武漢紡織大學 数理科学学院
WUHAN TEXTILE UNIVERSITY School of Mathematical & Physical Sciences

《数据挖掘》课程设计（考试）论文（报告）评分表

姓名：李圣璇	班级：应数 12202 班
学号：2226010406	E-mail：3151496431@qq.com
项目代码地址：(data、picture、code) https://pan.baidu.com/s/1TRTfFK1evolrAYReLaTF6g?pwd=d62m 提取码: d62m	

考核项目	考核分数
数据采集工具的选择、数据来源可靠性及完整性（10 分）	
数据清洗与集成方法正确性（15 分）	
数据挖掘过程中描述性、探索性分析的准确性（20 分）	
数据可视化分析、对比方法、主要结论(观点)创新性（20 分）	
技术报告的表达流畅性（15 分）	
技术报告的结构合理性（10 分）	
可视化分析工具的恰当性、附录代码的规范性（10 分）	
总成绩（总分 100 分）	

评语：

教师签名：

日　　期：

目录

融合深度学习与机器学习的网瘾预测模型探索

姓名：李圣璇

学号：2226010406　　E-mail：3151496431@qq.com

摘　要：

目的 本研究旨在通过系统的数据显示和分析，提升对网瘾程度（sii）的预测准确性。随着互联网的普及，网瘾问题日益严重，影响了青少年的身心健康。因此，准确评估网瘾程度对于制定干预措施至关重要。本研究探讨了有效的数据预处理和特征工程方法，以支持后续的模型构建和评估，为相关政策制定提供数据支持。

方法 首先，对季节性数据进行了存储优化，将字符串类型转换为枚举类型以减少存储空间并提升计算效率。接下来，采用自编码器进行数据降维，提取关键特征，并设计基于机器学习的缺失值插补策略，确保数据的完整性和结构性。在数据预处理阶段，针对缺失值和异常值进行了系统的处理，首先识别并替换无穷大和无穷小的值，确保数据质量。随后，通过计算各特征的缺失值比例，选择有效特征进行插补，使用Lasso回归模型以及KNN填补器填补缺失值，以确保数据的稳定性和有效性。在模型构建过程中，采用了多种回归模型，包括TabNet、LGBMRegressor、XGBRegressor和CatBoostRegressor。为提高模型性能，采用投票回归模型集成不同算法的优点。每个基模型的超参数通过Optuna进行优化，设定了多个超参数，包括学习率、树的深度、叶子节点数量和数据采样比例等。通过分层K折交叉验证评估每组超参数的表现，确保模型的泛化能力，并寻找最优的超参数组合以最大化模型的预测能力。最后，将使用深度学习的投票模型与只使用机器学习的投票模型进行对比，进一步验证模型的性能。

结果 模型训练结果显示，使用深度学习和机器学习的综合模型在训练集上的平均二次加权Kappa系数（QWK）为0.7016，而验证集得分为0.4727，经过优化后验证集得分提升至**0.536**。这表明模型在训练集上表现良好，但存在过拟合的风险，需进一步提升其泛化能力。通过对比不同模型的表现，发现投票回归模型在处理复杂关系时具有明显优势，能够综合不同模型的优点，显著提高预测准确性。同时，在文章最后研究了只使用机器学习的投票模型在训练集上的得分为0.7595，在验证集上的得分为0.3926，优化后的得分为**0.456**，横向对比同样说明了模型的泛化能力需要进一步增强，经过纵向对比，验证了深度学习和机器学习综合投票模型的性能更佳。

结论 本研究通过有效的数据预处理、特征工程和模型优化，为网瘾程度的量化评估提供了科学的方法论支持。创新点在于采用自编码器进行特征降维与提取、基于机器学习的缺失值填补策略、融合Tabnet的深度学习框架，以及通过投票回归集成多种模型以提升预测性能。尽管模型在训练集上表现良好，但验证集的较低得分提示了过拟合问题，表明仍需进一步优化模型的泛化能力。未来的研究可以考虑引入正则化技术和优化特征工程参数，以增强模型的稳定性和预测准确性，为实际应用提供更为可靠的依据。

关键词：数据挖掘;网瘾程度;Lasso模型;Kappa系数;Tabnet;LGBMRegressor；XGBRegressor;CatBoostRegressor

Exploration of Internet addiction prediction model combining deep learning and machine learning

Shengxuan Li

1.2226010406 2. 3151496431@qq.com

Abstract:

Objective The aim of this study was to improve the prediction accuracy of the degree of Internet addiction (sii) through systematic data display and analysis. With the popularization of the Internet, the problem of Internet addiction is becoming more and more serious, which affects the physical and mental health of teenagers. Therefore, an accurate assessment of the extent of Internet addiction is essential for the development of interventions. This study explores effective data preprocessing and feature engineering methods to support subsequent model construction and evaluation, and provide data support for relevant policy making.

Methods Firstly, the seasonal data is optimized by converting the string type into enumeration type to reduce the storage space and improve the computing efficiency. Next, the autoencoder is used to reduce the dimensionality of the data, extract key features, and design a missing value interpolation strategy based on machine learning to ensure the integrity and structure of the data. In the data preprocessing stage, the missing

values and outliers are systematically processed, and the infinitesimal and infinitesimal values are first identified and replaced to ensure the data quality. Then, by calculating the proportion of missing values of each feature, selecting effective features for interpolation, using Lasso regression model and KNN filler to fill in the missing values, to ensure the stability and validity of the data. In the process of model construction, several regression models were used, including TabNet, LGBMRegressor, XGBRegressor and CatBoostRegressor. In order to improve the performance of the model, voting regression model is used to integrate the advantages of different algorithms. The hyperparameters of each base model were optimized through Optuna, and several hyperparameters were set, including learning rate, tree depth, number of leaf nodes, and data sampling ratio. The performance of each set of hyperparameters is evaluated by hierarchical K-fold cross-validation to ensure the generalization ability of the model and to find the optimal combination of hyperparameters to maximize the predictive power of the model. Finally, the voting model using deep learning is compared with the voting model using only machine learning to further verify the performance of the model.

Results The model training results showed that the average quadratic weighted Kappa coefficient (QWK) of the integrated model using deep learning and machine learning on the training set was 0.7016, and the validation set score was 0.4727. After optimization, the validation set score increased to **0.536**. This indicates that the model performs well on the training set, but there is a risk of overfitting, and its generalization ability needs to be further improved. By comparing the performance of different models, it is found that voting regression model has obvious advantages in dealing with complex relationships, and can synthesize the advantages of different models to significantly improve the prediction accuracy. Meanwhile, at the end of the paper, it is studied that the voting model using only machine learning has a score of 0.7595 on the training set, 0.3926 on the verification set, and **0.456** after optimization. Horizontal comparison also shows that the generalization ability of the model needs to be further enhanced. The performance of deep learning and machine learning integrated voting model is verified to be better.

Conclusion Through effective data preprocessing, feature engineering and model optimization, this study provides scientific methodological support for quantitative evaluation of Internet addiction degree. Innovations include autoencoders for feature dimensionality reduction and extraction, machine learning-based missing value filling strategies, integration of Tabnet's deep learning framework, and integration of multiple models through voting regression to improve predictive performance. Although the model performs well on the training set, the low score of the validation set suggests an overfitting problem, indicating that further optimization of the model's generalization ability is still needed. Future studies can consider introducing regularization technology and optimizing feature engineering parameters to enhance the stability and prediction accuracy of the model and provide a more reliable basis for practical application.

Key words：Data mining; Degree of Internet addiction; Lasso model; Kappa coefficient; Tabnet; LGBMRegressor; XGBRegressor; CatBoostRegressor

0 引 言

在当今数字时代，互联网已经成为儿童和青少年生活中不可或缺的一部分。尽管互联网为年轻人提供了丰富的学习和社交机会，但其过度使用也引发了越来越多的心理健康问题，如抑郁和焦虑。这些问题的出现不仅影响了儿童的心理状态，还可能对他们的成长和发展产生深远的影响。因此，深入了解儿童和青少年互联网使用的问题显得尤为重要。(邓验，2014)目前，评估儿童和青少年互联网使用情况的方法通常相对复杂，往往需要专业的心理评估和干预。这不仅给家庭带来了获取这些评估的困难，还可能因为文化和语言的障碍而导致信息的不对称。由于这些限制，许多家庭无法直接衡量子女的互联网使用问题，而是被迫依赖于相关的心理健康问题作为间接指标。与此相比，身体健康和健身指标显得更加易于获取，且几乎不需要专业的临床知识。过度使用电子设备的青少年通常会表现出不健康的身体习惯，例如不良的姿势、饮食不规律以及缺乏身体活动。这使得身体健康指标可以作为识别互联网使用问题的有效替代工具，尤其是在缺乏临床专业知识或合适评估工具的情况下。(Kuss D J,2014)本项目旨在开发一个预测模型，利用儿童的身体活动数据来检测有问题的互联网和技术使用的早期指标。这一模型的建立将为家庭和教育工作者提供重要的工具，使他们能够及时识别潜在问题并采取相应的干预措施，进而促进儿童形成更健康的数字习惯。通过本研究的努力，希望为创造一个更健康、更快乐的未来贡献力量，使儿童能够更负责任地驾驭他们日益复杂的数字环境。

1 数据预处理

1.1 数据集描述

健康大脑网络(HBN)数据集是大约五千名 5-22 岁年轻人的临床样本，这些年轻人都接受过临床和研究筛查。HBN 研究的目的是从客观生物学角度寻找能够改善心理健康和学习障碍诊断和治疗的生物标记。这项研究的两个要素将用于此次模型：身体活动数据（腕戴式加速度计数据、健康评估和问卷）和互联网使用行为数据。本次项目的目标是根据这些数据预测参与者的严重程度损伤指数(sii)，这是衡量互联网使用问题的标准指标。

1.1.1 HBN 仪器

train.csv 和 test.csv 中的表格数据包含来自各种仪器的测量值。每个仪器内的字段在 data_dictionary.csv 中描述。这些仪器包括：

Demographics-有关参与者年龄和性别的信息。

Internet Use-每天使用电脑/互联网的小时数。

Children's Global Assessment Scale- 心理健康临床医生使用的数字量表来评估 18 岁以下青少年的一般功能。

Physical Measures- 收集血压、心率、身高、体重和腰围、臀围测量数据。

FitnessGram Vitals and Treadmill- 使用 NHANES 跑步机协议评估心血管健康测量。

FitnessGram Child- 与健康相关的体能评估，测量五种不同的参数，包括有氧能力、肌肉力量、肌肉耐力、柔韧性和身体成分。

Bio-electric Impedance Analysis- 测量关键身体组成元素，包括 BMI、脂肪、肌肉和水分含量。

Physical Activity Questionnaire- 有关过去 7 天儿童参与剧烈活动的信息。

Sleep Disturbance Scale- 用于对儿童睡眠障碍进行分类的量表。

Actigraphy- 通过研究级生物追踪器客观测量生态体力活动。

Parent-Child Internet Addiction Test- 20 项量表，用于测量与强迫性使用互联网相关的特征和行为，包括强迫性、逃避性和依赖性。

该项目的目标 sii 源自数据字典中描述的此字段（PCIAT）：0for None、1for Mild、2forModerate 和 3for Severe。此外，每个参与者都分配了一个唯一的标识符 id。

1.1.2 活动记录文件和字段描述

在参与 HBN 研究期间，一些参与者被给予一个加速度计，在家中和日常生活中连续佩戴长达 30 天。

series_{train|test}.parquet/id={id}用作训练数据的系列，按 id 进行分区。每个系列都是对单个受试者跨越多天的加速度计数据的连续记录。

id 是与 train/test.csvid 中的字段对应的患者标识符。

step - 系列中每个观察的整数时间步长。

X, Y, Z- 测量佩戴在手腕上的手表沿每个标准轴所受到的加速度，以 g 为单位。

enmo- 根据 wristpy 包的计算和描述，ENMO 是所有加速度计信号（沿 x、y 和 z 轴，以重力为单位测量）的欧几里得范数减一，负值四舍五入为零。零值表示无运动周期。虽然这个空间中没有标准的加速度测量方法，但这是几个常用计算特征之一。

anglez- 根据 wristpy 包的计算和描述，Angle-Z 是从单个加速度计组件得出的度量，指的是手臂相对于水平面的角度。

non-wear_flag- 一个标志（0：手表正在佩戴，1：手表未佩戴），用于帮助根据 GGIR 定义确定手表被摘下的时间，该定义使用加速度计数据的标准偏差和范围。

light- 以勒克斯为单位测量环境光。

battery_voltage- 以 mV 为单位测量电池电压。

time_of_day- 一天中的时间代表数据采样的 5 秒窗口的开始，格式为%H:%M:%S.%9f。

weekday- 星期几，以整数编码，1 表示星期一，7 表示星期日。

quarter- 一年中的季度，从 1 到 4 的整数。

relative_date_PCIAT- 自进行 PCIAT 测试以来的天数（整数）（负数天数表示在进行测试之前已经收集了活动记录仪数据）。

sample_submission.csv -最终的结果文件。

1.2 存储优化

观察数据我们发现对于季节数据（Spring, Summer, Autumn, Winter）是使用字符串类型进行存储，在这里首先将字符串类型转化成枚举类型。枚举类型通常在内部使用整数值来表示各个类别，而不是使用字符串，因此大大减少了存储空间。同时在执行数据操作时，比如过滤、分组或聚合，整数运算通常比字符串操作要快。处理整数比处理字符串所需的计算资源少，因此可以提高操作的效率。

1.3 数据降维

在数据预处理阶段，采用了自编码器（AutoEncoder）方法进行数据降维。自编码器是一种无监督学习模型，通过将高维输入数据压缩到较低维度的潜在空间，能够有效提取出数据中最重要的特征。在本研究中，首先对输入数据进行了标准化处理，以确保各特征具有相同的尺度。随后，通过训练自编码器模型，将数据降维至指定的编码维度。该过程不仅有助于减少数据的复杂性，还能提高后续分析和建模的效率。最终，从自编码器中获得的编码特征被用于进一步的数据分析，以提升模型的性能和可解释性。

自编码器的均方误差计算（MSE）和交叉熵损失公式如下：

$$\text{MSE：}\quad L(x,\hat{x}) = \left\| x - \hat{x} \right\|^2 \qquad (1)$$

$$\text{交叉熵损失：}\quad L(x,\hat{x}) = -\sum_i \left[x_i \cdot \log\left(\hat{x}_i\right) + (1-x_i) \cdot \log\left(1-\hat{x}_i\right) \right] \qquad (2)$$

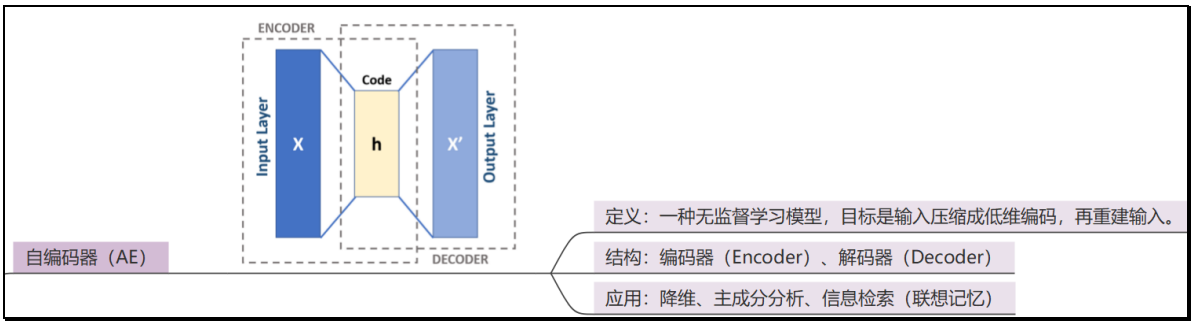

图 1 自编码器 AutoEncoder 概述

Figure 1AutoEncoder mind map

1.4 异常值/缺失值处理

本数据集在测量过程中一定会存在缺失值和异常值，同时该数据类型主要为表格型数据，因此针对表格型数据的缺失值填补和异常值处理成为数据预处理的优化要点。图 1 所示是针对本次数据得到的缺失值占比的可视化。由图 1 所示，很多字段都有缺失值，不妨以目标字段（sii）为依据，删除所有“sii 列” 为空所对应的缺失值，整理数据之后重新进行占比计算，于是我们得到了图 2，此时缺失值仍在部分字段中占了很高的比重。

同时数据中含有异常值，如 1.5 节中数据可视化的图 7 所示，在极小处仍然有数值存在。为了提高数据质量，首先检查数据集中是否存在无穷大（inf）或无穷小（-inf）的值，并将其替换为缺失值（NaN），以避免在模型训练过程中引发错误或不稳定。接着使用机器学习进行空缺值插补。定义了 Impute_With_Model 类，该类主要分为 3 个功能，首先初始化，设置允许的缺失值比例和最小样本数，接着初始化字典和特征列表；接着定义 find_features 方法计算有效特征（即：缺失值比例小于或等于允许的比例 40%的特征）；然后定义 fit_model 方法对每个特征进行模型拟合，若特征存在缺失值，则使用其他有效特征构建模型。如果有效特征数量和样本数量满足条件，则用有效特

征训练模型，并将模型保存到字典中；如果不满足条件，则保存特征的均值作为替代。最后使用 LassoCV 模型对有效特征进行填充，其余则使用均值进行填充。此处对于缺失值的处理保留数据的结构和特征之间的关系。同时，通过设置缺失值比例的阈值，确保了模型的稳定性和有效性，因此在一定程度上提高了后续数据分析的准确性。

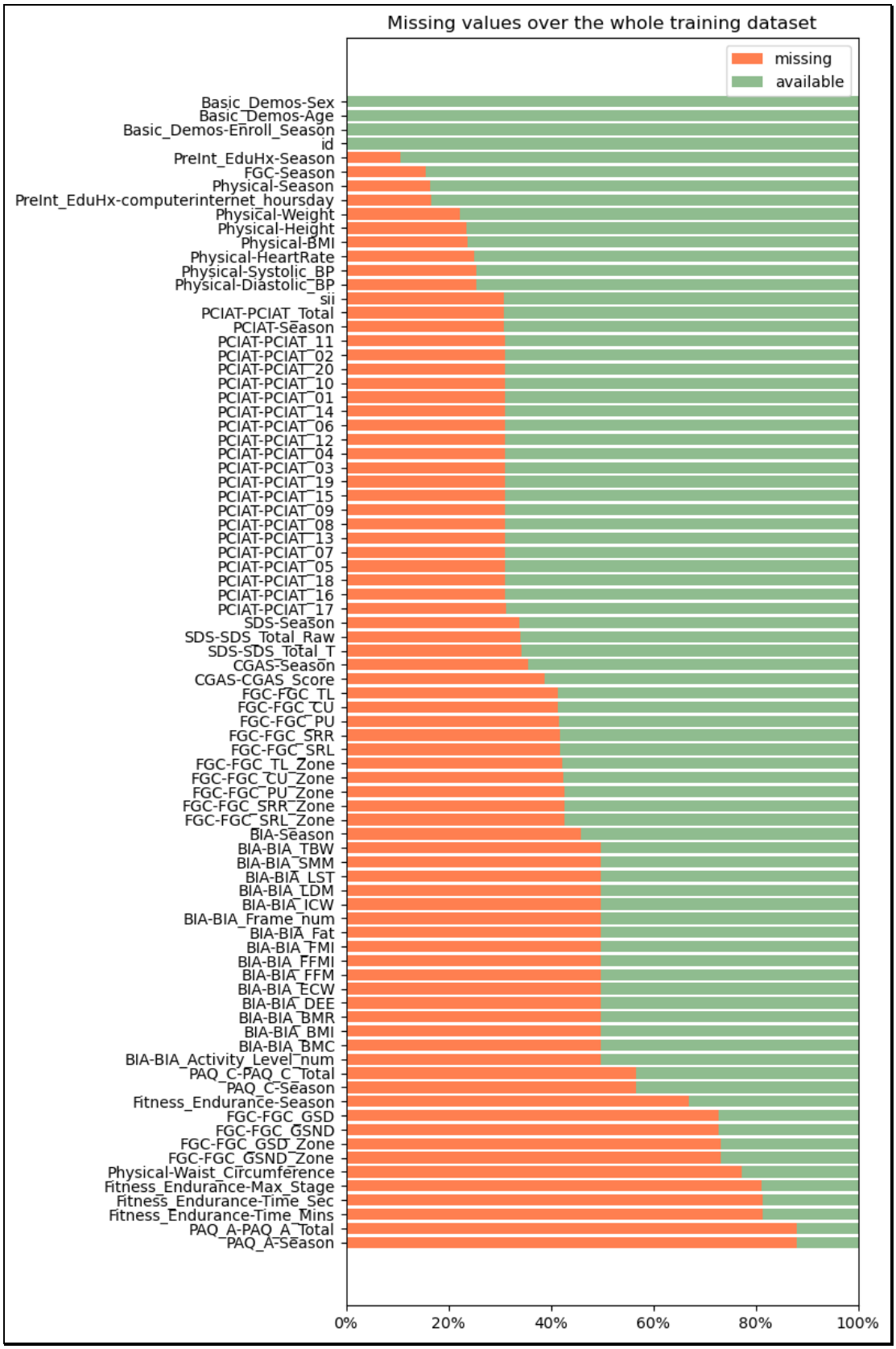

图 2 数据缺失值可视化（训练集）

Figure 2Missing value visualization

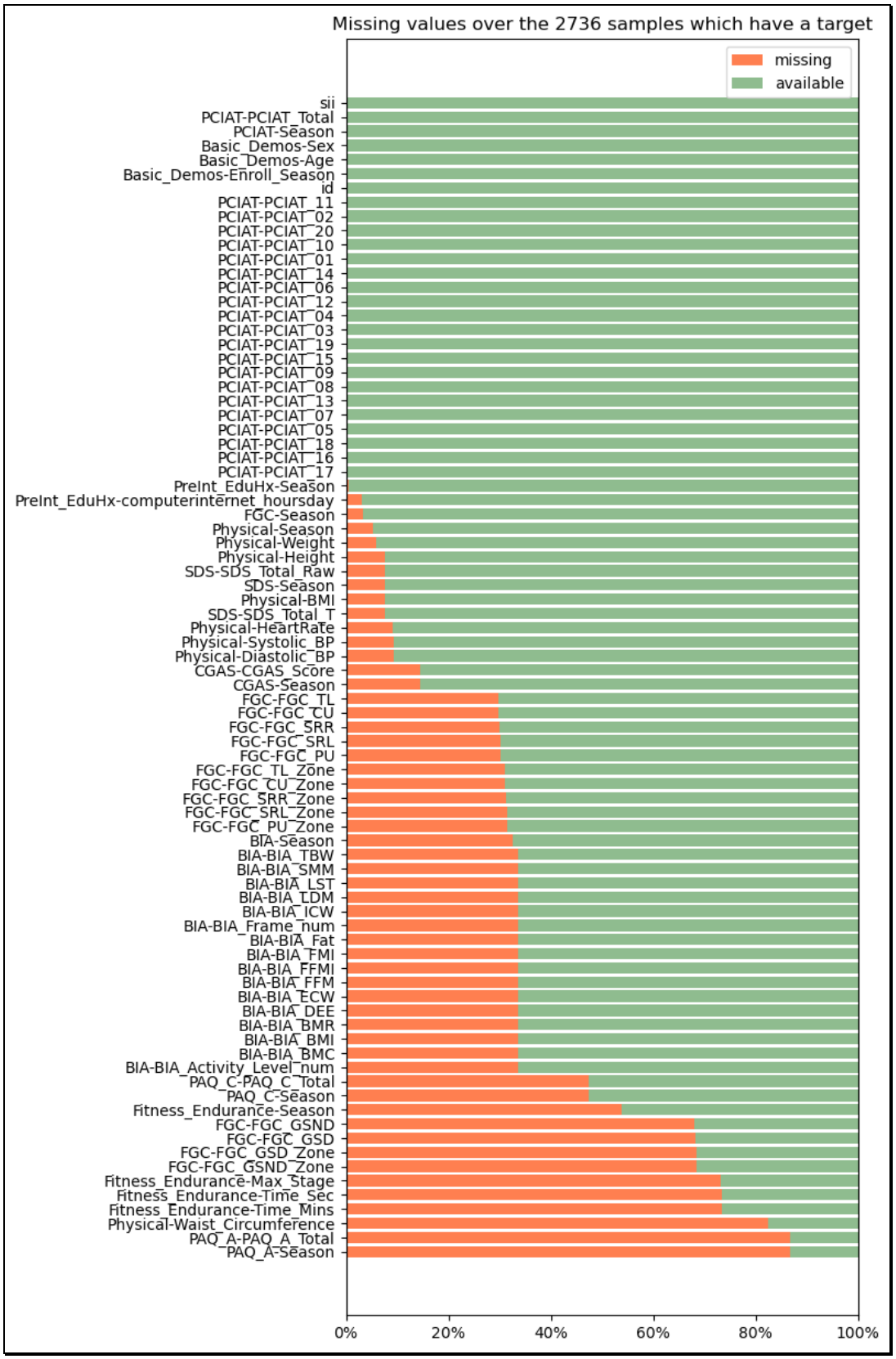

图 3 缺失值可视化（处理后）

Figure 3 Missing value visualization (after processing)

1.5 数据可视化

如下是针对数据进行的可视化绘图，从中可以更直观的了解到数据的分布特征。

如下是对于客观数据以及生理数据的记录，从图 3 可以发现入学季节分布较为均匀，春季人数最多。

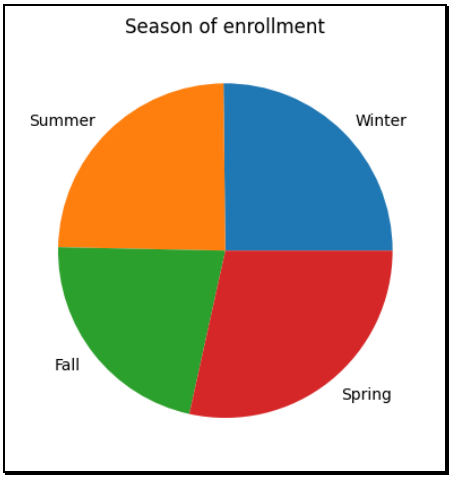

图 4 入学季节分布

Figure 4 Season of enrollment

从图 4 中可以发现，测试者男生占比多于女生。

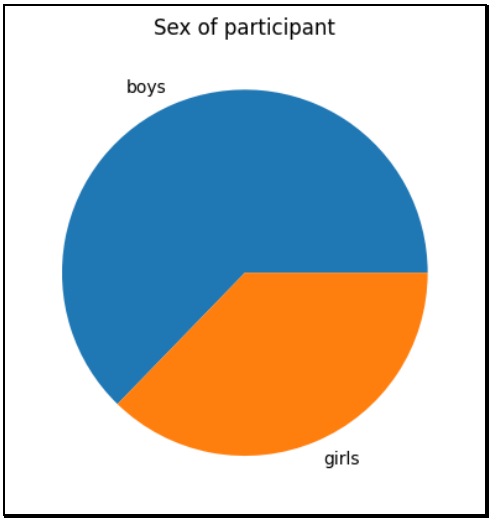

图 5 参与者性别分布

Figure 5 Sex of participant

从图 5 中可以发现，测试者年龄主要集中于 5-19 岁，其中 8-9 岁分布最为密集。

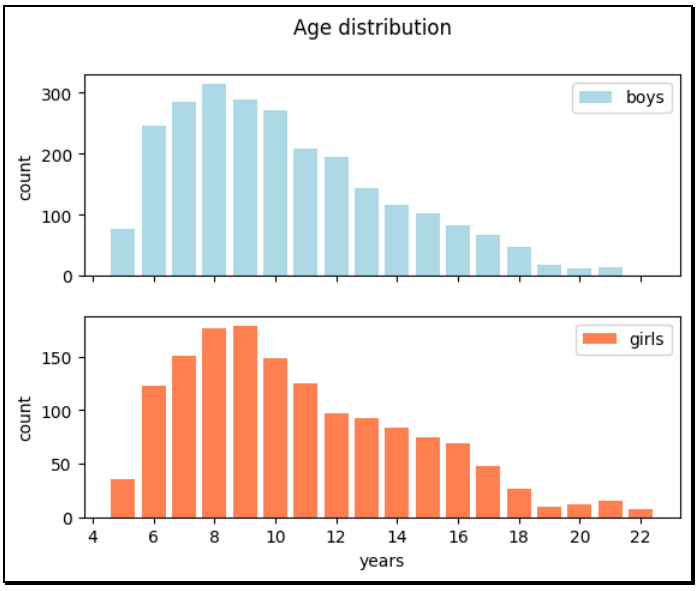

图 6 年龄分布

Figure 6 Age distribution

从图 6 中可以发现，男生和女生网瘾程度从无到重度人数依次递减，男生网瘾群体占比较女生大一些。

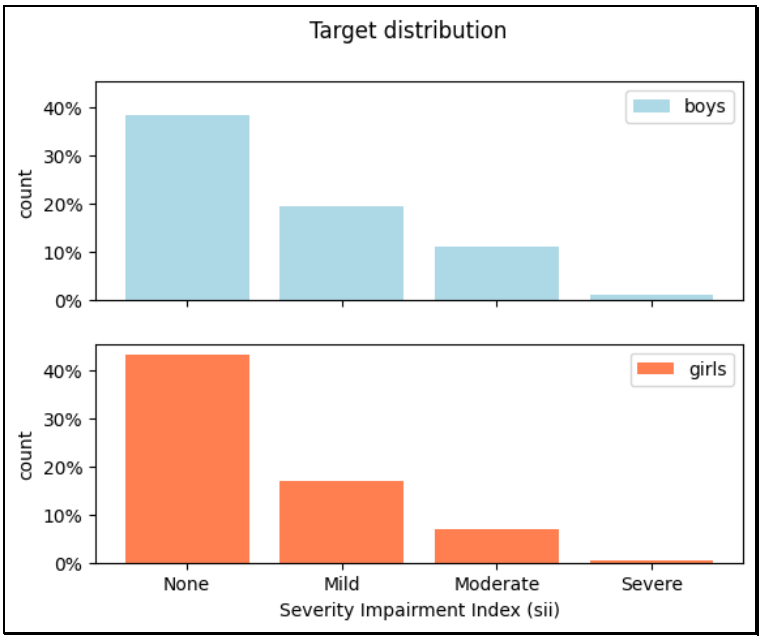

图 7 目标(sii)分布（训练集）

Figure 7 Target (sii) distribution (training set)

对于图 7 的心率测量可以发现在 0-50 范围内有异常值出现。

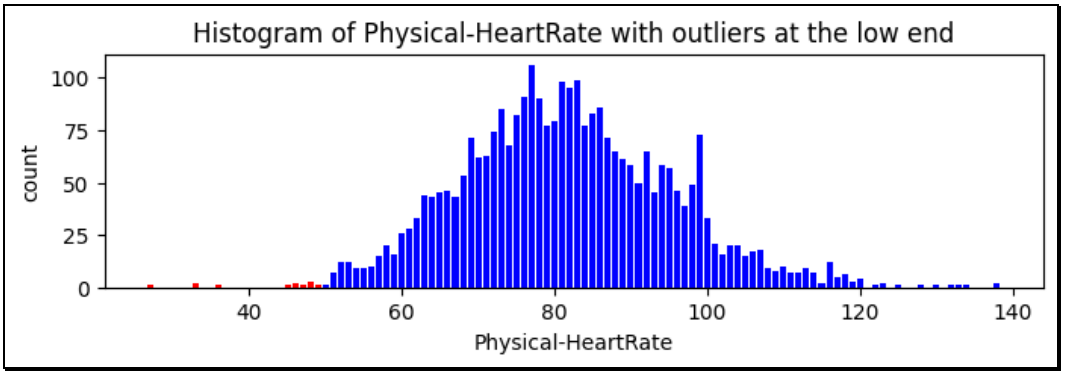

图 8 心率测量分布

Figure 8 Physical-HeartRate distribution

在图 8 的睡眠障碍量表中，可以发现主要集中于 26-50，但是少部分儿童存在睡眠障碍问题。

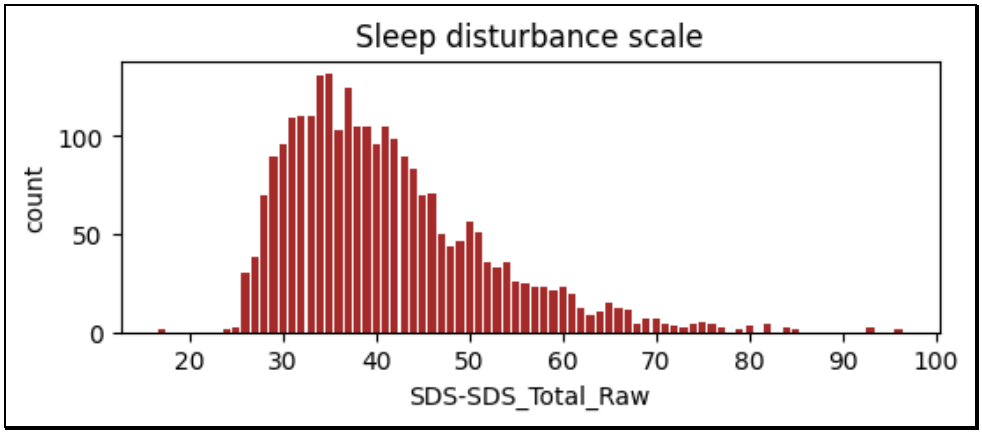

图 9 睡眠分布

Figure 9 Sleep disturbance scale

除了以上的生理数据之外，还有活动记录仪的数据，活动记录仪是一种监测人类休息/活动周期的非侵入性方法。一个小型活动记录仪，也称为活动测量传感器，佩戴一周或更长时间来测量总运动活动。该装置通常装在戴在手腕上的手表状包装中。活动记录仪单元所经历的运动会被连续记录下来，有些单元还会测量曝光量。有四分之一参与

者（准确地说是 996 人）的活动记录文件。文件名为 part-0.parench。因此针对这部分数据采用时间序列的方式进行呈现。

表 1 记录仪数据

Table 1 Recorder data

step	X	Y	Z	enmo	anglez	non-wear_flag	light
u32	f32	f32	f32	f32	f32	f32	f32
0	0.014375	-0.02011	-0.99536	0.00106	-88.4453	0	41
1	0.014167	-0.02328	-0.99616	0.000289	-88.3722	0	41
2	0.014036	-0.02296	-0.99632	0.000301	-88.3564	0	41.5
3	0.013593	-0.02205	-0.99676	0.002278	-88.5759	0	37.5
4	-0.06177	-0.06532	-0.97306	0.092321	-88.3913	0	55.66667
...	...	...	...	...	...	...	...
287174	-0.40743	0.091612	-0.37776	0.039733	-43.3194	0	7
287175	-0.70357	0.016187	0.15956	0.03598	14.12139	0	7
287176	-0.20961	-0.4697	0.636573	0.097799	44.99857	0	7
287177	-0.39038	0.284386	0.147654	0.057826	7.726313	0	7
287178	-0.48903	0.179624	-0.50961	0.077749	-36.995	0	7

表 2记录仪数据（附表1）

Table 2 Recorder Data (Schedule 1)

step	battery_voltage	time_of_day	weekday	quarter	relative_date_PCIAT
u32	f32	i64	i8	i8	f32
0	4195	4.41E+13	2	2	5
1	4194.833	4.41E+13	2	2	5
2	4194.667	4.41E+13	2	2	5
3	4194.5	4.41E+13	2	2	5
4	4199	4.48E+13	2	2	5
...	...	...	...	...	...
287174	3695	3.29E+13	1	3	53
287175	3695	3.29E+13	1	3	53
287176	3695	3.29E+13	1	3	53
287177	3695	3.29E+13	1	3	53
287178	3695	3.29E+13	1	3	53

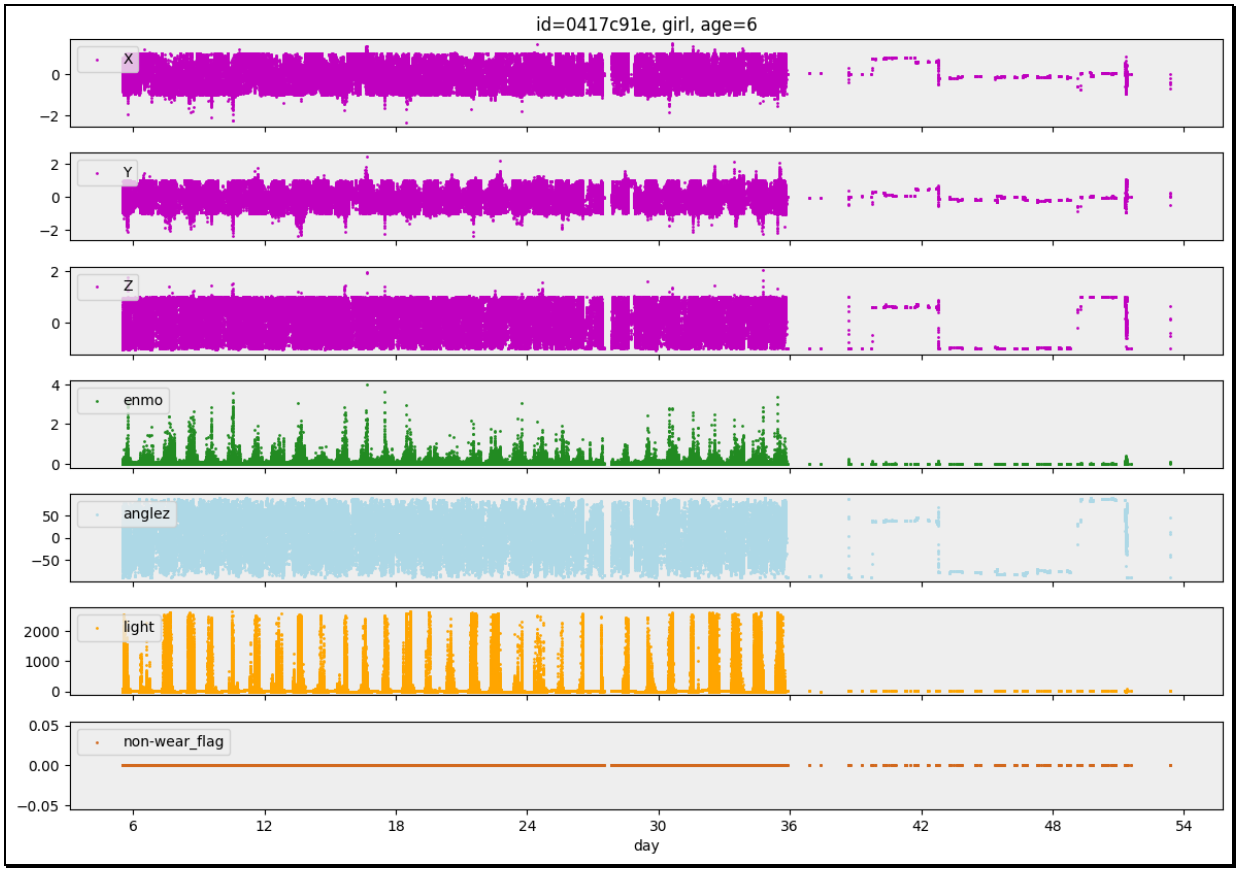

图 10 时序数据可视化（id=0417c91e）

Figure 10 Visualization of time series Data (id=0417c91e)

结合表格数据和图像可以发现这个 6 岁的小女孩的一种日常模式。可以发现这个女孩是一个右撇子，戴了 31 天的加速计，然后把它摘了下来。数据集有一个非 wear_flag 列，但该参与者的标志始终为零。女孩所处的环境每天的照度超过 2500 勒克斯（设备测量的照度不能超过 2500 勒克司）。如此高的照度意味着她在户外或在一个有巨大窗户的房间里。并且透过她的 enmo 值几乎每天都在 2 以上可以发现这个女孩经常活动。时间序列通常每 5 秒包含一次测量，但缺少一些时间步长，没有记录在什么条件下跳过时间步。对于该数据更倾向于使用 enmo 和 light 进行分析，nonwear_flag 值大多为 0，分析的参考性较低。再来看一个 15 岁男生的时序数据图，如图 10 所示。

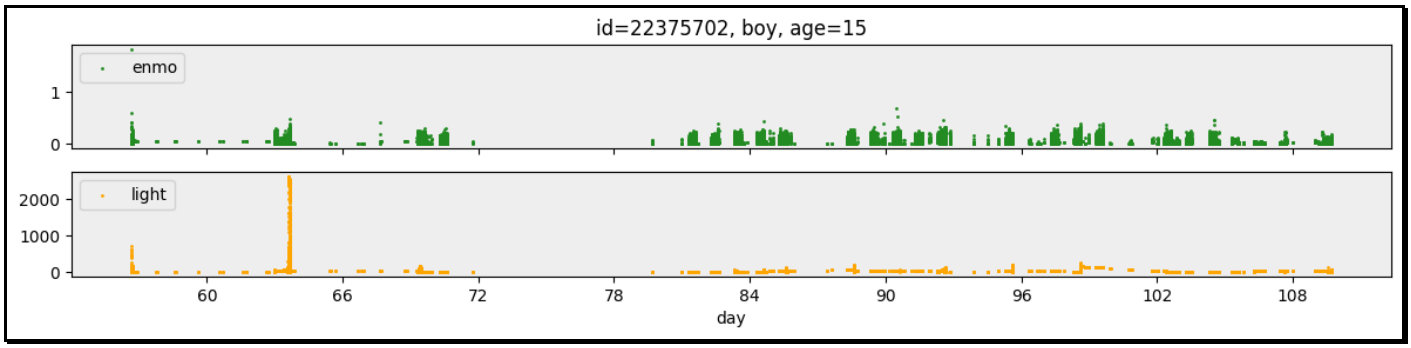

图 11 时序数据可视化（id=22375702）

Figure 11 Visualization of time series data (id=22375702)

根据图像可以大体判断这是一个 15 岁的左撇子男孩，很少活动（enmo<0.5），并且一个月很少见到阳光。而对于左撇子和右撇子的判断，只需要根据活动仪返回数据中加速度计 X 的坐标返回正数或者负数判断即可。

以上就完成了对数据的整体印象以及数据预处理操作，一个良好的数据预处理决定后续数据分析的效果以及模型的性能。本项目的最终目标是根据测量数据对网瘾程度（sii）进行 0-3 的判断，该问题可以采用两种思路解决：一种是看作分类问题，可以采用 LightGBM 等机器学习方法或深度学习对数据进行分类，也就是预测一个类别；另一种是看作回归问题，即将问题通过机器学习或深度学习建模为回归函数，最终得到的数值按照四舍五入确定为 0-3 的类别，也就是预测一个数。

2 特征工程

特征提取的方法有很多，项目的数据类别主要分为采集到的生理/物理数据和时序数据，前者为离散型，后者为连续型，因此针对不同的数据采取不同特征提取的方式。

2.1 物理数据/生理数据特征提取

由于本类数据表现为离散型，从而可以通过创建新特征（如年龄组、各类体能测试的标准化值等）的方法来增强模型的表现，对原始数据进行转换、组合或分解，以提取有用的信息。

首先，移除了与季节相关的特征列，以减少数据的复杂性并避免潜在的噪声影响。其次，根据领域知识和变量之间的关系，创建了多个新特征，这些特征旨在捕捉重要的生理和行为指标的相互作用。

具体包括：

1) BMI 与年龄的乘积 （BMI_Age）：反映身体质量指数与个体年龄的关系。
2) 上网时长与年龄的乘积（Internet_Hours_Age）：表征上网习惯与年龄之间的交互影响。
3) BMI 与上网时长的乘积（BMI_Internet_Hours）：探索身体质量指数与上网时间的关系。
4) 体脂与 BMI 的比值（BFP_BMI）：提供体脂和身体质量指数之间的相对关系。
5) 无脂体重与体脂的比值（FFMI_BFP）和脂肪质量与体脂的比值（FMI_BFP）：用于分析无脂体重和脂肪质量的相对成分。
6) 瘦体重与总水分的比值（LST_TBW）：衡量身体水分与瘦体重的关系。
7) 体脂与基础代谢率的乘积（BFP_BMR）以及体脂与日常能量消耗的乘积（BFP_DEE）：用于评估体脂对基础代谢和日常消耗的影响。
8) 基础代谢率与体重的比值（BMR_Weight）和日常消耗与体重的比值（DEE_Weight）：分析基础代谢和日常能量消耗相对于体重的影响。
9) 肌肉量与身高的比值（SMM_Height）和肌肉与脂肪的比值（Muscle_to_Fat）：探讨肌肉和脂肪成分的相对比例。
10) 水合状态与体重的比值（Hydration_Status）：反映身体水分状态。
11) 胞内水分与总体水分的比值（ICW_TBW）：用于分析细胞内液体与整体水分的关系。
12) BMI 与心率的乘积（BMI_PHR）：探索 BMI 与心率之间的关系。

以上是综合整体离散数据提出的具有代表性的特征进行新特征的创建，通过这些特征旨在捕捉生理特征之间的复杂关系，从而为后续的分析和模型构建提供更为丰富的信息。

2.2 时序数据特征提取

针对时序数据的特征提取主要采用一种基于自编码器的特征提取方法，并结合模型驱动的缺失值填充技术，以优化训练集和测试集的特征表现。首先从数据集中加载训练集和测试集，并提取与时间序列相关的数据。通过删除时间序列数据中的“id” 列，确保数据集中的特征列保持一致性。接着，我们使用自编码器对训练集和测试集进行编码，从而实现数据降维的同时提取特征。

自编码器是一种无监督学习算法，能够学习数据的低维表示。设置编码维度为 60，经过 100 个 epoch 的训练，生成了训练集和测试集的编码特征。这一步骤的关键在于自编码器的有效性，它能够捕捉到数据中的潜在结构，从而提取出重要特征。

在特征提取过程中，缺失值的存在会影响模型的训练效果。因此，设计了一个基于模型的缺失值填充策略。使用 Lasso 回归作为基础模型，创建了一个名为 Impute_With_Model 的类，该类能够针对每个特征拟合相应的模型，并进行缺失值填充。该方法首先计算每个特征的均值，并在模型拟合时检查缺失值。对于缺失值较少的特征，我们使用 Lasso 模型进行填充；对于缺失值较多的特征，则使用均值填充。这种灵活的填充策略在保持数据完整性的同时，最大程度地减少了信息损失。在此处衡量是否为有效特征采取比例的形式，即定义数据缺失值空缺小于等于 40% 为有效特征，使用 Lasso 模型进行填充，反之采用均值填充。同时，该比例可以进行进一步优化以达到数据增强的效果。

在完成缺失值填充后，我们对数据集进行了特征工程。特征工程的目标是生成新的特征，以提高模型的表现。

通过对训练集和测试集进行特征提取和转换，确保了数据的质量和可用性。此外，还删除了包含过多缺失值的样本，从而进一步优化数据集。

3 回归模型设计

3.1 深度学习模型：TabNet

TabNet 是一种新颖的深度学习模型，专门设计用于处理表格数据。(Arik, S. Ö., & Pfister, T. 2021).它结合了深度学习的强大表达能力和传统树模型的优势，能够有效地处理特征的异质性和缺失值。TabNet 的核心创新在于其使用了可解释的注意力机制，通过动态选择特征来进行决策，从而提高了模型的透明度和可理解性。TabNet 的架构包含多个决策步骤，每个步骤都可以独立选择特征并进行处理。这种特征选择机制使得模型在不同的输入条件下能够灵活调整，从而提高了预测的准确性。此外，TabNet 采用了稀疏性正则化，能够有效地减少过拟合，提高模型的泛化能力。

与其他模型相比，TabNet 在处理具有高维特征的表格数据时展现了显著的性能优势，特别是在小样本和不平衡数据集上。它的训练过程也相对高效，能够利用现代深度学习框架进行加速，使得 TabNet 成为一种理想的选择，适用于金融、医疗、市场营销等多个领域的数据分析任务。

由于本项目主要为表格型数据，为了进行回归预测，采用了 TabNet 模型进行回归任务的实现，具体通过 TabNetWrapper 类对其进行封装，以便于与 scikit-learn 框架的兼容性。该类实现了模型的初始化、训练、预测以及缺失值处理等功能。

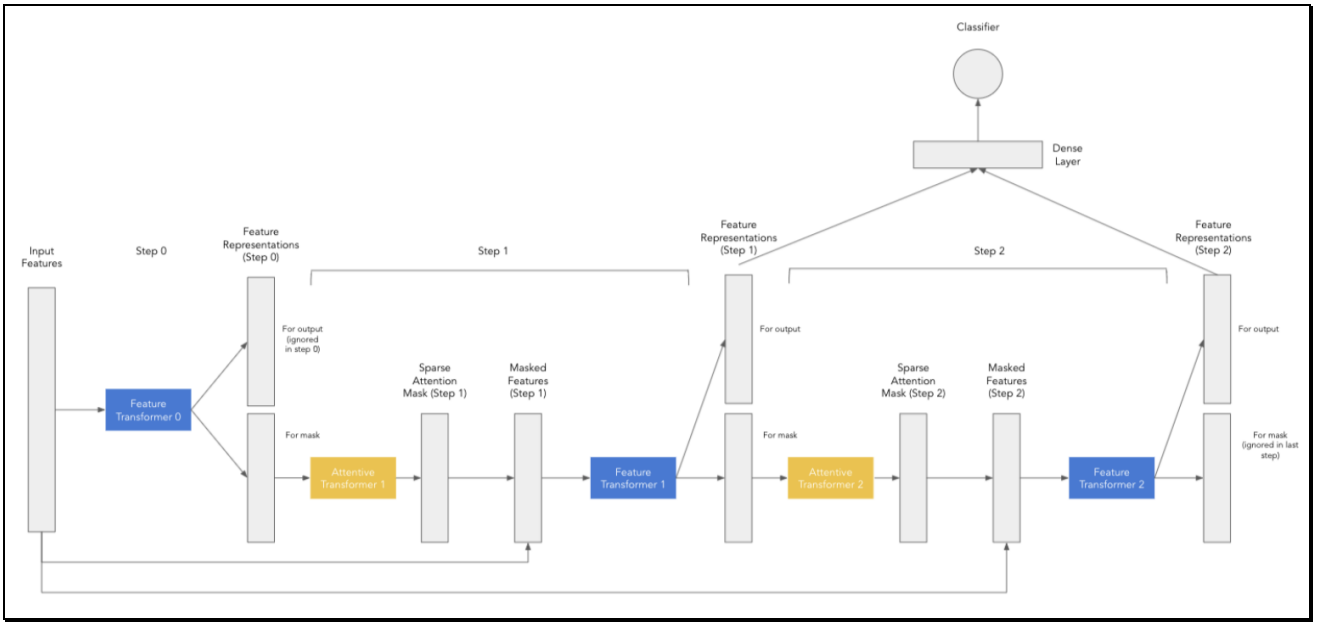

图 12 TabNet 流程

Figure 12 TabNet process

3.2 Tabnet 模型的封装与训练

3.2.1 模型初始化与超参数设置

在 TabNetWrapper 类的构造函数中，通过关键字参数传递 TabNet 模型的超参数。这些超参数包括决策预测层的宽度、注意力嵌入的宽度、模型的步骤数、特征选择的正则化系数等。此外，模型初始化时还设置了一个简单插补器，采用中位数策略处理缺失值，并定义了用于保存最佳模型的路径。

3.2.2 模型训练过程

在 fit 方法中，首先对输入特征进行缺失值插补，随后将数据集分为训练集和验证集。通过 train_test_split 函数将数据按 80/20 的比例进行划分，确保模型训练的多样性。接下来，调用 TabNet 模型的 fit 方法进行训练，设置了最大训练轮数、早停机制、批量大小以及回调函数等参数。其中，TabNetPretrainedModelCheckpoint 回调类用于监控验证集的均方误差（MSE），并在每轮训练结束时保存表现最优的模型。

3.2.3 模型的评估与特征重要性

训练完成后，若存在保存的最佳模型文件，程序将加载该模型，并在训练期间删除临时文件。此外，TabNetWrapper 类还通过 feature_importances_属性反映了模型训练后各特征的重要性，为模型的解释性提供了支持。

3.2.4 预测方法

在 predict 方法中，输入特征同样经过缺失值插补处理，随后使用训练好的 TabNet 模型进行预测，返回结果为一维数组，符合 scikit-learn 的接口要求。

3.2.5 超参数配置

TabNet 的超参数配置在 TabNet_Params 字典中进行定义，包括学习率、权重衰减、调度器参数等，以确保模型的优化过程高效且稳定。

3.3 机器学习模型

3.3.1 LGBM 回归器

LGBMRegressor（https://lightgbm.readthedocs.io/en/latest/pythonapi/lightgbm.LGBMRegressor.html）是微软开发的 LightGBM 库中的一种高效梯度提升树模型。其设计旨在处理大规模数据集，并提供快速的训练速度和高效的内存使用。LGBMRegressor 采用基于直方图的学习方法，通过将连续特征分割成离散的区间，显著减少了内存占用和计算时间。此外，该模型支持并行计算，从而提高了训练效率。

LGBMRegressor 的一个显著特点是其支持类别特征的直接输入，无需进行独热编码（One-Hot Encoding），这有助于降低计算复杂度并提升模型的性能。该模型在多种回归任务中表现优异，尤其在大规模数据和高维特征的情况下，能够有效捕捉数据中的复杂模式。(Guolin Ke,2017)

3.3.2 XGB 回归器

XGB Regressor（https://xgboost.readthedocs.io/en/latest/install.html）是 XGBoost 库中的一种广泛使用的梯度提升算法。其核心思想是通过集成多个弱学习器（通常是决策树）来构建一个强学习器，以提高预测性能。XGBoost 以其高效的计算能力和灵活性而著称，适用于各种类型的回归和分类问题。(Omarzai, F. 2024)

XGBoost 的优势体现在其正则化机制和处理缺失值的能力，能够有效防止过拟合并提高模型的泛化能力。此外，XGB Regressor 支持并行化和分布式计算，使其在大数据环境下依然能够保持高效的训练速度。

如图 13 所示，图中展示了一个集成学习模型的训练过程，特别是基于树的回归方法。数据集 X 被输入到多个决策树（如 Tree1 和 Tree2），每棵树通过目标函数进行节点划分，以最小化残差。每棵树生成的残差（Residual）被计算并用于进一步的模型优化。最终，所有树的输出通过求和聚合，形成最终的预测结果 $\sum f_k\left(X,\theta_k\right)$。该过程展示了如何通过多棵树的组合来逐步改进模型性能。

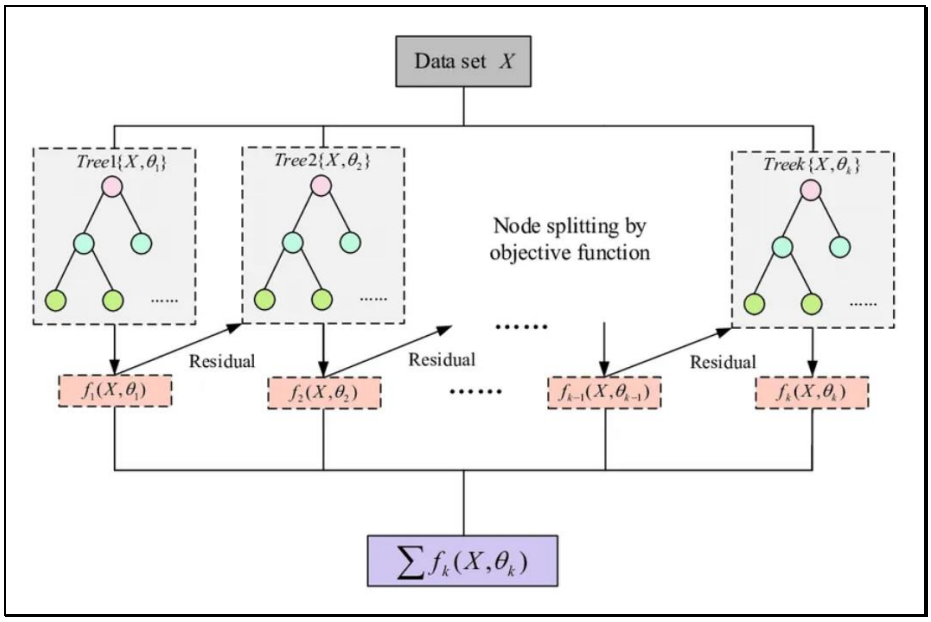

图 13 XGB 工作流程

Figure 13 XGB workflow

3.3.3 CatBoost 回归器

CatBoost Regressor（https://catboost.ai/docs/en/concepts/model-analysis）是由 Yandex 开发的一种基于梯度提升的决策树算法，专门针对类别特征的处理进行了优化。CatBoost 的“类别”指的是模型能够有效处理非数值特征，无需进行复杂的预处理（如独热编码）。这种特点使得 CatBoost 在处理包含大量类别变量的数据集时表现出色，减少了数据预处理的复杂性。

该模型采用了一种独特的顺序提升策略，通过对类别特征进行编码以保持信息的完整性，并减少了过拟合的风险。CatBoost Regressor 还具有出色的性能和可解释性，能够输出特征的重要性评分，为模型的决策过程提供了透明度。此外，CatBoost 的训练速度快且易于使用，使其在实际应用中得到了广泛的认可。

3.4 Voting 回归模型

在本项目中，最终利用投票回归模型（VotingRegressor）来集成多个基于不同算法的回归模型，以实现更高的预测性能。如图 14 所示，展示了使用多个分类模型进行集成学习的流程。首先，从训练集中生成多个分类模型 $C_1,C_2,\ldots,C_m$。对于新数据，这些模型封面别输出预测结果 $P_1,P_2,\ldots P_m$。最终，通过投票机制对所有模型的预测结果进行整合，形成最终预测结果 P_f。投票回归模型通过组合多个学习器的预测结果，利用其集成优势来提升整体预测的准确性和鲁棒性。

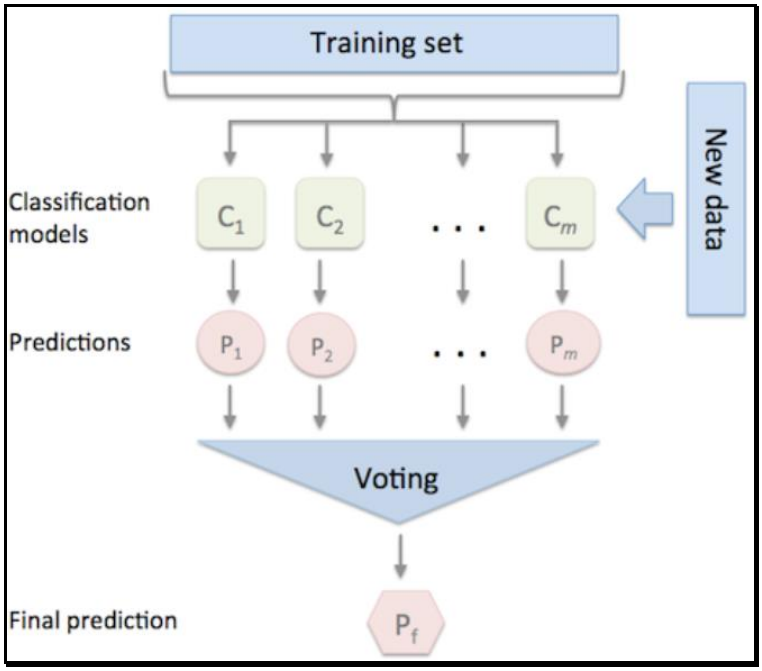

图 14 投票模型流程

Figure 14 Voting model process

首先选择了上述描述的四种基模型：LightGBM、XGBoost、CatBoost 和 TabNet。每个模型在处理表格数据时展现了独特的优势，简要概括如下：

(i) LightGBM：采用基于直方图的学习方法，能够高效处理大规模数据集。

(ii) XGBoost：以其正则化机制和强大的并行计算能力而著称，适合于复杂的回归任务。

(iii) CatBoost：特别擅长处理类别特征，减少了数据预处理的复杂性。

(iiii) TabNet：通过注意力机制动态选择特征，增强了模型的可解释性和性能。

在构建投票回归模型时，为每个基模型分配了不同的权重，以反映其在最终预测中的影响力。具体的权重设置为：Light GBM 的权重为 4.0，XGBoost 为 3.0，CatBoost 和 TabNet 均为 5.0。这种权重配置是基于各模型在验证集上的表现，并考虑了它们的复杂性和重要性。

完成模型构建后，将训练好的投票回归模型应用于测试集进行预测。此过程通过自定义函数 TrainML 实现，该函数接收投票模型和测试数据作为参数。通过这种集成方法，期望能够提高最终模型的预测准确性，并在数据集上取得更好的表现。

3.5 超参数选取

为了获取各个机器学习中的超参数（如：学习率、树的深度、叶子节点数量、叶子节点上的最小数据量等），本项目使用 optuna（https://optuna.readthedocs.io/zh-cn/latest/index.html）进行超参数调优，以提高机器学习模型的性能。下面以 LightGBM 为例阐释具体步骤。

3.5.1 基础参数设定：

首先定义了 LightGBM 模型的一组基础参数。这些参数包括目标函数设定为回归（objective: 'regression'），评估指标为均方根误差（metric: 'rmse'），以及设置最大迭代次数和早停轮次，以防止过拟合。

3.5.2Optuna 优化目标：

在超参数优化过程中，构建了一个优化目标函数。该函数通过使用 Optuna 的 suggest_float 和 suggest_int 方法，动态选择以下超参数：

1）学习率（learning_rate）：在 0.01 到 0.2 之间选择，控制模型更新的步伐。

2）叶子节点数（num_leaves）：范围为 20 到 150，决定树的复杂度。
3）最大深度（max_depth）：设定为-1 到 15，限制树的深度以防止过拟合。
4）每个叶节点的最小样本数（min_data_in_leaf）：范围为 1 到 50，避免小样本叶子节点的出现。
5）特征选择比例（feature_fraction）：在 0.6 到 1.0 之间选择，以防止过拟合。
6）数据采样比例（bagging_fraction）：范围为 0.6 到 1.0，用于控制每次迭代使用的训练样本。
7）数据采样频率（bagging_freq）：设定为 1 到 10，控制采样的频率。

3.5.3 交叉验证过程：

为了评估每组超参数的性能，使用分层 K 折交叉验证（Stratified K-Fold）将数据集划分为训练集和验证集。在每个折叠中，使用 LightGBM 进行训练，并根据预测结果计算 Quadratic Weighted Kappa（QWK）得分，以便评估模型的泛化能力。

3.5.4 优化结果记录：

Optuna 通过最大化 QWK 得分来选择最佳超参数，并记录下最优参数配置。最后，通过多次训练和模型评估，计算训练集和测试集的 QWK 均值，以衡量模型在不同数据集上的表现。(Polipireddy, 2022)

最终训练得到的参数如下表 1，2，3 所示：

表 3 LightGBM 最优参数

Table 3 lightGBM optimal parameter

lightGBM model	
Parameter	Value
Learning_rate	**0.046**
Max_depth	**12**
Num_leaves	**478**
Min_data_in_leaf	**13**
Feature_fraction	**0.893**
Bagging_fraction	**0.784**
Bagging_freq	**4**
Lambda_l1	**10**
Lambda_l2	**0.01**
device	**Cpu**

表 4 XGBoost 最优参数

Table 4 XGBoost optimal parameter

XGBoost model	
Parameter	Value
Learning_rate	**0.05**
Max_depth	**6**
N_estimators	**200**
Subsample	**0.8**
Colsample_bytree	**0.8**
Reg_alpha	**1**
Reg_lambda	**5**
Random_state	**Seed**
Tree_method	**Gpu_hist**

表 5 CatBoost 最优参数

Table 5 CatBoost optimal parameter

CatBoost model	
Parameter	Value
Learning_rate	**0.05**
depth	**6**
Iterations	**200**
Random_seed	**Seed**
verbose	**0**
L2_leaf_reg	**10**
Task_type	**Gpu**

4 模型训练和评估

4.1 评估指标

为了评估分类模型的性能，采用了二次加权 Kappa 系数（Quadratic Weighted Kappa）作为主要评估指标。(Chmura Kraemer H,2002)该指标通过对预测结果与真实标签之间的差异进行量化，提供了一个更加严格的性能衡量，尤其对错误分类的惩罚采用了平方权重。该指标通常从 0（随机一致）到 1（完全一致）不等，如果偶然出现一致性低于预期，该指标可能会低于 0。为了计算二次加权 Kappa，构造了 3 个矩阵：O, W, E，N 为不同标签的数量。

计算公式如下：

$$\kappa = 1 - \frac{\sum_{i,j} W_{i,j} \cdot O_{i,j}}{\sum_{i,j} W_{i,j} \cdot E_{i,j}} \quad (3)$$

其中：O 矩阵是一个 $N \times N$ 的直方图矩阵，$O_{i,j}$ 联系真实值和预测值的数量。W 矩阵是一个 $N \times N$ 的权重矩阵，根据真实值和预测值的平方计算出。E 矩阵是一个 $N \times N$ 的直方矩阵，假设值之间不存在关联性，被计算为结果的实际直方图向量与预测直方图向量之间的外部数据，标准化使得 E，O 具有相同的和。

此外，设计了一个阈值四舍五入函数，将连续的预测结果转换为离散的类别标签。这一过程对于多分类任务至关重要，因为它将模型的输出映射到具体的类别上。最后，通过将真实标签与四舍五入后的预测标签进行比较，计算了 Kappa 评分，并将其负值作为优化目标，以确保模型在实际应用中的准确性和可靠性。

4.2 模型训练与评估

在模型的训练部分，使用一种基于分层 K 折交叉验证的机器学习模型训练方法。具体而言，首先将训练数据集中的特征与目标变量分离。接着，使用分层 K 折交叉验证技术将数据随机分为若干折，以确保每个折中的类别分布与整体数据集一致。对于每一折，克隆模型实例并在训练集上进行拟合，随后在验证集上进行预测。

采用二次加权 Kappa 系数作为模型性能的评估指标，以更严格地惩罚错误分类。通过遍历每个折的训练和验证过程，记录了相应的 Kappa 评分，并将未四舍五入的预测结果存储以便后续分析。为了优化模型预测的阈值，我们利用最小化算法调整预测阈值，进一步提高模型的预测准确性。最后，生成了一个提交数据框，包含了样本 ID 及优化后的预测结果，为后续的模型评估与应用提供了基础。

4.3 结果分析

表 6 所示是融合深度学习和机器学习综合投票模型的训练结果得分，由数据可得模型在训练集上的平均 Quadratic Weighted Kappa（QWK）得分为 0.6865。QWK 是一种专门用于评估回归模型预测准确性的指标，尤其适用于有序类别数据。（姚丽华，2023）得分越接近 1，表示模型对训练数据的拟合能力越强。0.6865 的得分表明模型在训练集上表现良好，能够有效捕捉数据中的潜在模式。相比之下，模型在验证集上的平均 QWK 得分为 0.4445，

显著低于训练集得分。这一结果表明模型在训练集上出现了过拟合现象，即模型在训练数据上表现优秀，但未能有效泛化到未见过的数据。过拟合通常导致模型对新数据的预测能力下降。经过参数调整和模型优化后，验证集的 QWK 得分提升至 0.516。尽管这一优化效果高于初始验证集得分（0.4445），但仍低于训练集得分（0.6865），表明模型的泛化能力仍需进一步改进。优化后的得分显示出一定的改善，但距离理想状态仍存在差距。

表 6 综合模型结果得分

Table 6 Comprehensive model result score

训练集平均 QWK（Mean Train QWK）	验证集平均 QWK（Mean Validation QWK）	优化后 QWK
0.7016	0.4727	**0.536**

最终得到的分类结果如表 7 所示：

表 7 综合模型结果表格

Table 7 Comprehensive model submission1

	id	Sii
0	00008ff9	1
1	00008ff9	0
2	00105258	1
3	00115b9f	0
4	0016bb22	0
5	001f3379	1
6	0038ba98	1
7	0068a485	0
8	0069fbed	1
9	0083e397	0
10	0087dd65	0
11	00abe655	0
12	00ae59c9	1
13	00af6387	1
14	00bd4359	1
15	00c0cd71	1
16	00d56d4b	0
17	00d9913d	1
18	00e6167c	0
19	00ebc35d	1

4.4 对比研究

在研究的最后，为了探索深度学习和机器学习综合模型相比机器学习的优势，使用 Light GBM、XGBoost 、CatBoost 三种机器学习方法放入投票模型的框架中进行结果分析，最终得到的该模型在训练集上的得分为 0.7595，在验证集上的得分为 0.3926，经过优化模型后的得分为 **0.456**。相比于训练集得分下降、验证集上的得分有一定的提升。经过横向对比可以发现，该综合模型存在过拟合的问题，因此与上述得出的结论一样，需要进一步增强综合模型的泛化能力，同时纵向对比可以发现融合 Tabnet 深度学习框架可以一定程度上缓解此问题，因此验证了深度学习、机器学习综合投票模型的性能，通过对比研究也可以对进一步的模型优化有所启示。

5 讨论

通过对结果的分析可以发现模型的泛化能力需要进一步改进，因此可以引入正则化技术或简化模型结构，以减少过拟合风险。同时为了增强整体的分析效果，可以分别从数据清洗、特征提取、模型构建等多方面进行细节优化。例如，在数据预处理的过程中，可以进一步找出每一类型的异常值分别进行处理，如舒张压（60-80 mmHg）和伸

缩压（90-120 mmHg）(赵连友，2008)超出标准范围较多的异常值，可以进行相应的调整或记为 NULL，以做到数据的精细化。在特征提取的过程中，由于数据的维度和噪声相对较高，因此对于缺失值少于 40%的特征用 Lasso 模型进行插补，而其它特征使用均值插补，于是优化策略可以对比例进行超参数调整或经验调参，以便进一步增强数据特征。在模型的训练过程中，可以调整训练集与验证集的比例，增加验证集的样本量，以便更稳定地评估模型性能。

6 结论

本研究通过一系列数据预处理、特征工程与模型构建方法，旨在通过生理/物理数据提高对网瘾程度（sii）的预测准确性。在数据预处理阶段，采用了枚举类型替代字符串存储，显著减少了存储空间并提升了计算效率。同时，使用自编码器进行了数据降维，有效提取了数据中的关键特征。在异常值和缺失值处理方面，设计了基于机器学习的插补策略，确保了数据的完整性和结构关系的保留。通过对生理/物理数据和时序数据的特征提取，生成了丰富的特征集，为后续模型训练提供了坚实基础。模型构建过程中，选择了多种回归模型，包括 TabNet、LGBMRegressor、XGBRegressor 和 CatBoostRegressor，并通过投票回归模型集成不同算法的优势，提升了整体预测性能。最终，模型的评估指标采用了二次加权 Kappa 系数，反映了模型在训练集和验证集上的表现。尽管模型在训练集上表现良好，验证集的较低得分提示了过拟合问题，表明模型的泛化能力仍需进一步提升。未来的研究可以考虑引入正则化技术、优化特征工程参数，以及调整训练与验证集的比例，以增强模型的稳定性和预测准确性。

综上所述，本项目的研究为网瘾程度的量化评估提供了有效的方法论支持，为后续相关研究奠定了基础，同时也为实际应用提供了参考价值。

7 项目说明

使用到的数据有：Child Mind Institude---Problem Internet Use、pytorchtabnet（用于训练自定义 Tabnet 神经网络）
由于数据内存较大，因此将代码、数据、图片存放于百度网盘链接中，地址如下：
https://pan.baidu.com/s/1TRTfFK1evolrAYReLaTF6g?pwd=d62m 提取码: d62m

参考文献(References**)**

[1] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, Tie-Yan Liu. “LightGBM: A Highly Efficient Gradient Boosting Decision Tree.” Advances in Neural Information Processing Systems 30 (NIPS 2017), pp. 3149-3157.

[2] Mehta, Manish, Rakesh Agrawal, and Jorma Rissanen. “SLIQ: A fast scalable classifier for data mining.” International Conference on Extending Database Technology. Springer Berlin Heidelberg, 1996.

[3] Shafer, John, Rakesh Agrawal, and Manish Mehta. “SPRINT: A scalable parallel classifier for data mining.” Proc. 1996 Int. Conf. Very Large Data Bases. 1996.

[4] Akiba, T., Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (pp. 2623–2631). Association for Computing Machinery. https://doi.org/10.1145/3292500.3330701

[5] Polipireddy, S., & Katarya, R. (2022). hyOPTXg: OPTUNA hyper-parameter optimization framework for predicting cardiovascular disease using XGBoost. Biomedical Signal Processing and Control, 73, 103456. https://doi.org/10.1016/j.bspc.2021.103456

[6] 姚丽华, 乔宏 (2023). 比较特征提取方法和机器学习模型在作文自动评分中的表现. *教育测量与评估双语期刊*, 4(3), 文章 2. https://doi.org/10.59863/VLGU9815

[7] Arik, S. Ö., & Pfister, T. (2021). TabNet: Attentive Interpretable Tabular Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, *35*(8), 6679-6687. https://doi.org/10.1609/aaai.v35i8.16826

[8] Omarzai, F. (2024). XGBoost regression in depth. *Medium*. https://medium.com/@fraidoonomarzai99/xgboost-regression-in-depth-cb2b3f623281

[9] Optuna. (n.d.). *Optuna documentation*. Retrieved December 24, 2024, from

https://optuna.readthedocs.io/zh-cn/latest/index.html
[10] 赵连友. 老年人低舒张压的收缩期高血压研究现状[J]. 心脏杂志, 2008 (03).
[11] Chmura Kraemer H, Periyakoil V S, Noda A. Kappa coefficients in medical research[J]. Statistics in medicine, 2002, 21(14): 2109-2129.
[12] 邓验. 青少年网瘾问题的成因及其监控机制研究[J]. 湖南大学学报 (社会科学版), 2014, 28(1): 147-150.
[13] Kuss D J, Shorter G W, van Rooij A J, et al. Assessing internet addiction using the parsimonious internet addiction components model—a preliminary study[J]. International Journal of Mental Health and Addiction, 2014, 12: 351-366.

代码电子附录（Code）

```
import polars as pl
import polars.selectors as cs
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator, FormatStrFormatter, PercentFormatter
import numpy as np
import seaborn as sns
import lightgbm
from colorama import Fore, Style
from scipy.optimize import minimize

from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import cohen_kappa_score, ConfusionMatrixDisplay

target_labels = ['None', 'Mild', 'Moderate', 'Severe']
tmp =  pl.read_csv('../input/child-mind-institute-problematic-internet-use/train.csv')
tmp
```

```
    # 存储优化 转换字符型为枚举型
    season_dtype = pl.Enum(['Spring', 'Summer', 'Fall', 'Winter'])
    train = (
        pl.read_csv('../input/child-mind-institute-problematic-internet-use/train.csv')
        .with_columns(pl.col('^.*Season$').cast(season_dtype))
    )
    test = (
        pl.read_csv('../input/child-mind-institute-problematic-internet-use/test.csv')
        .with_columns(pl.col('^.*Season$').cast(season_dtype))
    )
    train
```

```
    missing_count = (
        train
        .null_count()
        .transpose(include_header=True,
                   header_name='feature',
                   column_names=['null_count'])
        .sort('null_count', descending=True)
        .with_columns((pl.col('null_count') / len(train)).alias('null_ratio'))
```

```
    )
    plt.figure(figsize=(6, 15))
    plt.title('Missing values over the whole training dataset')
    plt.barh(np.arange(len(missing_count)), missing_count.get_column('null_ratio'), color='coral', label='missing')
    plt.barh(np.arange(len(missing_count)),
             1 - missing_count.get_column('null_ratio'),
             left=missing_count.get_column('null_ratio'),
             color='darkseagreen', label='available')
    plt.yticks(np.arange(len(missing_count)), missing_count.get_column('feature'))
    plt.gca().xaxis.set_major_formatter(PercentFormatter(xmax=1, decimals=0))
    plt.xlim(0, 1)
    plt.legend()
    plt.show()
```

```
    supervised_usable = (
        train
        .filter(pl.col('sii').is_not_null())
    )
    # 计算在有效记录中每个特征的缺失值数量
    missing_count = (
        supervised_usable
        .null_count()  # 计算每列的缺失值数量
        .transpose(include_header=True,  # 转置数据，使特征名成为行
                   header_name='feature',  # 指定转置后列的名称
                   column_names=['null_count'])  # 指定新列的名称
        .sort('null_count', descending=True)  # 按缺失值数量降序排序
        .with_columns((pl.col('null_count') / len(supervised_usable)).alias('null_ratio'))  # 计算缺失值比率
    )

    # 绘制缺失值的条形图
    plt.figure(figsize=(6, 15))  # 设置图形大小
    plt.title(f'Missing values over the {len(supervised_usable)} samples which have a target')  # 设置标题
    plt.barh(np.arange(len(missing_count)), missing_count.get_column('null_ratio'), color='coral', label='missing')  # 绘制缺失值的条形
    plt.barh(np.arange(len(missing_count)),
             1 - missing_count.get_column('null_ratio'),
             left=missing_count.get_column('null_ratio'),
             color='darkseagreen', label='available')  # 绘制可用值的条形，左侧为缺失值的条形
    plt.yticks(np.arange(len(missing_count)), missing_count.get_column('feature'))  # 设置 y 轴刻度为特征名
    plt.gca().xaxis.set_major_formatter(PercentFormatter(xmax=1, decimals=0))  # 格式化 x 轴为百分比
    plt.xlim(0, 1)  # 设置 x 轴范围
    plt.legend()  # 显示图例
```

```
    plt.show()  # 显示图形

    # 研究参与者年龄在 5 至 22 岁之间。男孩的数量是女孩的两倍。
    _, axs = plt.subplots(2, 1, sharex=True)
    for sex in range(2):
        ax = axs.ravel()[sex]
        # pl.col 是用于指定列的函数，
        # .get_column('Basic_Demos-Age') 从筛选后的数据中提取 Basic_Demos-Age 列的数据。
        # .value_counts() 对提取的年龄列数据进行计数，返回每个唯一年龄值的出现次数。
        # 计算频率，获取某一列中各个值的分布情况。
        vc = train.filter(pl.col('Basic_Demos-Sex') == sex).get_column('Basic_Demos-Age'
).value_counts()
        ax.bar(vc.get_column('Basic_Demos-Age'),
               vc.get_column('count'),
               color=['lightblue', 'coral'][sex],
               label=['boys', 'girls'][sex])
        ax.xaxis.set_major_locator(MaxNLocator(integer=True))
        ax.set_ylabel('count')
        ax.legend()
    plt.suptitle('Age distribution')
    axs.ravel()[1].set_xlabel('years')
    plt.show()

    vc = train.get_column('Basic_Demos-Enroll_Season').value_counts()
    plt.pie(vc.get_column('count'), labels=vc.get_column('Basic_Demos-Enroll_Season'))
    plt.title('Season of enrollment')
    plt.show()

    _, axs = plt.subplots(2, 1, sharex=True, sharey=True)
    for sex in range(2):
        ax = axs.ravel()[sex]
        vc = train.filter(pl.col('Basic_Demos-Sex') == sex).get_column('sii').value_coun
ts()
        ax.bar(vc.get_column('sii'),
               vc.get_column('count') / vc.get_column('count').sum(),
               color=['lightblue', 'coral'][sex],
               label=['boys', 'girls'][sex])
        ax.set_xticks(np.arange(4), target_labels)
        ax.yaxis.set_major_formatter(PercentFormatter(xmax=1, decimals=0))
        ax.set_ylabel('count')
        ax.legend()
    plt.suptitle('Target distribution')
    axs.ravel()[1].set_xlabel('Severity Impairment Index (sii)')
    plt.show()

    vc = train.get_column('Physical-HeartRate').value_counts()
    color = np.where(vc.get_column('Physical-HeartRate') < 50, 'r', 'b')
    plt.figure(figsize=(8, 2))
```

```
    plt.title('Histogram of Physical-HeartRate with outliers at the low end')
    plt.bar(vc.get_column('Physical-HeartRate'), vc.get_column('count'), color=color)
    plt.xlabel('Physical-HeartRate')
    plt.ylabel('count')
    plt.show()
```

```
    vc = train.get_column('SDS-SDS_Total_Raw').value_counts()
    plt.figure(figsize=(6, 2))
    plt.title('Sleep disturbance scale')
    plt.bar(vc.get_column('SDS-SDS_Total_Raw'), vc.get_column('count'), color='brown')
    plt.xlabel('SDS-SDS_Total_Raw')
    plt.ylabel('count')
    plt.show()
```

```
    actigraphy = pl.read_parquet('../input/child-mind-institute-problematic-internet-use/series_train.parquet/id=0417c91e/part-0.parquet')
    actigraphy
```

```
    def analyze_actigraphy(id, only_one_week=False, small=False):
        actigraphy = pl.read_parquet(f'../input/child-mind-institute-problematic-internet-use/series_train.parquet/id={id}/part-0.parquet')
        # time_of_day 列（以纳秒为单位）转换为天数（通过除以 86400e9）
        day = actigraphy.get_column('relative_date_PCIAT') + actigraphy.get_column('time_of_day') / 86400e9
        # 筛选出 id 列等于变量 id 的行。
        sample = train.filter(pl.col('id') == id)
        # 筛选后的样本中获取年龄
        age = sample.get_column('Basic_Demos-Age').item()
        # 使用 item() 方法获取性别的索引（通常是 0 或 1），然后从 ['boy', 'girl'] 列表中取出对应的性别字符串。
        sex = ['boy', 'girl'][sample.get_column('Basic_Demos-Sex').item()]
        # 在 actigraphy 中添加两列：diff_seconds 和 norm。
        # day.diff() * 86400 计算 day 列的差异（即前后行之间的时间差），并将结果转换为秒数。
        actigraphy = (
            actigraphy
            .with_columns(
                (day.diff() * 86400).alias('diff_seconds'),
                # 计算三维空间中 X、Y、Z 的范数（即向量长度），结果命名为 norm。
                (np.sqrt(np.square(pl.col('X')) + np.square(pl.col('Y')) + np.square(pl.col('Z')))).alias('norm'))
            )
        )

        if only_one_week:
            # 获取 day 列的最小值，并向上取整，赋值给 start。这通常用于确定一周的开始日期。
            start = np.ceil(day.min())
            # 创建一个布尔掩码，筛选出在 start 到 start + 21（即 3 周）之间的日期。
            mask = (start <= day.to_numpy()) & (day.to_numpy() <= start + 7*3)
```

```
            # 进一步修改掩码，排除那些 non-wear_flag 为真的行（即不穿戴状态），将该列转换为布尔类型。
            mask &= ~ actigraphy.get_column('non-wear_flag').cast(bool).to_numpy()
        else:
            # 创建一个全为 True 的掩码，长度与 day 相同，这意味着不对数据进行任何筛选。
            mask = np.full(len(day), True)

        if small:
            timelines = [
                ('enmo', 'forestgreen'),
                ('light', 'orange'),
            ]
        else:
            timelines = [
                ('X', 'm'), #  表示三个方向的加速度数据，
                ('Y', 'm'),
                ('Z', 'm'),
    #             ('norm', 'c'),
                ('enmo', 'forestgreen'),  #表示 ENMO（加速度数据的一种处理方式）
                ('anglez', 'lightblue'),  # 表示 Z 轴角度数据
                ('light', 'orange'), # 表示光强数据
                ('non-wear_flag', 'chocolate') # 表示不穿戴状态，
        #         ('diff_seconds', 'k'),
            ]

        _, axs = plt.subplots(len(timelines), 1, sharex=True, figsize=(12, len(timelines) * 1.1 + 0.5))
        for ax, (feature, color) in zip(axs, timelines):
            ax.set_facecolor('#eeeeee')
            ax.scatter(day.to_numpy()[mask],
                       actigraphy.get_column(feature).to_numpy()[mask],
                       color=color, label=feature, s=1)
            ax.legend(loc='upper left', facecolor='#eeeeee')
            if feature == 'diff_seconds':
                ax.set_ylim(-0.5, 20.5)
        axs[-1].set_xlabel('day')
        axs[-1].xaxis.set_major_locator(MaxNLocator(integer=True))
        plt.tight_layout()
        axs[0].set_title(f'id={id}, {sex}, age={age}')
        plt.show()

    analyze_actigraphy('0417c91e', only_one_week=False)
```

```
    analyze_actigraphy('22375702', small=True)
```

```
    def feature_engineering(df):
        season_cols = [col for col in df.columns if 'Season' in col]
        df = df.drop(season_cols, axis=1)                                    # 删除季节列
```

```
        # 创建新特征
        df['BMI_Age'] = df['Physical-BMI'] * df['Basic_Demos-Age']
              #BMI 和年龄的乘积
        df['Internet_Hours_Age'] = df['PreInt_EduHx-computerinternet_hoursday'] * df['Basic_Demos-Age'] #上网时长和年龄的乘积
        df['BMI_Internet_Hours'] = df['Physical-BMI'] * df['PreInt_EduHx-computerinternet_hoursday']    #BMI 和上网时长的乘积
        df['BFP_BMI'] = df['BIA-BIA_Fat'] / df['BIA-BIA_BMI']
              #体脂和 BMI 的比值
        df['FFMI_BFP'] = df['BIA-BIA_FFMI'] / df['BIA-BIA_Fat']
              #无脂体重和体脂的比值
        df['FMI_BFP'] = df['BIA-BIA_FMI'] / df['BIA-BIA_Fat']
              #脂肪质量和体脂的比值
        df['LST_TBW'] = df['BIA-BIA_LST'] / df['BIA-BIA_TBW']
              #瘦体重和总水分的比值
        df['BFP_BMR'] = df['BIA-BIA_Fat'] * df['BIA-BIA_BMR']
              #体脂和基础代谢率的乘积
        df['BFP_DEE'] = df['BIA-BIA_Fat'] * df['BIA-BIA_DEE']
              #体脂和日常消耗的乘积
        df['BMR_Weight'] = df['BIA-BIA_BMR'] / df['Physical-Weight']
              #基础代谢率和体重的比值
        df['DEE_Weight'] = df['BIA-BIA_DEE'] / df['Physical-Weight']
              #日常消耗和体重的比值
        df['SMM_Height'] = df['BIA-BIA_SMM'] / df['Physical-Height']
              #肌肉量与身高的比值
        df['Muscle_to_Fat'] = df['BIA-BIA_SMM'] / df['BIA-BIA_FMI']
              #肌肉与脂肪的比值
        df['Hydration_Status'] = df['BIA-BIA_TBW'] / df['Physical-Weight']
              #水合状态与体重的比值
        df['ICW_TBW'] = df['BIA-BIA_ICW'] / df['BIA-BIA_TBW']
              #胞内水分和总体水分的比值
        df['BMI_PHR'] = df['Physical-BMI'] * df['Physical-HeartRate']
              #BMI 和心率的乘积

        return df
```

```
    from pytorch_tabnet.tab_model import TabNetRegressor
    import torch
    import numpy as np
    import pandas as pd
    import os
    import re
    from sklearn.base import clone
    from sklearn.metrics import cohen_kappa_score
    from sklearn.model_selection import StratifiedKFold
    from scipy.optimize import minimize
    from concurrent.futures import ThreadPoolExecutor
    from tqdm import tqdm
```

```
import polars as pl
import polars.selectors as cs
import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator, FormatStrFormatter, PercentFormatter
import seaborn as sns

from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from keras.models import Model
from keras.layers import Input, Dense
from keras.optimizers import Adam
import torch
import torch.nn as nn
import torch.optim as optim

from colorama import Fore, Style
from IPython.display import clear_output
import warnings
from lightgbm import LGBMRegressor
from xgboost import XGBRegressor
from catboost import CatBoostRegressor
from sklearn.ensemble import VotingRegressor, RandomForestRegressor, GradientBoostingRegressor
from sklearn.impute import SimpleImputer, KNNImputer
from sklearn.pipeline import Pipeline
warnings.filterwarnings('ignore')
pd.options.display.max_columns = None
```

```
import random
def seed_everything(seed):
    random.seed(seed)              #使用 NumPy 设置随机种子，以确保NumPy 生成的随机数是可重复的。
    os.environ['PYTHONHASHSEED'] = str(seed) #将 PYTHONHASHSEED 环境变量设置为指定的种子。这有助于确保哈希函数的输出是可预测的。
    np.random.seed(seed)
    torch.manual_seed(seed)        #为CPU 设置随机种子（如果可用）
    torch.cuda.manual_seed(seed)   #为GPU 设置随机种子（如果可用）
    torch.backends.cudnn.deterministic = True  #保证每次运行时，卷积操作的结果是确定的。
    torch.backends.cudnn.benchmark = True      #启用基准模式，可能会影响性能，但在某些情况下会使结果更一致。
seed_everything(2024)              #传入 2024 作为种子值。这将保证在运行后续代码时，所有随机数生成器的行为都是可预测的。
```

```
SEED = 42
n_splits = 5
```

```
def process_file(filename, dirname):
    # 删去 step 列，重塑一维数据，返回文件名中等号后面的部分作为索引
```

```
        df = pd.read_parquet(os.path.join(dirname, filename, 'part-0.parquet'))
        df.drop('step', axis=1, inplace=True)
        return df.describe().values.reshape(-1), filename.split('=')[1]

    def load_time_series(dirname) -> pd.DataFrame:
        # 并行处理文件
        ids = os.listdir(dirname)
        with ThreadPoolExecutor() as executor:
            results = list(tqdm(executor.map(lambda fname: process_file(fname, dirname), ids), total=len(ids)))

        stats, indexes = zip(*results)

        df = pd.DataFrame(stats, columns=[f"stat_{i}" for i in range(len(stats[0]))])
        df['id'] = indexes
        return df
```

```
    import torch
    import torch.nn as nn
    import torch.optim as optim
    import pandas as pd
    from sklearn.preprocessing import StandardScaler


    class AutoEncoder(nn.Module):
        def __init__(self, input_dim, encoding_dim):
            super(AutoEncoder, self).__init__()
            self.encoder = nn.Sequential(                   #编码器 多个线性层和relu 激活函数
                nn.Linear(input_dim, encoding_dim*3),
                nn.ReLU(),
                nn.Linear(encoding_dim*3, encoding_dim*2),
                nn.ReLU(),
                nn.Linear(encoding_dim*2, encoding_dim),
                nn.ReLU()
            )
            self.decoder = nn.Sequential(                   #解码器 恢复原始数据
                nn.Linear(encoding_dim, input_dim*2),
                nn.ReLU(),
                nn.Linear(input_dim*2, input_dim*3),
                nn.ReLU(),
                nn.Linear(input_dim*3, input_dim),
                nn.Sigmoid()
            )

        def forward(self, x):                               #前向传播
            encoded = self.encoder(x)
            decoded = self.decoder(encoded)
```

```
            return decoded


    def perform_autoencoder(df, encoding_dim=50, epochs=50, batch_size=32):
        scaler = StandardScaler()                    # 数据标准化--均值为0，方差为1
        df_scaled = scaler.fit_transform(df)

        data_tensor = torch.FloatTensor(df_scaled)  # 标准化后的数据转换成 FloatTensor 格式

        input_dim = data_tensor.shape[1]
        autoencoder = AutoEncoder(input_dim, encoding_dim)   # 创建 AutoEncoder 实例，输入维度和编码维度作为参数

        criterion = nn.MSELoss()                                # 使用 MSE（均方误差）作为损失函数
        optimizer = optim.Adam(autoencoder.parameters())    # 使用 Adam 优化器更新模型参数

        for epoch in range(epochs):                             #对每个 epoch 进行循环，数据分批次处理，清零梯度
            for i in range(0, len(data_tensor), batch_size):  #进行前向传播，计算重构与原始输入的损失
                batch = data_tensor[i : i + batch_size]
                optimizer.zero_grad()
                reconstructed = autoencoder(batch)
                loss = criterion(reconstructed, batch)
                loss.backward()                                 #反向传播并更新参数
                optimizer.step()

            if (epoch + 1) % 10 == 0:                           #每 10 个 epoch 打印一次损失之以监控训练过程
                print(f'Epoch [{epoch + 1}/{epochs}], Loss: {loss.item():.4f}]')

        with torch.no_grad():                           # 禁用梯度计算，得到编码后的数据
            encoded_data = autoencoder.encoder(data_tensor).numpy()

        df_encoded = pd.DataFrame(encoded_data, columns=[f'Enc_{i + 1}' for i in range(encoded_data.shape[1])])

        return df_encoded


import pandas as pd
import numpy as np
from sklearn.linear_model import LassoCV
from sklearn.base import clone
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from pytorch_tabnet.callbacks import Callback
import os
```

```
    import torch
    from tqdm import tqdm
    # 加载数据集
    train = pd.read_csv('/kaggle/input/child-mind-institute-problematic-internet-use/train.csv')
    test = pd.read_csv('/kaggle/input/child-mind-institute-problematic-internet-use/test.csv')
    sample = pd.read_csv('/kaggle/input/child-mind-institute-problematic-internet-use/sample_submission.csv')

    # 加载时间序列数据
    train_ts = load_time_series("/kaggle/input/child-mind-institute-problematic-internet-use/series_train.parquet")
    test_ts = load_time_series("/kaggle/input/child-mind-institute-problematic-internet-use/series_test.parquet")

    # 从时间序列数据中移除 'id' 列
    df_train = train_ts.drop('id', axis=1)
    df_test = test_ts.drop('id', axis=1)

    # 使用自编码器对特征进行编码
    train_ts_encoded = perform_autoencoder(df_train, encoding_dim=60, epochs=100, batch_size=32)
    test_ts_encoded = perform_autoencoder(df_test, encoding_dim=60, epochs=100, batch_size=32)

    # 将 'id' 列重新添加到编码后的数据框中
    train_ts_encoded["id"] = train_ts["id"]
    test_ts_encoded['id'] = test_ts["id"]

    # 将编码后的特征与原始训练和测试数据集合并
    train = pd.merge(train, train_ts_encoded, how="left", on='id')
    test = pd.merge(test, test_ts_encoded, how="left", on='id')


    # 定义插补类
    class Impute_With_Model:
        def __init__(self, na_frac=0.5, min_samples=0):
            self.model_dict = {}  # 存储插补模型
            self.mean_dict = {}   # 存储特征均值
            self.features = None
            self.na_frac = na_frac
            self.min_samples = min_samples

        def find_features(self, data, feature, tmp_features):
            # 查找可用于插补的有效特征
            missing_rows = data[feature].isna()
            na_fraction = data[missing_rows][tmp_features].isna().mean(axis=0)
```

```
            valid_features = np.array(tmp_features)[na_fraction <= self.na_frac]
            return valid_features

        def fit_models(self, model, data, features):
            # 为每个缺失值特征拟合模型
            self.features = features
            for feature in features:
                self.mean_dict[feature] = np.mean(data[feature])
            for feature in tqdm(features):
                if data[feature].isna().sum() > 0:
                    model_clone = clone(model)
                    X = data[data[feature].notna()].copy()
                    tmp_features = [f for f in features if f != feature]
                    tmp_features = self.find_features(data, feature, tmp_features)
                    if len(tmp_features) >= 1 and X.shape[0] > self.min_samples:
                        for f in tmp_features:
                            X[f] = X[f].fillna(self.mean_dict[f])
                        model_clone.fit(X[tmp_features], X[feature])
                        self.model_dict[feature] = (model_clone, tmp_features.copy())
                    else:
                        self.model_dict[feature] = ("mean", np.mean(data[feature]))

        def impute(self, data):
            # 插补缺失值
            imputed_data = data.copy()
            for feature, model in self.model_dict.items():
                missing_rows = imputed_data[feature].isna()
                if missing_rows.any():
                    if model[0] == "mean":
                        imputed_data[feature].fillna(model[1], inplace=True)
                    else:
                        tmp_features = [f for f in self.features if f != feature]
                        X_missing = data.loc[missing_rows, tmp_features].copy()
                        for f in tmp_features:
                            X_missing[f] = X_missing[f].fillna(self.mean_dict[f])
                        imputed_data.loc[missing_rows, feature] = model[0].predict(X_mis
sing[model[1]])
            return imputed_data
```

```
    train = pd.read_csv('/kaggle/input/child-mind-institute-problematic-internet-use/tra
in.csv')
    test = pd.read_csv('/kaggle/input/child-mind-institute-problematic-internet-use/test
.csv')
    sample = pd.read_csv('/kaggle/input/child-mind-institute-problematic-internet-use/sa
mple_submission.csv')
    # 加载时间序列数据
    train_ts = load_time_series("/kaggle/input/child-mind-institute-problematic-internet
-use/series_train.parquet")
```

```
    test_ts = load_time_series("/kaggle/input/child-mind-institute-problematic-internet-use/series_test.parquet")
    # 时间序列中删除 id 列，分别存储为 df_train 和 df_test
    df_train = train_ts.drop('id', axis=1)
    df_test = test_ts.drop('id', axis=1)
    # 对训练集和测试集用 autoencoder 生成编码后的特征
    train_ts_encoded = perform_autoencoder(df_train, encoding_dim=60, epochs=100, batch_size=32)
    test_ts_encoded = perform_autoencoder(df_test, encoding_dim=60, epochs=100, batch_size=32)
    # 获取编码后的列表名称并将 id 添加回编码后的数据框中
    time_series_cols = train_ts_encoded.columns.tolist()
    train_ts_encoded["id"]=train_ts["id"]
    test_ts_encoded['id']=test_ts["id"]
    # 编码后的特征与原训练集和测试集按 id 列进行左连接，更新 train 和 test 数据框
    train = pd.merge(train, train_ts_encoded, how="left", on='id')
    test = pd.merge(test, test_ts_encoded, how="left", on='id')
    # 初始化 KNN 填充器 邻居设为5，选择训练集中所有数值型
    imputer = KNNImputer(n_neighbors=5)
    numeric_cols = train.select_dtypes(include=['float64', 'int64']).columns
    imputed_data = imputer.fit_transform(train[numeric_cols])
    train_imputed = pd.DataFrame(imputed_data, columns=numeric_cols)      # 用 KNN 对数值型列进行缺失值填充
    train_imputed['sii'] = train_imputed['sii'].round().astype(int)       #将 sii 列的值四舍五入并转换为整数类型
    for col in train.columns:                 # 原训练集中的非数值型列添加到填充后的数据框中
        if col not in numeric_cols:
            train_imputed[col] = train[col]
    # 更新 train 数据
    train = train_imputed
    # 分别对训练集和测试集应用特征工程  生成新特征。从训练集中删除缺少 10 个及以上非缺失值的行
    train = feature_engineering(train)
    train = train.dropna(thresh=10, axis=0)
    test = feature_engineering(test)
    # 删除 id 列进行模型训练
    train = train.drop('id', axis=1)
    test  = test .drop('id', axis=1)

    # 特征列的 列表，包含模型训练所需的所有特征
    featuresCols = ['Basic_Demos-Age', 'Basic_Demos-Sex',
                    'CGAS-CGAS_Score', 'Physical-BMI',
                    'Physical-Height', 'Physical-Weight', 'Physical-Waist_Circumference',
                    'Physical-Diastolic_BP', 'Physical-HeartRate', 'Physical-Systolic_BP',
                    'Fitness_Endurance-Max_Stage',
                    'Fitness_Endurance-Time_Mins', 'Fitness_Endurance-Time_Sec',
                    'FGC-FGC_CU', 'FGC-FGC_CU_Zone', 'FGC-FGC_GSND',
```

```
                    'FGC-FGC_GSND_Zone', 'FGC-FGC_GSD', 'FGC-FGC_GSD_Zone', 'FGC-FGC_PU'
,
                    'FGC-FGC_PU_Zone', 'FGC-FGC_SRL', 'FGC-FGC_SRL_Zone', 'FGC-FGC_SRR',
                    'FGC-FGC_SRR_Zone', 'FGC-FGC_TL', 'FGC-FGC_TL_Zone',
                    'BIA-BIA_Activity_Level_num', 'BIA-BIA_BMC', 'BIA-BIA_BMI',
                    'BIA-BIA_BMR', 'BIA-BIA_DEE', 'BIA-BIA_ECW', 'BIA-BIA_FFM',
                    'BIA-BIA_FFMI', 'BIA-BIA_FMI', 'BIA-BIA_Fat', 'BIA-BIA_Frame_num',
                    'BIA-BIA_ICW', 'BIA-BIA_LDM', 'BIA-BIA_LST', 'BIA-BIA_SMM',
                    'BIA-BIA_TBW', 'PAQ_A-PAQ_A_Total',
                    'PAQ_C-PAQ_C_Total', 'SDS-SDS_Total_Raw',
                    'SDS-SDS_Total_T',
                    'PreInt_EduHx-computerinternet_hoursday', 'sii', 'BMI_Age','Internet
_Hours_Age','BMI_Internet_Hours',
                    'BFP_BMI', 'FFMI_BFP', 'FMI_BFP', 'LST_TBW', 'BFP_BMR', 'BFP_DEE', '
BMR_Weight', 'DEE_Weight',
                    'SMM_Height', 'Muscle_to_Fat', 'Hydration_Status', 'ICW_TBW','BMI_PH
R']
    # 时间序列编码后的列名添加到新特征列列表中
    featuresCols += time_series_cols
    # 从训练集中选择特征列，删除 sii 列中缺失值的行
    train = train[featuresCols]
    train = train.dropna(subset='sii')
    # 新的特征列列表
    featuresCols = ['Basic_Demos-Age', 'Basic_Demos-Sex',
                    'CGAS-CGAS_Score', 'Physical-BMI',
                    'Physical-Height', 'Physical-Weight', 'Physical-Waist_Circumference'
,
                    'Physical-Diastolic_BP', 'Physical-HeartRate', 'Physical-Systolic_BP
',
                    'Fitness_Endurance-Max_Stage',
                    'Fitness_Endurance-Time_Mins', 'Fitness_Endurance-Time_Sec',
                    'FGC-FGC_CU', 'FGC-FGC_CU_Zone', 'FGC-FGC_GSND',
                    'FGC-FGC_GSND_Zone', 'FGC-FGC_GSD', 'FGC-FGC_GSD_Zone', 'FGC-FGC_PU'
,
                    'FGC-FGC_PU_Zone', 'FGC-FGC_SRL', 'FGC-FGC_SRL_Zone', 'FGC-FGC_SRR',
                    'FGC-FGC_SRR_Zone', 'FGC-FGC_TL', 'FGC-FGC_TL_Zone',
                    'BIA-BIA_Activity_Level_num', 'BIA-BIA_BMC', 'BIA-BIA_BMI',
                    'BIA-BIA_BMR', 'BIA-BIA_DEE', 'BIA-BIA_ECW', 'BIA-BIA_FFM',
                    'BIA-BIA_FFMI', 'BIA-BIA_FMI', 'BIA-BIA_Fat', 'BIA-BIA_Frame_num',
                    'BIA-BIA_ICW', 'BIA-BIA_LDM', 'BIA-BIA_LST', 'BIA-BIA_SMM',
                    'BIA-BIA_TBW', 'PAQ_A-PAQ_A_Total',
                    'PAQ_C-PAQ_C_Total', 'SDS-SDS_Total_Raw',
                    'SDS-SDS_Total_T',
                    'PreInt_EduHx-computerinternet_hoursday', 'BMI_Age','Internet_Hours_
Age','BMI_Internet_Hours',
                    'BFP_BMI', 'FFMI_BFP', 'FMI_BFP', 'LST_TBW', 'BFP_BMR', 'BFP_DEE', '
BMR_Weight', 'DEE_Weight',
```

```
                  'SMM_Height', 'Muscle_to_Fat', 'Hydration_Status', 'ICW_TBW','BMI_PHR']
    #时间序列编码后的列名添加到测试集特征列中，从测试集中选择这些特征。
    featuresCols += time_series_cols
    test = test[featuresCols]


    #检查train 中是否有 inf/-inf 的值，如果存在就转化成 NAN
    if np.any(np.isinf(train)):
        train = train.replace([np.inf, -np.inf], np.nan)
    #定义一个函数计算 K 系数，用于评估分类模型性能
    def quadratic_weighted_kappa(y_true, y_pred):
        return cohen_kappa_score(y_true, y_pred, weights='quadratic')  # 权重定义为 quadratic:意味错误分类的惩罚是平方级别的
    #用于将非整型预测结果根据给定的阈值进行四舍五入
    def threshold_Rounder(oof_non_rounded, thresholds):
        return np.where(oof_non_rounded < thresholds[0], 0,
                        np.where(oof_non_rounded < thresholds[1], 1,
                                 np.where(oof_non_rounded < thresholds[2], 2, 3)))
    #评估模型预测的质量（阈值，真实标签，四舍五入的预测值）
    def evaluate_predictions(thresholds, y_true, oof_non_rounded):
        rounded_p = threshold_Rounder(oof_non_rounded, thresholds)   #将四舍五入的值转换为整数标签
        return -quadratic_weighted_kappa(y_true, rounded_p)         #计算真实标签和四舍五入后的预测标签之间的 Kappa 评分，并返回其负值，以便在优化时最小化该评分


    # 用于训练机器学习模型的函数 TrainML,并对模型进行交叉验证和预测
    def TrainML(model_class, test_data):   # 两个参数：模型类+测试数据
        X = train.drop(['sii'], axis=1)    # 特征集
        y = train['sii']                   #目标变量
    # 初始化分层 K 折交叉验证对象 StratifiedFold，将数据划分为 n_splits 份，并随机打乱
        SKF = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=SEED)
    # 创建两个空列表 用于存储每折的 Kappa 评分
        train_S = []
        test_S = []
    # 初始化三个数组
        oof_non_rounded = np.zeros(len(y), dtype=float)     #存储每个折的验证集预测值（未四舍五入）
        oof_rounded = np.zeros(len(y), dtype=int)           #存储每个折的验证集预测值（四舍五入）
        test_preds = np.zeros((len(test_data), n_splits))  #存储每个折的测试数据的预测结果
    #使用 tqdm 显示训练折的进度 遍历每个折的索引，train_idx 和 test_idx 是索引
        for fold, (train_idx, test_idx) in enumerate(tqdm(SKF.split(X, y), desc="Training Folds", total=n_splits)):
            X_train, X_val = X.iloc[train_idx], X.iloc[test_idx]
            y_train, y_val = y.iloc[train_idx], y.iloc[test_idx]
    # 克隆实例，并用训练集数据拟合模型
            model = clone(model_class)
            model.fit(X_train, y_train)
```

```
    # 对训练集和验证集进行预测，分别存储结果
            y_train_pred = model.predict(X_train)
            y_val_pred = model.predict(X_val)
    # 将验证集的未四舍五入预测值存入 oof_non_rounded。
    # 对验证集的预测结果进行四舍五入并转换为整数，存入 oof_rounded。
            oof_non_rounded[test_idx] = y_val_pred
            y_val_pred_rounded = y_val_pred.round(0).astype(int)
            oof_rounded[test_idx] = y_val_pred_rounded
    # 将训练和验证的 Kappa 评分添加到各自的列表中。
            train_kappa = quadratic_weighted_kappa(y_train, y_train_pred.round(0).astype
(int))
            val_kappa = quadratic_weighted_kappa(y_val, y_val_pred_rounded)
    # 计算训练集和验证集的二次加权 Kappa 评分
            train_S.append(train_kappa)
            test_S.append(val_kappa)

            test_preds[:, fold] = model.predict(test_data)
    # 对测试数据进行预测，并将结果存入 test_preds 的当前折
            print(f"Fold {fold+1} - Train QWK: {train_kappa:.4f}, Validation QWK: {val_k
appa:.4f}")
            clear_output(wait=True)

        print(f"Mean Train QWK --> {np.mean(train_S):.4f}")
        print(f"Mean Validation QWK ---> {np.mean(test_S):.4f}")
    #使用 minimize 函数来优化预测阈值，初始值为 [0.5, 1.5, 2.5]，目标是最小
化 evaluate_predictions 函数的返回值。
        KappaOPtimizer = minimize(evaluate_predictions,
                                  x0=[0.5, 1.5, 2.5], args=(y, oof_non_rounded),
                                  method='Nelder-Mead')
        assert KappaOPtimizer.success, "Optimization did not converge."
    #检查优化是否成功，如果不成功，则抛出异常。
    #使用优化得到的阈值对未四舍五入的预测值进行四舍五入，得到调整后的预测结果。
        oof_tuned = threshold_Rounder(oof_non_rounded, KappaOPtimizer.x)
    #计算优化后的 Kappa 评分。
        tKappa = quadratic_weighted_kappa(y, oof_tuned)

        print(f"----> || Optimized QWK SCORE :: {Fore.CYAN}{Style.BRIGHT} {tKappa:.3f}{S
tyle.RESET_ALL}")
    #对测试预测结果取平均，得到 tpm，并使用优化后的阈值进行四舍五入，得到调整后的测试预测
值 tpTuned。
        tpm = test_preds.mean(axis=1)
        tpTuned = threshold_Rounder(tpm, KappaOPtimizer.x)
    #创建一个提交数据框 submission，包含样本 ID 和优化后的预测值。
        submission = pd.DataFrame({
            'id': sample['id'],
            'sii': tpTuned
        })

```

```
        return submission

from sklearn.base import BaseEstimator, RegressorMixin
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from pytorch_tabnet.callbacks import Callback
import os
import torch
from pytorch_tabnet.callbacks import Callback

class TabNetWrapper(BaseEstimator, RegressorMixin):
    def __init__(self, **kwargs):
        self.model = TabNetRegressor(**kwargs)
        self.kwargs = kwargs
        self.imputer = SimpleImputer(strategy='median')
        self.best_model_path = 'best_tabnet_model.pt'

    def fit(self, X, y):
        # Handle missing values
        X_imputed = self.imputer.fit_transform(X)

        if hasattr(y, 'values'):
            y = y.values

        # Create internal validation set
        X_train, X_valid, y_train, y_valid = train_test_split(
            X_imputed,
            y,
            test_size=0.2,
            random_state=42
        )

        # Train TabNet model
        history = self.model.fit(
            X_train=X_train,
            y_train=y_train.reshape(-1, 1),
            eval_set=[(X_valid, y_valid.reshape(-1, 1))],
            eval_name=['valid'],
            eval_metric=['mse'],
            max_epochs=200,
            patience=20,
            batch_size=1024,
            virtual_batch_size=128,
            num_workers=0,
            drop_last=False,
            callbacks=[
                TabNetPretrainedModelCheckpoint(
                    filepath=self.best_model_path,
```

```
                        monitor='valid_mse',
                        mode='min',
                        save_best_only=True,
                        verbose=True
                    )
                ]
            )

            # Load the best model
            if os.path.exists(self.best_model_path):
                self.model.load_model(self.best_model_path)
                os.remove(self.best_model_path)  # Remove temporary file

            # Reflect feature importances to the wrapper class
            self.feature_importances_ = self.model.feature_importances_

            return self

        def predict(self, X):
            X_imputed = self.imputer.transform(X)
            return self.model.predict(X_imputed).flatten()

        def __deepcopy__(self, memo):
            # Add deepcopy support for scikit-learn
            cls = self.__class__
            result = cls.__new__(cls)
            memo[id(self)] = result
            for k, v in self.__dict__.items():
                setattr(result, k, deepcopy(v, memo))
            return result

    # TabNet hyperparameters
    TabNet_Params = {
        'n_d': 64,              # Width of the decision prediction layer
        'n_a': 64,              # Width of the attention embedding for each step
        'n_steps': 5,           # Number of steps in the architecture
        'gamma': 1.5,           # Coefficient for feature selection regularization
        'n_independent': 2,     # Number of independent GLU layers in each GLU block
        'n_shared': 2,          # Number of shared GLU layers in each GLU block
        'lambda_sparse': 1e-4,  # Sparsity regularization
        'optimizer_fn': torch.optim.Adam,
        'optimizer_params': dict(lr=2e-2, weight_decay=1e-5),
        'mask_type': 'entmax',
        'scheduler_params': dict(mode="min", patience=10, min_lr=1e-5, factor=0.5),
        'scheduler_fn': torch.optim.lr_scheduler.ReduceLROnPlateau,
        'verbose': 1,
        'device_name': 'cuda' if torch.cuda.is_available() else 'cpu'
    }
```

```
87.
88.    class TabNetPretrainedModelCheckpoint(Callback):
89.        def __init__(self, filepath, monitor='val_loss', mode='min',
90.                     save_best_only=True, verbose=1):
91.            super().__init__()  # Initialize parent class
92.            self.filepath = filepath
93.            self.monitor = monitor
94.            self.mode = mode
95.            self.save_best_only = save_best_only
96.            self.verbose = verbose
97.            self.best = float('inf') if mode == 'min' else -float('inf')
98.
99.        def on_train_begin(self, logs=None):
100.           self.model = self.trainer  # Use trainer itself as model
101.
102.       def on_epoch_end(self, epoch, logs=None):
103.           logs = logs or {}
104.           current = logs.get(self.monitor)
105.           if current is None:
106.               return
107.
108.           # Check if current metric is better than best
109.           if (self.mode == 'min' and current < self.best) or \
110.              (self.mode == 'max' and current > self.best):
111.               if self.verbose:
112.                   print(f'\nEpoch {epoch}: {self.monitor} improved from {self.best:.4f
} to {current:.4f}')
113.               self.best = current
114.               if self.save_best_only:
115.                   self.model.save_model(self.filepath)  # Save the entire model

1.    # Model parameters for LightGBM
2.    Params = {
3.        'learning_rate': 0.046,
4.        'max_depth': 12,
5.        'num_leaves': 478,
6.        'min_data_in_leaf': 13,
7.        'feature_fraction': 0.893,
8.        'bagging_fraction': 0.784,
9.        'bagging_freq': 4,
10.       'lambda_l1': 10,  # Increased from 6.59
11.       'lambda_l2': 0.01,  # Increased from 2.68e-06
12.       'device': 'cpu'
13.
14.   }
15.
16.
17.   # XGBoost parameters
```

```
    XGB_Params = {
        'learning_rate': 0.05,
        'max_depth': 6,
        'n_estimators': 200,
        'subsample': 0.8,
        'colsample_bytree': 0.8,
        'reg_alpha': 1,  # Increased from 0.1
        'reg_lambda': 5,  # Increased from 1
        'random_state': SEED,
        'tree_method': 'gpu_hist',

    }


    CatBoost_Params = {
        'learning_rate': 0.05,
        'depth': 6,
        'iterations': 200,
        'random_seed': SEED,
        'verbose': 0,
        'l2_leaf_reg': 10,  # Increase this value
        'task_type': 'GPU'

    }
```

```
    # 创建模型实例（回归模型）
    Light = LGBMRegressor(**Params, random_state=SEED, verbose=-1, n_estimators=300)
    # **Params: 使用字典 Params 中定义的超参数，** 操作符将字典中的键值对作为参数传递给构造函数。
    # random_state=SEED: 设置随机种子以确保结果可重复。SEED 是预定义的随机数种子值。
    # verbose=-1: 设置模型训练的输出信息级别。-1 表示不输出任何信息。
    # n_estimators=300: 设置模型中的树的数量为 300。

    XGB_Model = XGBRegressor(**XGB_Params)
    # **XGB_Params: 使用字典 XGB_Params 中定义的超参数，** 操作符将字典中的键值对作为参数传递给构造函数。

    CatBoost_Model = CatBoostRegressor(**CatBoost_Params)
    #**CatBoost_Params: 使用字典 CatBoost_Params 中定义的超参数，** 操作符将字典中的键值对作为参数传递给构造函数
    #TabNet_Model = TabNetWrapper(**TabNet_Params) # New
    #**TabNet_Params: 使用字典 TabNet_Params 中定义的超参数,** 操作符将字典中的键值对作为参数传递给构造函数。
    # TabNet
    TabNet_Model = TabNetWrapper(**TabNet_Params) # New

    voting_model = VotingRegressor(estimators=[
        ('lightgbm', Light),
```

```
        ('xgboost', XGB_Model),
        ('catboost', CatBoost_Model),
        ('tabnet', TabNet_Model)
    ],weights=[4.0,3.0,5.0,5.0])

    Submission = TrainML(voting_model, test)
    Submission
```

```
    train = pd.read_csv('/kaggle/input/child-mind-institute-problematic-internet-use/train.csv')
    test = pd.read_csv('/kaggle/input/child-mind-institute-problematic-internet-use/test.csv')
    sample = pd.read_csv('/kaggle/input/child-mind-institute-problematic-internet-use/sample_submission.csv')

    def process_file(filename, dirname):
        df = pd.read_parquet(os.path.join(dirname, filename, 'part-0.parquet'))
        df.drop('step', axis=1, inplace=True)
        return df.describe().values.reshape(-1), filename.split('=')[1]

    def load_time_series(dirname) -> pd.DataFrame:
        ids = os.listdir(dirname)

        with ThreadPoolExecutor() as executor:
            results = list(tqdm(executor.map(lambda fname: process_file(fname, dirname), ids), total=len(ids)))

        stats, indexes = zip(*results)

        df = pd.DataFrame(stats, columns=[f"stat_{i}" for i in range(len(stats[0]))])
        df['id'] = indexes
        return df

    train_ts = load_time_series("/kaggle/input/child-mind-institute-problematic-internet-use/series_train.parquet")
    test_ts = load_time_series("/kaggle/input/child-mind-institute-problematic-internet-use/series_test.parquet")

    time_series_cols = train_ts.columns.tolist()
    time_series_cols.remove("id")

    train = pd.merge(train, train_ts, how="left", on='id')
    test = pd.merge(test, test_ts, how="left", on='id')

    train = train.drop('id', axis=1)
    test = test.drop('id', axis=1)

    featuresCols = ['Basic_Demos-Enroll_Season', 'Basic_Demos-Age', 'Basic_Demos-Sex',
```

```
                    'CGAS-Season', 'CGAS-CGAS_Score', 'Physical-Season', 'Physical-BMI',
                    'Physical-Height', 'Physical-Weight', 'Physical-Waist_Circumference'
,
                    'Physical-Diastolic_BP', 'Physical-HeartRate', 'Physical-Systolic_BP
',
                    'Fitness_Endurance-Season', 'Fitness_Endurance-Max_Stage',
                    'Fitness_Endurance-Time_Mins', 'Fitness_Endurance-Time_Sec',
                    'FGC-Season', 'FGC-FGC_CU', 'FGC-FGC_CU_Zone', 'FGC-FGC_GSND',
                    'FGC-FGC_GSND_Zone', 'FGC-FGC_GSD', 'FGC-FGC_GSD_Zone', 'FGC-FGC_PU'
,
                    'FGC-FGC_PU_Zone', 'FGC-FGC_SRL', 'FGC-FGC_SRL_Zone', 'FGC-FGC_SRR',
                    'FGC-FGC_SRR_Zone', 'FGC-FGC_TL', 'FGC-FGC_TL_Zone', 'BIA-Season',
                    'BIA-BIA_Activity_Level_num', 'BIA-BIA_BMC', 'BIA-BIA_BMI',
                    'BIA-BIA_BMR', 'BIA-BIA_DEE', 'BIA-BIA_ECW', 'BIA-BIA_FFM',
                    'BIA-BIA_FFMI', 'BIA-BIA_FMI', 'BIA-BIA_Fat', 'BIA-BIA_Frame_num',
                    'BIA-BIA_ICW', 'BIA-BIA_LDM', 'BIA-BIA_LST', 'BIA-BIA_SMM',
                    'BIA-BIA_TBW', 'PAQ_A-Season', 'PAQ_A-PAQ_A_Total', 'PAQ_C-Season',
                    'PAQ_C-PAQ_C_Total', 'SDS-Season', 'SDS-SDS_Total_Raw',
                    'SDS-SDS_Total_T', 'PreInt_EduHx-Season',
                    'PreInt_EduHx-computerinternet_hoursday', 'sii']

    featuresCols += time_series_cols

    train = train[featuresCols]
    train = train.dropna(subset='sii')

    cat_c = ['Basic_Demos-Enroll_Season', 'CGAS-Season', 'Physical-Season',
             'Fitness_Endurance-Season', 'FGC-Season', 'BIA-Season',
             'PAQ_A-Season', 'PAQ_C-Season', 'SDS-Season', 'PreInt_EduHx-Season']

    def update(df):
        global cat_c
        for c in cat_c:
            df[c] = df[c].fillna('Missing')
            df[c] = df[c].astype('category')
        return df

    train = update(train)
    test = update(test)

    def create_mapping(column, dataset):
        unique_values = dataset[column].unique()
        return {value: idx for idx, value in enumerate(unique_values)}

    for col in cat_c:
        mapping = create_mapping(col, train)
        mappingTe = create_mapping(col, test)

```

```
        train[col] = train[col].replace(mapping).astype(int)
        test[col] = test[col].replace(mappingTe).astype(int)

    def quadratic_weighted_kappa(y_true, y_pred):
        return cohen_kappa_score(y_true, y_pred, weights='quadratic')

    def threshold_Rounder(oof_non_rounded, thresholds):
        return np.where(oof_non_rounded < thresholds[0], 0,
                        np.where(oof_non_rounded < thresholds[1], 1,
                                 np.where(oof_non_rounded < thresholds[2], 2, 3)))

    def evaluate_predictions(thresholds, y_true, oof_non_rounded):
        rounded_p = threshold_Rounder(oof_non_rounded, thresholds)
        return -quadratic_weighted_kappa(y_true, rounded_p)

    def TrainML(model_class, test_data):
        X = train.drop(['sii'], axis=1)
        y = train['sii']

        SKF = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=SEED)

        train_S = []
        test_S = []

        oof_non_rounded = np.zeros(len(y), dtype=float)
        oof_rounded = np.zeros(len(y), dtype=int)
        test_preds = np.zeros((len(test_data), n_splits))

        for fold, (train_idx, test_idx) in enumerate(tqdm(SKF.split(X, y), desc="Trainin
g Folds", total=n_splits)):
            X_train, X_val = X.iloc[train_idx], X.iloc[test_idx]
            y_train, y_val = y.iloc[train_idx], y.iloc[test_idx]

            model = clone(model_class)
            model.fit(X_train, y_train)

            y_train_pred = model.predict(X_train)
            y_val_pred = model.predict(X_val)

            oof_non_rounded[test_idx] = y_val_pred
            y_val_pred_rounded = y_val_pred.round(0).astype(int)
            oof_rounded[test_idx] = y_val_pred_rounded

            train_kappa = quadratic_weighted_kappa(y_train, y_train_pred.round(0).astype
(int))
            val_kappa = quadratic_weighted_kappa(y_val, y_val_pred_rounded)

            train_S.append(train_kappa)
```

```
126.           test_S.append(val_kappa)
127.
128.           test_preds[:, fold] = model.predict(test_data)
129.
130.           print(f"Fold {fold+1} - Train QWK: {train_kappa:.4f}, Validation QWK: {val_k
appa:.4f}")
131.           clear_output(wait=True)
132.
133.       print(f"Mean Train QWK --> {np.mean(train_S):.4f}")
134.       print(f"Mean Validation QWK ---> {np.mean(test_S):.4f}")
135.
136.       KappaOPtimizer = minimize(evaluate_predictions,
137.                                 x0=[0.5, 1.49, 2.5], args=(y, oof_non_rounded),
138.                                 method='Nelder-Mead')
139.       assert KappaOPtimizer.success, "Optimization did not converge."
140.       thresholds = KappaOPtimizer.x
141.
142.       oof_tuned = threshold_Rounder(oof_non_rounded, thresholds)
143.       tKappa = quadratic_weighted_kappa(y, oof_tuned)
144.
145.       print(f"----> || Optimized QWK SCORE :: {Fore.CYAN}{Style.BRIGHT} {tKappa:.3f}{S
tyle.RESET_ALL}")
146.
147.       fold_weights = [1.25, 1.0, 1.0, 1.0, 1.0]
148.       tpm = test_preds.dot(fold_weights) / np.sum(fold_weights)
149.       tpTuned = threshold_Rounder(tpm, thresholds)
150.
151.       submission = pd.DataFrame({
152.           'id': sample['id'],
153.           'sii': tpTuned
154.       })
155.
156.       return submission
157.
158.   # Model parameters for LightGBM
159.   Params = {
160.       'learning_rate': 0.046,
161.       'max_depth': 12,
162.       'num_leaves': 478,
163.       'min_data_in_leaf': 13,
164.       'feature_fraction': 0.893,
165.       'bagging_fraction': 0.784,
166.       'bagging_freq': 4,
167.       'lambda_l1': 10,  # Increased from 6.59
168.       'lambda_l2': 0.01  # Increased from 2.68e-06
169.   }
170.
171.
```

```
# XGBoost parameters
XGB_Params = {
    'learning_rate': 0.05,
    'max_depth': 6,
    'n_estimators': 200,
    'subsample': 0.8,
    'colsample_bytree': 0.8,
    'reg_alpha': 1,  # Increased from 0.1
    'reg_lambda': 5,  # Increased from 1
    'random_state': SEED
}


CatBoost_Params = {
    'learning_rate': 0.05,
    'depth': 6,
    'iterations': 200,
    'random_seed': SEED,
    'cat_features': cat_c,
    'verbose': 0,
    'l2_leaf_reg': 10  # Increase this value
}

# Create model instances
Light = LGBMRegressor(**Params, random_state=SEED, verbose=-1, n_estimators=300)
XGB_Model = XGBRegressor(**XGB_Params)
CatBoost_Model = CatBoostRegressor(**CatBoost_Params)

# Combine models using Voting Regressor
voting_model = VotingRegressor(estimators=[
    ('lightgbm', Light),
    ('xgboost', XGB_Model),
    ('catboost', CatBoost_Model)
])

# Train the ensemble model
Submission2 = TrainML(voting_model, test)

# Save submission
#Submission2.to_csv('submission.csv', index=False)
Submission2
```