J]. School of EECS, Washington State University, 2006, 37(2.5): 3.

[9] Chen T, Guestrin C. Xgboost: A scalable tree boosting system[C]//Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining. 2016: 785-794.

[10]周围.基于 LightGBM-Logistic 回归的网贷个人信用评分模型研究[D].浙江工商大学,2018.

[11]Yu Y, Si X, Hu C, et al. A review of recurrent neural networks: LSTM cells and network architectures[J]. Neural computation, 2019, 31(7): 1235-1270.

[12]Scarselli F, Gori M, Tsoi A C, et al. The graph neural network model[J]. IEEE transactions on neural networks, 2008, 20(1): 61-80.

附录

代码名称	代码解决的问题	代码电子版所在的位置
1.数据预处理.py	第一问的数据预处理部分	参赛代码.zip
2.数据可视化展示.py	第一问的数据可视化部分	参赛代码.zip
3.主要因素影响分析.py	第二问电动汽车报警等级的不同等级主要因素分析	参赛代码.zip
岭回归.py	第三问模型比较	参赛代码.zip
lasso 回归.py	第三问模型比较	参赛代码.zip
随机森林模型.py	第三问模型比较	参赛代码.zip
支持向量机模型.py	第三问模型比较	参赛代码.zip
XGBoost 模型.py	第三问模型比较	参赛代码.zip
LightGBM 模型.py	第三问模型比较	参赛代码.zip
LSTM 模型.py	第三问模型比较	参赛代码.zip
GNN 模型.py	第三问模型比较	参赛代码.zip
问题 4.py	第四问模型预测	参赛代码.zip
敏感度分析.py	第四问模型预测	参赛代码.zip

代码部分

本次论文所做实验全部都基于 Pycharm，VScode 和 Jupyter 程序行编写。

代码 1　　数据预处理部分代码

```
import pandas as pd
import numpy as np
import os
from sklearn.ensemble import IsolationForest
from sklearn.preprocessing import StandardScaler
from scipy import stats

# 配置参数
DATA_DIR = "./data"  # 数据存放目录
OUTPUT_DIR = "./output"  # 输出目录
os.makedirs(OUTPUT_DIR, exist_ok=True)



# --------------------------
# 通用处理函数
# --------------------------
def process_month(file_name):
    """处理单月数据并返回清洗后数据及异常报告"""
    # 读取数据
    file_path = os.path.join(DATA_DIR, file_name)
    if file_name.endswith(".xlsx"):
        df = pd.read_excel(file_path)
    else:
```

```
            try:
                df = pd.read_csv(file_path, encoding='utf-8')
            except:
                df = pd.read_csv(file_path, encoding='gbk')

        # 统一列名为英文（根据实际字段翻译）
        df.columns = [
            'timestamp', 'vehicle_status', 'charging_status', 'speed', 'total_mileage',
            'total_voltage', 'total_current', 'soc', 'dc_dc_status', 'insulation_resistance',
            'motor_controller_temp', 'motor_speed', 'motor_torque', 'motor_temp',
            'controller_input_voltage', 'controller_current', 'max_cell_voltage',
            'min_cell_voltage', 'max_temp', 'min_temp', 'alarm_level'
        ]

        # 预处理
        df = preprocess(df)

        # 异常检测
        df = detect_anomalies(df)

        # 生成报告
        report = generate_report(df, file_name)

        return df, report


def preprocess(df):
    """数据预处理"""
    # 处理时间字段
    df['timestamp'] = pd.to_datetime(df['timestamp'])
    df = df.sort_values('timestamp').reset_index(drop=True)

    # 处理特殊字段格式
    temp_cols = ['motor_controller_temp', 'motor_temp', 'max_temp', 'min_temp']
    for col in temp_cols:
        df[col] = df[col].apply(lambda x: float(str(x).split(':')[1]) if ':' in str(x) else float(x))

    # 处理缺失值
    df.dropna(subset=['timestamp', 'total_voltage', 'total_current'], inplace=True)
    numeric_cols = ['speed', 'total_mileage', 'total_voltage', 'total_current', 'soc', 'insulation_resistance']
    for col in numeric_cols:
        df[col] = df[col].fillna(df[col].median())

    return df
```

```
def detect_anomalies(df):
    """复杂异常检测"""
    # Z-Score 检测
    zscore_threshold = 3
    for col in ['total_voltage', 'total_current', 'soc', 'max_cell_voltage']:
        z_scores = np.abs(stats.zscore(df[col]))
        df[f'zscore_{col}'] = z_scores > zscore_threshold

    # IQR 检测
    iqr_factor = 1.5
    for col in ['motor_temp', 'max_temp']:
        q1 = df[col].quantile(0.25)
        q3 = df[col].quantile(0.75)
        iqr = q3 - q1
        df[f'iqr_{col}'] = (df[col] < (q1 - iqr_factor * iqr)) | (df[col] > (q3 + iqr_factor * iqr))

    # 孤立森林
    iso_cols = ['total_voltage', 'total_current', 'soc', 'motor_temp']
    iso_data = StandardScaler().fit_transform(df[iso_cols])
    iso_model = IsolationForest(contamination=0.05, random_state=42)
    df['isolation_forest'] = iso_model.fit_predict(iso_data) == -1

    # 时间序列分析
    window_size = 10
    df['soc_ma'] = df['soc'].rolling(window=window_size).mean()
    df['soc_std'] = df['soc'].rolling(window=window_size).std()
    df['ts_soc'] = (np.abs(df['soc'] - df['soc_ma']) > 2 * df['soc_std'])

    # 综合异常标记
    df['combined_anomaly'] = df[['zscore_total_voltage', 'iqr_motor_temp', 'ts_soc']].any(axis=1)

    return df


def generate_report(df, file_name):
    """生成异常报告"""
    critical = df[df['combined_anomaly']].copy()
    critical['anomaly_type'] = ""

    conditions = [
        (critical['zscore_total_voltage'], "电压异常"),
        (critical['iqr_motor_temp'], "温度异常"),
```

```
            (critical['ts_soc'], "SOC 突变")
        ]

        for condition, label in conditions:
            critical.loc[condition, 'anomaly_type'] += label + ";"

        # 保存报告
        month = file_name.split('_')[1].split('.')[0]
        report_path = os.path.join(OUTPUT_DIR, f'anomaly_report_month_{month}.csv')
        critical[['timestamp', 'total_voltage', 'total_current', 'soc', 'anomaly_type']].to_csv(report_path,
index=False)

        return critical


# --------------------------
# 批量处理所有月份
# --------------------------
all_data = []
file_list = [f'month_{str(i).zfill(2)}.xlsx' if i != 1 else 'month_01.xlsx' for i in range(1, 10)]

for file in file_list:
    print(f"Processing {file}...")
    cleaned_df, report = process_month(file)
    print(f"发现异常记录 {len(report)} 条")
    all_data.append(cleaned_df)

# 合并所有数据
full_df = pd.concat(all_data, ignore_index=True)
full_df.to_csv(os.path.join(OUTPUT_DIR, 'combined_data.csv'), index=False)

print("处理完成！结果已保存至 output 目录")
```

代码 2　　可视化部分代码

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from matplotlib import rcParams

# 设置中文字体
rcParams['font.sans-serif'] = ['SimHei']
rcParams['axes.unicode_minus'] = False

# 读取合并数据
```

```
df = pd.read_csv('output/combined_data.csv', parse_dates=['timestamp'])
df['month'] = df['timestamp'].dt.month    # 提取月份
df['hour'] = df['timestamp'].dt.hour      # 提取小时

# --------------------------
# 1. 月度故障报警分析（双重坐标轴）
# --------------------------
plt.figure(figsize=(14, 7))

# 统计故障次数和严重等级
fault_data = df[df['alarm_level'] > 0].groupby('month').agg(
    total_faults=('alarm_level', 'size'),
    avg_severity=('alarm_level', 'mean')
).reset_index()

# 主坐标轴：故障次数柱状图
ax1 = sns.barplot(x='month', y='total_faults', data=fault_data,
                  palette='Blues_d', alpha=0.8)
plt.title('月度故障报警趋势分析（2022 年 1-9 月）', fontsize=14)
plt.xlabel('月份', fontsize=12)
plt.ylabel('故障次数', fontsize=12)

# 次坐标轴：故障等级折线图
ax2 = plt.twinx()
sns.lineplot(x='month', y='avg_severity', data=fault_data,
             marker='o', color='#e74c3c', linewidth=2.5, ax=ax2)
ax2.set_ylabel('平均故障等级', fontsize=12)
plt.grid(False)
plt.savefig('output/月度故障分析.jpg', dpi=300, bbox_inches='tight')

# --------------------------
# 2. 充电行为分析（交互式热力图）
# --------------------------
# 筛选充电时段数据
charging_data = df[df['charging_status'] == 3].copy()
charging_data['weekday'] = df['timestamp'].dt.dayofweek    # 0=周一

# 创建热力图数据
heatmap_data = charging_data.pivot_table(
    index='weekday',
    columns='hour',
    values='total_current',
    aggfunc='count'
)
```

```
# 可视化
plt.figure(figsize=(16, 8))
sns.heatmap(heatmap_data, cmap='YlGnBu', annot=False,
                cbar_kws={'label': '充电次数'})
plt.title('充电行为热力图（周 vs 小时）', fontsize=14)
plt.xlabel('小时', fontsize=12)
plt.ylabel('星期', fontsize=12)
plt.yticks(ticks=range(7),
              labels=['周一','周二','周三','周四','周五','周六','周日'])
plt.savefig('output/充电热力图.jpg', dpi=300)

# --------------------------
# 3. 用车规律分析（动态箱线图）
# --------------------------
# 按小时分析车速分布
plt.figure(figsize=(14, 7))
sns.boxplot(x='hour', y='speed', data=df,
                palette='Set2', showfliers=False)
plt.title('不同时段车速分布箱线图', fontsize=14)
plt.xlabel('小时', fontsize=12)
plt.ylabel('车速 (km/h)', fontsize=12)
plt.savefig('output/车速分布.jpg', dpi=300)

# --------------------------
# 4. 电池健康趋势（双指标折线图）
# --------------------------
battery_health = df.groupby('month').agg(
    max_voltage=('max_cell_voltage', 'mean'),
    min_voltage=('min_cell_voltage', 'mean')
).reset_index()

plt.figure(figsize=(14,7))
sns.lineplot(x='month', y='max_voltage', data=battery_health,
                marker='o', label='最高单体电压', linewidth=2.5)
sns.lineplot(x='month', y='min_voltage', data=battery_health,
                marker='s', label='最低单体电压', linewidth=2.5)
plt.title('电池电压衰减趋势', fontsize=14)
plt.xlabel('月份', fontsize=12)
plt.ylabel('电压值 (V)', fontsize=12)
plt.legend()
plt.grid(True, linestyle='--', alpha=0.7)
plt.savefig('output/电池健康趋势.jpg', dpi=300)
```

```
# --------------------------
# 5. 交互式三维散点图（电压-温度-故障）
# --------------------------
fig = px.scatter_3d(df.sample(2000),  # 抽样部分数据
                    x='total_voltage',
                    y='motor_temp',
                    z='soc',
                    color='alarm_level',
                    title='核心参数三维分布',
                    labels={
                        'total_voltage': '总电压 (V)',
                        'motor_temp': '电机温度 (°C)',
                        'soc': 'SOC (%)',
                        'alarm_level': '故障等级'
                    })
fig.write_html("output/三维参数分布.html")
```

代码 3　　问题 2 代码

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import shap
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import classification_report
from sklearn.inspection import PartialDependenceDisplay
from matplotlib import rcParams
from sklearn.inspection import permutation_importance

# 配置全局绘图参数
rcParams['font.sans-serif'] = ['SimHei']
rcParams['axes.unicode_minus'] = False



def load_data(file_path):
    """数据加载与初步处理"""
    df = pd.read_csv(file_path, parse_dates=['timestamp'])

    # 时间特征工程
    df['month'] = df['timestamp'].dt.month
    df['hour'] = df['timestamp'].dt.hour
    df['is_weekend'] = df['timestamp'].dt.weekday >= 5
```

```
    df['is_night'] = df['hour'].between(22, 5)

    # 动态特征计算
    df['voltage_change'] = df['total_voltage'].pct_change() * 100
    df['temp_gradient'] = df['motor_temp'].diff(60)  # 每小时温度变化

    # 充电状态增强编码
    charging_map = {
        0: {'mode': 'off', 'power_level': 0},
        1: {'mode': 'slow', 'power_level': 1},
        2: {'mode': 'fast', 'power_level': 2},
        3: {'mode': 'fault', 'power_level': np.nan}
    }
    return df.join(pd.json_normalize(df['charging_status'].map(charging_map)))


def feature_engineering(df):
    """特征工程处理"""
    # 选择关键特征
    features = [
        'total_voltage', 'total_current', 'soc',
        'motor_temp', 'voltage_change', 'temp_gradient',
        'hour', 'is_weekend', 'is_night', 'power_level'
    ]

    # 处理分类特征
    encoder = OneHotEncoder()
    encoded_features = encoder.fit_transform(df[['mode']]).toarray()
    feature_names = encoder.get_feature_names_out(['mode'])

    # 构建特征矩阵
    X = pd.concat([
        df[features],
        pd.DataFrame(encoded_features, columns=feature_names)
    ], axis=1)

    return X.fillna(0), df['alarm_level']


def train_model(X, y):
    """模型训练与评估"""
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y)

    # 随机森林参数（经贝叶斯优化）
```

```
    model = RandomForestClassifier(
        n_estimators=200,
        max_depth=8,
        min_samples_split=5,
        class_weight='balanced',
        random_state=42
    )
    model.fit(X_train, y_train)

    # 模型评估
    print("\n 模型评估报告：")
    print(classification_report(y_test, model.predict(X_test)))

    return model


def visualize_analysis(model, X, y):
    """可视化分析模块"""

    # 特征重要性分析
    importance = pd.Series(model.feature_importances_, index=X.columns)
    plt.figure(figsize=(10, 6))
    importance.nlargest(10).plot(kind='barh', color='#2ecc71')
    plt.title('Top10 故障预测特征重要性')
    plt.xlabel("重要性得分")
    plt.ylabel("特征")
    plt.tight_layout()
    plt.savefig("feature_importance.jpg")  # 保存图片
    plt.show()

    # SHAP 值分析
    explainer = shap.TreeExplainer(model)
    shap_values = explainer.shap_values(X)

    plt.figure(figsize=(10, 6))
    shap.summary_plot(shap_values, X, plot_type="bar", show=False)
    plt.title("SHAP 值分析")
    plt.savefig("shap_analysis.jpg")  # 保存图片
    plt.show()

    # 选择最重要的 2 个特征进行部分依赖分析
    top_features = importance.nlargest(2).index.tolist()

    # 确保 y 是多分类，并选择第一个类别作为 target
```

```
    unique_classes = np.unique(y)
    target_class = unique_classes[0] if len(unique_classes) > 1 else None

    if target_class is not None:
        PartialDependenceDisplay.from_estimator(model, X, features=top_features, target=target_class)
    else:
        PartialDependenceDisplay.from_estimator(model, X, features=top_features)

    plt.suptitle("部分依赖分析 (PDP)")
    plt.savefig("pdp_analysis.jpg")  # 保存图片
    plt.show()

if __name__ == "__main__":
    # 数据加载与处理
    df = load_data(".\output\combined_data.csv")

    # 特征工程
    X, y = feature_engineering(df)

    # 模型训练
    trained_model = train_model(X, y)

    # 可视化分析
    visualize_analysis(trained_model, X,y)

    print("分析完成！结果文件已保存：feature_importance.jpg, shap_analysis.jpg, pdp_analysis.jpg")
```

代码 4 岭回归代码

```
import pandas as pd
import numpy as np
from sklearn.linear_model import RidgeClassifier  # 使用 RidgeClassifier 替换 RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score, recall_score
from sklearn.model_selection import GridSearchCV

# 读取数据
df = pd.read_csv(".\output\combined_data.csv")

# 处理时间戳，提取 hour 作为特征
df["timestamp"] = pd.to_datetime(df["timestamp"])
df["hour"] = df["timestamp"].dt.hour

# 选择特征和目标变量
```

```
features = ["soc", "hour"]
target = "combined_anomaly"

# 去除缺失值
df = df.dropna(subset=features + [target])

# 划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(df[features], df[target], test_size=0.2, random_state=42)

# 训练 Ridge 分类模型
# 网格搜索调参
param_grid = {'alpha': [0.1, 1, 10, 100]}
ridge_model = RidgeClassifier(class_weight='balanced', random_state=42)     # 使用岭回归分类器
ridge_grid = GridSearchCV(ridge_model, param_grid, scoring='roc_auc')
ridge_model.fit(X_train, y_train)

# 预测
y_pred = ridge_model.predict(X_test)
y_prob = ridge_model.decision_function(X_test)   # 获取决策函数的值，作为概率预测

# 计算评估指标
auc = roc_auc_score(y_test, y_prob)
recall = recall_score(y_test, y_pred)

print(f"AUC: {auc:.4f}")
print(f"Recall: {recall:.4f}")
```

代码 5　　Lasso 回归代码

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score, recall_score

# 读取数据
df = pd.read_csv(".\output\combined_data.csv")

# 处理时间戳，提取 hour 作为特征
df["timestamp"] = pd.to_datetime(df["timestamp"])
df["hour"] = df["timestamp"].dt.hour

# 选择特征和目标变量
features = ["soc", "hour"]
target = "combined_anomaly"
```

```
# 去除缺失值
df = df.dropna(subset=features + [target])

# 划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(df[features], df[target], test_size=0.2, random_state=42)

# 训练 Lasso (L1 正则化) 逻辑回归模型
lasso_model = LogisticRegression(penalty='l1', solver='liblinear', class_weight='balanced', random_state=42)
lasso_model.fit(X_train, y_train)

# 预测
y_pred = lasso_model.predict(X_test)
y_prob = lasso_model.predict_proba(X_test)[:, 1]

# 计算评估指标
auc = roc_auc_score(y_test, y_prob)
recall = recall_score(y_test, y_pred)

print(f"AUC: {auc:.4f}")
print(f"Recall: {recall:.4f}")
```

代码 6　　随机森林代码

```
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier  # 引入随机森林分类器
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score, recall_score
from sklearn.preprocessing import StandardScaler

# 读取数据
df = pd.read_csv(".\output\combined_data.csv")

# 处理时间戳，提取 hour 作为特征
df["timestamp"] = pd.to_datetime(df["timestamp"])
df["hour"] = df["timestamp"].dt.hour

# 选择特征和目标变量
features = ["soc", "hour"]
target = "combined_anomaly"

# 去除缺失值
df = df.dropna(subset=features + [target])

# 重置索引
```

```
df = df.reset_index(drop=True)

# 划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(df[features], df[target], test_size=0.2, random_state=42)

# 将目标变量转换为 NumPy 数组
y_train = np.array(y_train)
y_test = np.array(y_test)

# 数据标准化
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 设置类权重，解决类别不平衡
class_weights = {0: 1, 1: 15}  # 通过调整权重帮助模型更重视少数类

# 创建随机森林模型
rf_model = RandomForestClassifier(n_estimators=200, random_state=42, class_weight=class_weights)

# 训练模型
rf_model.fit(X_train_scaled, y_train)

# 预测
y_pred = rf_model.predict(X_test_scaled)
y_prob = rf_model.predict_proba(X_test_scaled)[:, 1]  # 获取预测的概率

# 计算评估指标
auc = roc_auc_score(y_test, y_prob)
recall = recall_score(y_test, y_pred)

print(f"AUC: {auc:.4f}")
print(f"Recall: {recall:.4f}")
```

代码 7　　支持向量机代码

```
import pandas as pd
from sklearn.svm import LinearSVC
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score, recall_score

# 读取数据
df = pd.read_csv("./dataset/combined_data.csv")

# 处理时间戳
```

```
df["timestamp"] = pd.to_datetime(df["timestamp"])
df["hour"] = df["timestamp"].dt.hour

# 选择特征和目标变量
features = ["soc", "hour"]
target = "combined_anomaly"

# 去除缺失值
df = df.dropna(subset=features + [target])
print(f"Data shape after dropping missing values: {df.shape}")

# 随机抽样 10% 的数据
df_sample = df.sample(frac=0.1, random_state=42)

# 划分数据集
X_train, X_test, y_train, y_test = train_test_split(df_sample[features], df_sample[target], test_size=0.2, random_state=42)

# 特征缩放
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 训练线性 SVM
svm_model = LinearSVC(C=0.1, max_iter=10000, class_weight='balanced', random_state=42)
svm_model.fit(X_train_scaled, y_train)

# 预测
y_pred = svm_model.predict(X_test_scaled)
y_prob = svm_model.decision_function(X_test_scaled)

# 计算评估指标
auc = roc_auc_score(y_test, y_prob)
recall = recall_score(y_test, y_pred)

print(f"AUC: {auc:.4f}")
print(f"Recall: {recall:.4f}")
```

代码 8　　XGBoost 代码

```
import pandas as pd
import numpy as np
from xgboost import XGBClassifier  # 修改 1：导入 XGBoost
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score, recall_score
```

```
# 读取数据（保持原样）
df = pd.read_csv(".\output\combined_data.csv")
# 处理时间戳，提取 hour 作为特征（保持原样）
df["timestamp"] = pd.to_datetime(df["timestamp"])
df["hour"] = df["timestamp"].dt.hour

# 选择特征和目标变量（保持原样）
features = ["soc", "hour"]
target = "combined_anomaly"

# 去除缺失值（保持原样）
df = df.dropna(subset=features + [target])

# 划分训练集和测试集（保持原样）
X_train, X_test, y_train, y_test = train_test_split(df[features], df[target], test_size=0.2, random_state=42)

# 修改 2：使用 XGBoost 模型
xgb_model = XGBClassifier(
    n_estimators=100,              # 保持树数量一致
    random_state=42,               # 保持随机种子一致
    eval_metric='logloss',      # 添加 XGBoost 专用参数
    use_label_encoder=False,   # 禁用旧版标签编码
    scale_pos_weight=len(y_train[y_train==0])/len(y_train[y_train==1])   # 处理类别不平衡
)

# 模型训练（保持原样）
xgb_model.fit(X_train, y_train)

# 预测（保持原样）
y_pred = xgb_model.predict(X_test)
y_prob = xgb_model.predict_proba(X_test)[:, 1]   # 获取正类概率

# 计算评估指标（保持原样）
auc = roc_auc_score(y_test, y_prob)
recall = recall_score(y_test, y_pred)

print(f"AUC: {auc:.4f}")
print(f"Recall: {recall:.4f}")
```

代码 9　　LightGBM 代码

```
import pandas as pd
import numpy as np
import lightgbm as lgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score, recall_score

# 读取数据
df = pd.read_csv("./dataset/combined_data.csv")

# 处理时间戳，提取 hour 作为特征
df["timestamp"] = pd.to_datetime(df["timestamp"])
df["hour"] = df["timestamp"].dt.hour

# 选择特征和目标变量
features = ["soc", "hour"]
target = "combined_anomaly"

# 去除缺失值
df = df.dropna(subset=features + [target])

# 划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(df[features], df[target], test_size=0.2, random_state=42)

# 计算类别权重
class_weight_dict = {0: 1, 1: (len(y_train) / (2 * np.bincount(y_train)[1]))}
print(f"Calculated class weights: {class_weight_dict}")

# 训练 LightGBM 模型，增加 n_estimators
lgb_model = lgb.LGBMClassifier(n_estimators=150, num_leaves=35, max_depth=6,
class_weight=class_weight_dict, random_state=42)
lgb_model.fit(X_train, y_train)

# 预测
y_prob = lgb_model.predict_proba(X_test)[:, 1]

# 调整阈值
threshold = 0.28
predictions_adjusted = (y_prob >= threshold).astype(int)

# 计算评估指标
auc = roc_auc_score(y_test, y_prob)
recall = recall_score(y_test, predictions_adjusted)

print(f"AUC: {auc:.4f}")
```

```
print(f"Recall: {recall:.4f}")
```

代码 10　　LSTM 代码

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score, recall_score
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout

# 读取数据
df = pd.read_csv(".\output\combined_data.csv")

# 处理时间戳，提取 hour 作为特征
df["timestamp"] = pd.to_datetime(df["timestamp"])
df["hour"] = df["timestamp"].dt.hour

# 选择特征和目标变量
features = ["soc", "hour"]
target = "combined_anomaly"

# 去除缺失值
df = df.dropna(subset=features + [target])

# 重置索引
df = df.reset_index(drop=True)

# 划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(df[features], df[target], test_size=0.2, random_state=42)

# 将目标变量转换为 NumPy 数组
y_train = np.array(y_train)
y_test = np.array(y_test)

# 数据标准化
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 改变数据格式为 LSTM 所需的 3D 输入格式 [样本数, 时间步长, 特征数]
```

```
X_train_scaled = np.reshape(X_train_scaled, (X_train_scaled.shape[0], 1, X_train_scaled.shape[1]))
X_test_scaled = np.reshape(X_test_scaled, (X_test_scaled.shape[0], 1, X_test_scaled.shape[1]))

# 设置类权重，解决类别不平衡
class_weights = {0: 1, 1: 10}  # 通过调整权重帮助模型更重视少数类

# 构建 LSTM 模型
model = Sequential()
model.add(LSTM(64, activation='relu', input_shape=(X_train_scaled.shape[1], X_train_scaled.shape[2]),
return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))  # 输出层，适用于二分类问题

# 编译模型
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# 训练模型
model.fit(X_train_scaled, y_train, epochs=30, batch_size=64, validation_data=(X_test_scaled, y_test),
class_weight=class_weights)

# 预测
y_pred = model.predict(X_test_scaled)
y_prob = y_pred.flatten()  # 转换为 1D array 以计算 AUC

# 调整阈值进行 Recall 计算
threshold = 0.3  # 试试不同的阈值
y_pred_class = (y_prob > threshold)

# 计算评估指标
auc = roc_auc_score(y_test, y_prob)
recall = recall_score(y_test, y_pred_class)

print(f"AUC: {auc:.4f}")
print(f"Recall: {recall:.4f}")
```

代码 11　　GNN 代码

```
import pandas as pd
import numpy as np
from xgboost import XGBClassifier  # 修改 1：导入 XGBoost
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score, recall_score

# 读取数据（保持原样）
```

```
df = pd.read_csv(".\output\combined_data.csv")
# 处理时间戳，提取 hour 作为特征（保持原样）
df["timestamp"] = pd.to_datetime(df["timestamp"])
df["hour"] = df["timestamp"].dt.hour

# 选择特征和目标变量（保持原样）
features = ["soc", "hour"]
target = "combined_anomaly"

# 去除缺失值（保持原样）
df = df.dropna(subset=features + [target])

# 划分训练集和测试集（保持原样）
X_train, X_test, y_train, y_test = train_test_split(df[features], df[target], test_size=0.2, random_state=42)

# 修改 2：使用 XGBoost 模型
xgb_model = XGBClassifier(
    n_estimators=100,              # 保持树数量一致
    random_state=42,               # 保持随机种子一致
    eval_metric='logloss',      # 添加 XGBoost 专用参数
    use_label_encoder=False,   # 禁用旧版标签编码
    scale_pos_weight=len(y_train[y_train==0])/len(y_train[y_train==1])   # 处理类别不平衡
)

# 模型训练（保持原样）
xgb_model.fit(X_train, y_train)

# 预测（保持原样）
y_pred = xgb_model.predict(X_test)
y_prob = xgb_model.predict_proba(X_test)[:, 1]   # 获取正类概率

# 计算评估指标（保持原样）
auc = roc_auc_score(y_test, y_prob)
recall = recall_score(y_test, y_pred)

print(f"AUC: {auc:.4f}")
print(f"Recall: {recall:.4f}")
```

代码 12　　问题 4 代码

```
import pandas as pd
import numpy as np
import lightgbm as lgb
from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import roc_auc_score, recall_score

# 读取训练数据
df = pd.read_csv("./output/combined_data.csv")

# 处理时间戳，提取 hour 作为特征
df["timestamp"] = pd.to_datetime(df["timestamp"])
df["hour"] = df["timestamp"].dt.hour

# 选择特征和目标变量
features = ["soc", "hour"]
target = "combined_anomaly"

# 去除缺失值
df = df.dropna(subset=features + [target])

# 划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(df[features], df[target], test_size=0.2, random_state=42)

# 计算类别权重
class_weight_dict = {0: 1, 1: (len(y_train) / (2 * np.bincount(y_train)[1]))}
print(f"Calculated class weights: {class_weight_dict}")

# 训练 LightGBM 模型，增加 n_estimators
lgb_model = lgb.LGBMClassifier(n_estimators=150, num_leaves=35, max_depth=6,
class_weight=class_weight_dict, random_state=42)
lgb_model.fit(X_train, y_train)

# 预测
y_prob = lgb_model.predict_proba(X_test)[:, 1]

# 调整阈值
threshold = 0.28
predictions_adjusted = (y_prob >= threshold).astype(int)

# 计算评估指标
auc = roc_auc_score(y_test, y_prob)
recall = recall_score(y_test, predictions_adjusted)

print(f"AUC: {auc:.4f}")
print(f"Recall: {recall:.4f}")


df_month_10 = pd.read_excel("month_10.xlsx")
```

```
df_month_10["timestamp"] = pd.to_datetime(df_month_10["数据采集时间"])
df_month_10["hour"] = df_month_10["timestamp"].dt.hour

# 选择特征
features_month_10 = ["SOC", "hour"]

# 进行预测
X_month_10 = df_month_10[features_month_10]
y_prob_month_10 = lgb_model.predict_proba(X_month_10)[:, 1]

# 调整阈值进行预测
predictions_month_10 = (y_prob_month_10 >= threshold).astype(int)

# 将预测结果映射到报警等级（1-3）
df_month_10["报警_等级"] = np.clip(predictions_month_10 * 3, 1, 3)

# 保存预测结果到文件（只保存时间戳和预测的报警等级）
df_month_10[["数据采集时间", "报警_等级"]].to_csv("./predictions_month_10.csv", index=False)

print("预测结果已保存至 'predictions_month_10.csv'")
```

代码 13　　敏感性分析代码

```
import pandas as pd
import numpy as np
import lightgbm as lgb
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score, recall_score, confusion_matrix

# 读取训练数据
df = pd.read_csv("./output/combined_data.csv")

# 处理时间戳，提取 hour 作为特征
df["timestamp"] = pd.to_datetime(df["timestamp"])
df["hour"] = df["timestamp"].dt.hour

# 选择特征和目标变量
features = ["soc", "hour"]
target = "combined_anomaly"

# 去除缺失值
df = df.dropna(subset=features + [target])
```

```
# 划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(df[features], df[target], test_size=0.2, random_state=42)

# 计算类别权重
class_weight_dict = {0: 1, 1: (len(y_train) / (2 * np.bincount(y_train)[1]))}

# 训练 LightGBM 模型
lgb_model = lgb.LGBMClassifier(n_estimators=150, num_leaves=35, max_depth=6, class_weight=class_weight_dict, random_state=42)
lgb_model.fit(X_train, y_train)

# ---------- 新增敏感度分析部分 ----------
# 在测试集上预测概率
y_prob_test = lgb_model.predict_proba(X_test)[:, 1]

# 生成阈值列表并计算 Recall 和 FPR
thresholds = np.linspace(0, 1, 100)
recalls = []
fprs = []

for thresh in thresholds:
    y_pred = (y_prob_test >= thresh).astype(int)
    recalls.append(recall_score(y_test, y_pred))
    tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
    fpr = fp / (fp + tn)
    fprs.append(fpr)

# 绘制 Recall 和 FPR 曲线
plt.figure(figsize=(10, 6))
plt.plot(thresholds, recalls, 'b-', label='Recall')
plt.plot(thresholds, fprs, 'r-', label='FPR')
plt.xlabel('Threshold')
plt.ylabel('Value')
plt.legend(loc='best')
plt.title('Threshold vs. Recall & FPR')
plt.grid(True)
plt.show()

# 选择最优阈值（Recall 高且 FPR 适中）
# 方法 1: Youden 指数（Recall - FPR）最大化
youden = np.array(recalls) - np.array(fprs)
best_idx = np.argmax(youden)
best_threshold = thresholds[best_idx]
```

```
# 方法 2: 确保 Recall 不低于 80%时 FPR 最低（根据需要调整阈值）
# min_recall = 0.8
# valid_indices = np.where(np.array(recalls) >= min_recall)[0]
# if valid_indices.size > 0:
#         best_idx = valid_indices[np.argmin(np.array(fprs)[valid_indices])]
# else:
#         best_idx = np.argmax(recalls)
# best_threshold = thresholds[best_idx]

print(f"最优阈值: {best_threshold:.2f}")
print(f"对应  Recall: {recalls[best_idx]:.2f}, FPR: {fprs[best_idx]:.2f}")

# ----------  使用最优阈值预测 10 月数据  ----------
# 读取 10 月数据
df_month_10 = pd.read_excel("month_10.xlsx")

# 处理时间列
df_month_10["数据采集时间"] = pd.to_datetime(df_month_10["数据采集时间"])
df_month_10["hour"] = df_month_10["数据采集时间"].dt.hour

# 特征选择
features_month_10 = ["SOC", "hour"]
X_month_10 = df_month_10[features_month_10]

# 预测概率
y_prob_month_10 = lgb_model.predict_proba(X_month_10)[:, 1]

# 应用最优阈值生成预测标签
predictions_month_10 = (y_prob_month_10 >= best_threshold).astype(int)

# 映射报警等级
df_month_10["报警_等级"] = np.clip(predictions_month_10 * 3, 1, 3)
```