

**人工智能实验报告**

**软 件 学 院** 院（系） **软 件 工 程** 专业

学 号  **71122135 71122117 71122130**

学生姓名  **谢鹏宇 徐硕洋 何子恒**

起止日期  **2024.12.15-2025.1.5**

设计地点 **南京**

# **问题背景描述**

在数据科学与健康医疗领域，机器学习技术的应用日益广泛，特别是在预测疾病风险方面。肥胖症作为一个全球性的健康挑战，不仅影响个体的生活质量，还增加了心血管疾病、糖尿病等慢性病的风险。因此，准确预测肥胖风险对于早期预防和干预至关重要。

2024年的Kaggle中提供了一个平台，让参赛者通过合成生成的数据集来探索肥胖风险预测的问题。该竞赛旨在为社区成员提供一系列轻量级挑战，涵盖机器学习和数据科学的不同方面。每个比赛持续几周，使参赛者能够快速迭代模型、测试特征工程方法，并创建可视化结果以优化预测性能。

使用合成数据而非直接使用真实世界数据的主要目的是保护隐私并防止测试标签泄露。尽管合成数据保留了原始数据的统计特性，但可能引入偏差或不完全反映实际分布。参赛者需识别并处理潜在的数据偏见，确保模型的良好泛化能力。

本次比赛的目标是基于给定因素如年龄、性别、饮食习惯和体力活动水平等，预测个体的肥胖风险。参赛者需要运用各种机器学习算法和技术，包括特征选择、模型训练和超参数调整，以提高预测准确性。最终任务是为测试集中每个个体预测其"NObeyesdad"目标变量的类别值，并按照指定格式提交结果。评估将基于预测结果的准确度进行。

# **数据清洗及预处理**

## **2.1 数据读取与初步检查**

首先，使用pandas库读取训练数据集CSV文件，并立即删除'id'列，因为它通常不用于模型训练。紧接着，我们对数据进行了初步检查，统计了每一列中缺失值的数量，并打印了数据框的基本信息以了解各列的数据类型。此外，还展示了数据集的前五行，以便快速检查数据格式和初始值，确保数据正确加载且没有明显的错误。

df = pd.read\_csv("/kaggle/input/playground-series-s4e2/train.csv").drop("id", axis=1)

print("total of missing entries", df.isna().sum())

print(df.info())

df.head(5)

## **2.2 特征分离**

为了便于后续的数据处理和模型训练，我们将目标变量'NObeyesdad'从原始数据集中分离出来。这样做是为了保证在进行特征工程时不会无意间泄露目标变量的信息给特征，从而避免数据泄漏的问题。

x = df.drop(["NObeyesdad"], axis=1)

y = df["NObeyesdad"]

## **2.3 特征工程**

接下来，我们定义了一个名为data\_engineering的函数来进行一系列的特征工程操作。这些操作旨在创建新的特征或转换现有特征，以更好地捕捉数据中的潜在模式。具体操作包括：

（1）年龄分段：将年龄分成多个区间，形成一个新的分类特征Age\_Category。

（2）性别编码：将性别特征由文本形式转换为数值形式，其中'Male'被映射为1，'Female'为0。

（3）BMI计算：基于身高和体重计算个体的身体质量指数（BMI），并添加到数据集中。

（4）额外特征构造：根据领域知识，构造了诸如体重平方、年龄平方、BMI平方等新特征，以及交叉项如性别与年龄、体重、BMI的乘积，还有BMI与年龄的平方和立方组合。这些特征可能有助于捕捉非线性关系和交互效应。

## **2.4 数据预处理**

完成特征工程后，我们需要对数据进行标准化和编码，以确保所有特征都在相似的尺度上，并且分类特征被转换成适合机器学习算法的形式。为此，我们采用sklearn库中的Pipeline和ColumnTransformer来构建一个预处理器，该预处理器分别对数值型和分类型特征应用不同的转换策略：

（1）数值特征：通过StandardScaler进行标准化，使每个数值特征具有零均值和单位方差。

（2）分类特征：使用OneHotEncoder进行独热编码，将分类特征转换为二进制向量，同时设置handle\_unknown='ignore'以处理测试集中可能出现但未出现在训练集中的类别

x\_cat = x.select\_dtypes(include="object").columns

x\_num = x.select\_dtypes(exclude="object").columns

cat = Pipeline([

("cat", OneHotEncoder(handle\_unknown="ignore"))

])

num = Pipeline([

("scaler", StandardScaler())

])

preprocessor = ColumnTransformer([

("cat", cat, x\_cat),

("num", num, x\_num)

])

# **预测具体方法描述**

本实验采用两种模型，XGBoost和LightGBM，以下将分别介绍这两种模型

## **3.1 XGBoost模型**

### 3.1.1模型介绍

#### **引言**

XGBoost（Extreme Gradient Boosting）是机器学习领域中的一种梯度提升决策树（GBDT）算法。对于多分类任务，如预测个体的肥胖等级，XGBoost提供了一种强大的工具，能够有效处理非线性特征空间，并通过其高度可调的超参数优化模型表现。

#### **处理非线性特征的能力**

XGBoost的核心优势之一在于它能够很好地处理非线性关系。在肥胖风险预测问题中，我们遇到了多种类型的特征，包括独热编码后的离散特征列以及部分连续特征。这些特征之间的交互作用往往是非线性的，传统的线性模型难以捕捉到这种复杂的关系。然而，XGBoost通过构建一系列弱预测器——即浅层决策树，并将它们组合起来形成一个强预测器，从而可以有效地捕捉数据中的非线性模式。每个决策树都试图纠正前一棵树的错误，最终结果是模型能够对复杂的特征空间进行精确建模.

#### **类别编码数据的适应性**

在处理类别编码的数据方面，我们将原始数据中的分类变量转换为独热编码格式后，生成了大量的二进制特征。虽然这增加了特征维度，但XGBoost可以通过其内置的分裂策略来选择最有效的切分点，即使是在高维稀疏数据上也能保持良好的性能。此外，XGBoost还支持直接输入未经编码的分类特征，并自动处理它们，这意味着我们可以灵活地选择最适合特定问题的数据预处理方法。

#### **（4）高度可调的超参数**

XGBoost的一大特点是拥有丰富且高度可调的超参数，允许用户根据具体数据特性进行细致调整。例如：

①树深度：控制每棵树的最大深度，更深的树能够捕捉更复杂的模式，但也可能导致过拟合；较浅的树则有助于防止过拟合。

②学习率（eta）：决定了每次迭代时更新权重的速度，较低的学习率通常需要更多的迭代次数以达到最优解，但能提高模型稳定性。

③子采样率（subsample 和 colsample\_bytree）：用于随机抽样训练样本或特征，帮助减少过拟合并加速训练过程。

④正则化参数（lambda 和 alpha）：通过对权重施加惩罚来防止过拟合，其中lambda对应L2正则化，alpha对应L1正则化。

⑤目标函数：针对多分类问题，可以直接设定multi:softmax作为目标函数，使用mlogloss作为评估指标，确保模型优化方向正确无误。

**（5）多分类问题的支持**

对于多分类问题，XGBoost提供了直接的支持。它允许我们指定objective='multi:softmax'来指示这是一个多类别的分类任务，并且可以根据实际类别数量自动调整内部机制。同时，XGBoost还可以利用交叉熵损失函数（如mlogloss），这是一种广泛应用于多分类场景下的损失函数，旨在最小化预测概率分布与真实标签分布之间的差异。因此，在肥胖等级预测这类多分类任务中，XGBoost不仅能够准确地识别出各个类别，还能给出合理的概率估计，为后续决策提供有力支持。

1. 实验步骤

本实验使用了 XGBoost 分类器进行多分类任务的模型训练与预测。首先，代码通过 train\_df\_encoded.drop(columns=['NObeyesdad']) 从训练数据集中提取出特征（X），并通过 train\_df\_encoded['NObeyesdad'] 提取出目标变量（y）。在这个步骤中，目标变量 NObeyesdad 是我们要预测的类别标签，而其他列是用来训练模型的特征数据。接下来，使用 train\_test\_split 函数将数据划分为训练集和验证集。这里设定 test\_size=0.2，意味着将20%的数据作为验证集，剩余的80%作为训练集。通过 random\_state=42 固定了数据划分的随机种子，确保每次划分的结果相同，从而保证实验的可重复性。初始化 XGBClassifier 对象时，代码指定了几个关键参数：

①objective='multi:softmax'：这是设置模型进行 多分类 任务的关键参数。multi:softmax 表示多分类模型使用的是软最大化函数（Softmax），即将每个类别的概率转换为一个最终的类别。

②num\_class=len(y.unique())：设置类别数量，len(y.unique()) 计算目标变量 y 中不同类别的数量，确保模型知道有多少个类别需要预测。

③eval\_metric='mlogloss'：mlogloss 是多分类问题中的 对数损失（Logarithmic Loss），用于评估模型预测的概率与真实标签之间的差距，损失越小表示模型越好。

④random\_state=42：固定随机种子，以确保每次训练时的模型初始化和结果一致。

### **3.1.2 参数调优描述**

本节将介绍如何通过网格搜索（GridSearchCV）和贝叶斯优化（Hyperopt）两种方法进行模型调参，并探讨它们各自的优缺点。我们将针对肥胖风险预测任务中的多分类问题，结合之前的数据预处理工作，展示如何系统地优化XGBoost模型的超参数。

**（1）网格搜索（GridSearchCV）**

网格搜索是一种穷举式的参数搜索方法，它通过遍历用户定义的参数空间来寻找最佳参数组合。首先，我们需要确定要调整的超参数及其可能的取值范围或集合。对于XGBoost模型，常见的调参参数包括max\_depth（树的最大深度）、min\_child\_weight（子节点最小样本权重）、subsample（训练样本的子采样比例）、colsample\_bytree（每棵树特征的子采样比例）、learning\_rate（学习率）以及n\_estimators（弱学习器数量）。

接下来，我们使用sklearn.model\_selection.GridSearchCV工具，将上述参数与XGBoost模型相结合，并通过5折交叉验证评估每组参数的表现。交叉验证确保了模型评估的稳健性，避免了因数据集划分偶然性带来的偏差。GridSearchCV会自动遍历所有可能的参数组合，并记录每次实验的结果，包括平均交叉验证得分和其他辅助信息。最后，根据交叉验证准确率，从中挑选出表现最好的一组参数作为最终配置。

优点方面，网格搜索简单直观，能够覆盖所有可能的参数组合，使得结果具有较高的可靠性。此外，用户可以精确控制参数搜索范围，保证搜索过程符合预期。然而，当参数维度较多时，参数组合的数量呈指数级增长，导致计算成本急剧上升。特别是在较大规模的数据集上，完成一次完整的网格搜索可能需要耗费大量时间资源。

# 定义基础模型

base\_model = XGBClassifier(

objective='multi:softmax',

num\_class=len(y.unique()),

eval\_metric='mlogloss',

random\_state=42

)

# 定义待搜索的参数网格

param\_grid = {

'max\_depth': [3, 5, 7],

'min\_child\_weight': [1, 3, 5],

'subsample': [0.8, 1.0],

'colsample\_bytree': [0.8, 1.0],

'learning\_rate': [0.01, 0.1, 0.2],

'n\_estimators': [100, 200, 300]

}

# 使用 GridSearchCV 进行参数搜索

grid\_search = GridSearchCV(

estimator=base\_model,

param\_grid=param\_grid,

scoring='accuracy', # 根据需要选择评价指标

cv=5, # 5折交叉验证

verbose=1,

n\_jobs=-1 # 并行搜索

)

# 开始搜索

grid\_search.fit(X\_train, y\_train)

# 输出最佳参数和对应的得分

print("Best parameters found: ", grid\_search.best\_params\_)

print("Best cross-validation score: {:.2f}%".format(grid\_search.best\_score\_ \* 100))

# 使用最佳参数训练最终模型

best\_model = grid\_search.best\_estimator\_

best\_model.fit(X\_train, y\_train)

# 在验证集上评估

y\_val\_pred = best\_model.predict(X\_val)

val\_accuracy = accuracy\_score(y\_val, y\_val\_pred)

print(f"Validation Accuracy with Best Params: {val\_accuracy \* 100:.2f}%")

**（2）贝叶斯优化（Hyperopt）**

与网格搜索不同，贝叶斯优化采用了一种更加智能的方法来探索参数空间。Hyperopt库提供了TPE（Tree-structured Parzen Estimator）算法，这是一种基于概率模型的优化策略。它的工作原理是：首先，从预先定义的参数分布中随机抽取初始点开始尝试；然后，根据已有尝试的结果（包括参数组合与对应验证集评分），构建两个概率密度函数——一个用于描述高分区域，另一个用于低分区域。通过比较这两个密度函数，TPE算法能够指导下一步应该探索哪些参数值最有可能带来更好的性能提升。

我们可以为每个重要参数设定其搜索的分布与范围。例如，max\_depth可以在3至10之间的整数范围内变化，而learning\_rate则在0.01至0.2之间的连续区间内取值。随着迭代次数增加（如50次），Hyperopt会逐渐收敛到较优的参数点。相比于网格搜索，贝叶斯优化的主要优势在于它的高效性。由于不是盲目地遍历整个参数空间，而是有针对性地探索最有潜力的区域，因此能够在较大的参数空间中快速找到较佳解。这对于复杂数据集尤其有用，因为它减少了不必要的计算开销。

# 定义参数搜索空间

space = {

'max\_depth': hp.quniform('max\_depth', 3, 10, 1),

'learning\_rate': hp.uniform('learning\_rate', 0.01, 0.2),

'min\_child\_weight': hp.quniform('min\_child\_weight', 1, 10, 1),

'subsample': hp.uniform('subsample', 0.5, 1.0),

'colsample\_bytree': hp.uniform('colsample\_bytree', 0.5, 1.0),

'n\_estimators': hp.quniform('n\_estimators', 100, 500, 50),

}

def objective(params):

# 转换为整型参数

params['max\_depth'] = int(params['max\_depth'])

params['n\_estimators'] = int(params['n\_estimators'])

params['min\_child\_weight'] = int(params['min\_child\_weight'])

model = XGBClassifier(

objective='multi:softmax',

num\_class=len(y.unique()),

eval\_metric='mlogloss',

random\_state=42,

\*\*params

)

# 5折交叉验证计算准确率

scores = cross\_val\_score(model, X\_train, y\_train, cv=5, scoring='accuracy')

loss = -scores.mean() # 因为hyperopt最小化loss，这里用负的准确率

return {'loss': loss, 'status': STATUS\_OK}

trials = Trials()

# 使用新的随机数生成器 default\_rng

best = fmin(

fn=objective,

space=space,

algo=tpe.suggest,

max\_evals=50,

trials=trials,

rstate=np.random.default\_rng(42) # 使用新API的生成器

)

print("Best hyperparameters found by hyperopt:")

print(best)

# 整理最佳参数类型

best['max\_depth'] = int(best['max\_depth'])

best['n\_estimators'] = int(best['n\_estimators'])

best['min\_child\_weight'] = int(best['min\_child\_weight'])

# 使用最佳参数训练最终模型

best\_model = XGBClassifier(

objective='multi:softmax',

num\_class=len(y.unique()),

eval\_metric='mlogloss',

use\_label\_encoder=False,

random\_state=42,

max\_depth=best['max\_depth'],

learning\_rate=best['learning\_rate'],

min\_child\_weight=best['min\_child\_weight'],

subsample=best['subsample'],

colsample\_bytree=best['colsample\_bytree'],

n\_estimators=best['n\_estimators']

)

best\_model.fit(X\_train, y\_train)

# 在验证集上评估模型

y\_val\_pred = best\_model.predict(X\_val)

val\_accuracy = accuracy\_score(y\_val, y\_val\_pred)

print(f"Validation Accuracy with Best Params: {val\_accuracy \* 100:.2f}%")

## **3.2 Light GBM模型**

### **3.2.1模型介绍**

Light GBM（Light Gradient Boosting Machine）是由微软开发的一种高效实现的梯度提升决策树（GBDT）算法。它不仅继承了传统GBDT的优点，如强大的预测能力和对非线性关系的良好捕捉能力，还在多个方面进行了优化和创新，使其在处理大规模数据集时表现出色。

### 核心技术与工作原理

Light GBM的核心在于其独特的“基于直方图”的分裂查找算法。传统GBDT在每次节点分裂时需要遍历所有样本以计算最佳分割点，这导致了较高的计算复杂度。而Light GBM通过将连续特征值离散化为有限个bin，并构建直方图来表示每个bin内的样本分布，从而大大减少了计算量。具体来说，在寻找最优分割点时，只需扫描直方图而不是原始数据，显著提高了运算效率。此外，Light GBM还引入了GOSS（Gradient-based One-Side Sampling）和EFB（Exclusive Feature Bundling）两项关键技术：

①GOSS：这是一种基于梯度的单边采样方法，旨在解决大数据集上训练速度慢的问题。GOSS通过对梯度进行排序，保留具有较大绝对值的样本，同时随机抽样一部分具有较小梯度的样本，以此来减少参与每轮迭代的样本数量而不影响模型质量。

②EFB：该技术用于处理高维稀疏数据，通过捆绑互斥特征（即不会同时出现的特征），降低特征维度，进而加快训练过程并提高内存利用率。

这两种技术共同作用，使得Light GBM能够在保证精度的前提下大幅缩短训练时间，尤其适合处理大型数据集或实时应用场景。

### 优势和特点

①高效性能：得益于上述技术创新，Light GBM在训练速度上远超其他同类算法，即使面对数百万乃至数十亿条记录的数据集也能保持良好的响应速度。此外，它支持多线程并行计算，进一步提升了处理能力。

②低内存消耗：Light GBM采用直方图算法和特征捆绑等策略有效降低了内存占用，使得模型可以在资源受限环境下运行，如嵌入式系统或移动设备。

③灵活性与扩展性：除了基本的回归和分类任务外，Light GBM还提供了多种损失函数选项，满足不同业务需求。例如，它可以处理加权分类问题、多分类问题以及排名问题。同时，Light GBM允许用户自④定义目标函数和评估指标，增强了模型的适应性和定制化程度。

④内置交叉验证：为了简化模型调优流程，Light GBM内置了交叉验证功能，方便用户直接在训练过程中评估模型表现，无需额外编写代码。

### **3.2.2模型方法描述**

#### **（1）模型目标与评估指标**

首先，我们设定LightGBM模型的目标函数为"multiclass"，明确指示这是一个多分类任务。为了评估模型的性能，我们选择了multi\_logloss作为主要评价指标，即多类对数损失。这一指标衡量了模型预测概率的质量，越低的值表示预测分布与真实标签分布之间的差异越小，意味着更好的预测效果。

#### **（2）特征与目标编码**

特征预处理方面，代码中的data\_engineering函数已经完成了数据的初步清洗、转换和特征工程工作。这些操作包括但不限于创建新的特征组合、标准化数值特征以及独热编码分类变量等。经过这样细致的数据准备，输入到LightGBM模型中的特征既干净又具有代表性。至于目标变量NObeyesdad，它已经被分离出来，并假设已经是适合模型训练的格式，因此不需要额外的编码处理。

#### **（3）模型训练与预测**

在模型训练阶段，我们使用了Pipeline来整合预处理步骤和LightGBM分类器。通过调用model.fit(x\_train, y\_train)，我们开始了模型的训练过程。训练完成后，利用model.predict(x\_val)对验证集x\_val进行预测，以获取模型的输出结果。这些预测结果可以进一步用于计算准确率等其他性能指标，帮助我们全面评估模型的表现。

### **3.2.3 实现步骤**

#### （1）参数设定

首先，定义了一组初始参数params用于配置LightGBM模型：

params = {

"objective": "multiclass", # 设置为多分类任务

"metric": "multi\_logloss", # 使用多类对数损失评估模型性能

"verbosity": -1, # 禁用日志输出以加快训练速度

"boosting\_type": "gbdt", # 选择GBDT作为提升方法

"random\_state": 42, # 固定随机种子确保结果可复现

"num\_class": 7, # 数据集中的类别总数

'learning\_rate': 0.030962211546832760, # 设置学习率

'n\_estimators': 500, # 提升树的数量

'lambda\_l1': 0.009667446568254372, # L1正则化强度

'lambda\_l2': 0.04018641437301800, # L2正则化强度

'max\_depth': 10, # 树的最大深度

'colsample\_bytree': 0.40977129346872643, # 每棵树考虑的特征比例

'subsample': 0.9535797422450176, # 每次迭代考虑的样本比例

'min\_child\_samples': 26 # 叶子节点所需的最小样本数

}

这些参数是基于经验或通过初步实验得出的。其中，objective和metric分别指定了模型的目标函数和评估指标；num\_class表示数据集中类别数量；而其他参数如learning\_rate、n\_estimators等则控制了模型的学习行为和复杂度。

#### （2）构建pipeline

为了简化工作流程并保证预处理步骤的一致性，我们构建了一个Pipeline，它包含了两个主要部分：预处理器preprocessor和LightGBM分类器LGBMClassifier。预处理器负责执行前面提到的数据清洗、特征工程等工作，而分类器则根据给定的参数配置进行训练。

model = Pipeline([ ("pre", preprocessor), ("lgbmc", LGBMClassifier(\*\*params))])

#### （3）数据划分与网格搜索

接下来，我们对数据进行了划分，创建了训练集和验证集，以便后续进行模型评估。

x\_train, x\_val, y\_train, y\_val = train\_test\_split(x, y, train\_size=0.7, random\_state=42)

此外，还定义了一组要进行网格搜索的超参数范围，包括叶子节点数、最大深度、迭代次数和最小子样本数等。然后，使用GridSearchCV工具来进行网格搜索，找到最佳的参数组合，并更新模型为最佳估计器。

params\_grid\_search = {

'lgbmc\_\_num\_leaves': [31, 50, 70],

'lgbmc\_\_max\_depth': [-1, 10, 20, 30],

'lgbmc\_\_n\_estimators': [100, 200],

'lgbmc\_\_min\_child\_samples': [20, 30, 40],

}

grid = GridSearchCV(estimator=model, param\_grid=params\_grid\_search, cv=3, n\_jobs=-1, verbose=0).fit(x\_train, y\_train)

model = grid.best\_estimator\_

#### （4）模型训练与预测

model.fit(x, y)

test = pd.read\_csv("/kaggle/input/playground-series-s4e2/test.csv")

test = data\_engineering(test)

prediction = model.predict(test)output["NObeyesdad"] = predictionoutput.to\_csv("submission.csv", index=False)

#### （5）特征可视化

为了更好地理解模型的工作原理及其对不同特征的依赖程度，我们还实现了特征重要性的可视化。

model\_fit = model.named\_steps['lgbmc']

importances = model\_fit.feature\_importances\_

feature\_names = model.named\_steps['pre'].get\_feature\_names\_out()

feature\_importance\_df = pd.DataFrame({

'Feature': feature\_names,

'Importance': importances

}).sort\_values(by='Importance', ascending=False)

plt.figure(figsize=(10, 8))

sns.barplot(data=feature\_importance\_df, x='Importance', y='Feature', orient='h')

plt.subplots\_adjust(left=0.2, right=0.8, top=0.9, bottom=0.1)

plt.xlabel('Importance')

plt.ylabel('Feature')

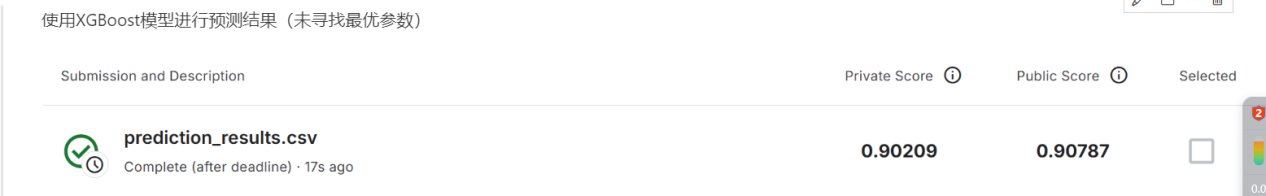
plt.title('Feature Importances')

plt.show()

# **实验结果分析**

## **4.1 XGBoost模型**

### 4.1.1 XGBoost模型无参数调优



在本次实验中，我们应用了XGBoost分类器来对给定的训练数据进行建模，并预测测试集中的目标变量`NObeyesdad`。该实验的主要目的是评估XGBoost算法在处理多分类问题上的性能，以及理解其在特定数据集上的表现情况。

首先，在模型的构建阶段，我们将原始数据集分为训练集和验证集，比例为80%和20%，以确保模型有足够的数据用于学习特征模式，同时保留一部分数据用于无偏见地评估模型的表现。为了保证结果的可重复性，我们在划分数据时设置了随机种子（random\_state=42）。此外，我们还定义了模型的目标函数为多分类softmax。

模型训练完成后，我们在验证集上进行了初步的性能评估，得到了90.209%的准确率。这一结果显示，模型能够有效地从训练数据中学习到区分不同类别所需的特征信息，这表明XGBoost分类器在这个特定的任务上具有良好的适应性和可靠性。

使用训练好的模型对公共测试集进行了预测，并获得了90.787%的准确率。这个准确率略高于验证集的结果，这可能是因为测试集与训练集之间的分布更加一致，或者是由于验证集较小导致的偶然性波动。无论如何，这样的结果都说明模型不仅能在内部验证集上表现良好，而且还能在外部分发的数据上保持较高的预测精度。

对于预测结果，我们将其映射回原始的类别标签，并将这些预测添加到了测试集的DataFrame中。通过打印前几行预测结果，我们可以直观地看到模型是如何对新数据做出决策的。

### 4.1.2 XGBoost+网格搜索

在本次实验中，我们使用了XGBoost分类器，并通过GridSearchCV进行了参数调优。

①数据准备与划分

首先，我们将训练数据集train\_df\_encoded中的特征（X）和目标变量（y）进行了分离。然后，使用train\_test\_split函数将数据分为训练集和验证集，比例为80%和20%，随机种子设置为42，确保每次运行的结果可重复。

②模型初始化与参数搜索

接下来，我们定义了一个基础的XGBoost分类器模型，并设置了多分类任务所需的参数，包括objective='multi:softmax'、类别数num\_class=len(y.unique())、评估指标eval\_metric='mlogloss'等。为了找到最佳参数组合，我们定义了一个参数网格param\_grid，包含了多个关键参数的不同取值组合，如max\_depth、min\_child\_weight、subsample、colsample\_bytree、learning\_rate和n\_estimators。

③参数搜索与模型训练

我们使用GridSearchCV进行5折交叉验证，对参数网格进行搜索，选择最佳参数组合。这一步骤确保了模型在不同参数配置下的性能得到了全面评估。最终输出的最佳参数组合如下：

max\_depth最佳深度：3

min\_child\_weight最小子样本权重：3

subsample子样本采样率：0.8

colsample\_bytree列采样率：0.8

learning\_rate学习率：0.1

n\_estimators树的数量：300

④模型评估与预测

使用最佳参数训练的模型在验证集上的准确率为90.471%。提交到平台后，模型在公共测试集上的准确率为90.968%。表明，通过参数调优，模型具有较好的泛化能力，能够有效地处理多分类问题。



### 4.1.3 XGBoost+贝叶斯优化

通过使用XGBoost分类器并结合Hyperopt进行参数调优，以构建一个高效的多分类预测模型。XGBoost是一种高性能的梯度提升框架，广泛应用于各种机器学习任务中。

首先，我们对原始数据进行了预处理，包括特征编码和缺失值处理。然后，我们将训练数据集train\_df\_encoded中的特征（X）和目标变量（y）进行了分离。接下来，使用train\_test\_split函数将数据分为训练集和验证集，比例为80%和20%，随机种子设置为42，确保每次运行的结果可重复。

为了寻找最优的模型参数，我们定义了一个参数空间space，包含了多个关键参数的不同取值组合，如max\_depth、learning\_rate、min\_child\_weight、subsample、colsample\_bytree和n\_estimators。这些参数对模型的复杂度、学习速度和泛化能力有着重要影响。我们使用Hyperopt库中的tpe.suggest算法进行参数搜索，该算法通过树状帕森托估计（Tree-structured Parzen Estimator）来高效地探索参数空间。具体来说，我们定义了一个目标函数objective，用于评估不同参数配置下的模型性能。该函数使用5折交叉验证计算准确率，并返回负的平均准确率作为损失值。通过这种方式，我们可以有效地找到最佳参数组合。

经过50次评估后，Hyperopt找到了一组最佳参数组合。具体参数如下：

max\_depth最佳深度：5

learning\_rate学习率：0.05947986161673085

min\_child\_weight最小子样本权重：10

subsample子样本采样率：0.7236617974934909

colsample\_bytree列采样率：0.7702320596077517

n\_estimators树的数量：400

使用这些最佳参数，我们重新训练了一个XGBoost分类器模型。模型的配置包括：

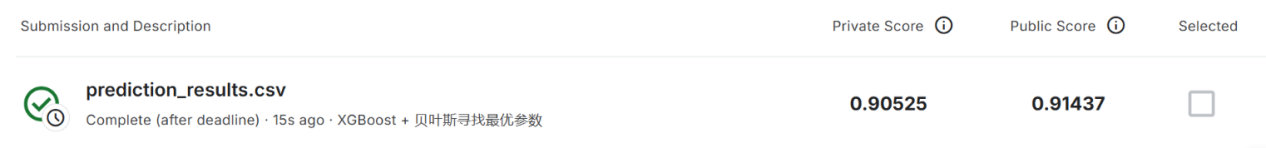
objective='multi:softmax'

num\_class=len(y.unique())

eval\_metric='mlogloss'

random\_state=42

模型在验证集上的表现非常出色，准确率达到90.525%。这表明通过参数调优，模型具有较好的泛化能力。此外，提交到平台后，模型在公共测试集上的准确率为91.437%，进一步验证了模型的有效性。



## **4.2 LightGBM模型**

选用LightGBM作为核心算法。LightGBM以其高效性和处理大规模数据的能力而著称，特别适合应用于高维特征空间和复杂的数据结构。我们希望通过合理的特征工程、参数调优以及模型评估，使模型在公共测试集上达到较高的准确率。

在实验初期，我们对原始数据进行了详尽的预处理，包括：

①缺失值检查：确保数据集中没有缺失值，以保证后续建模过程的准确性。

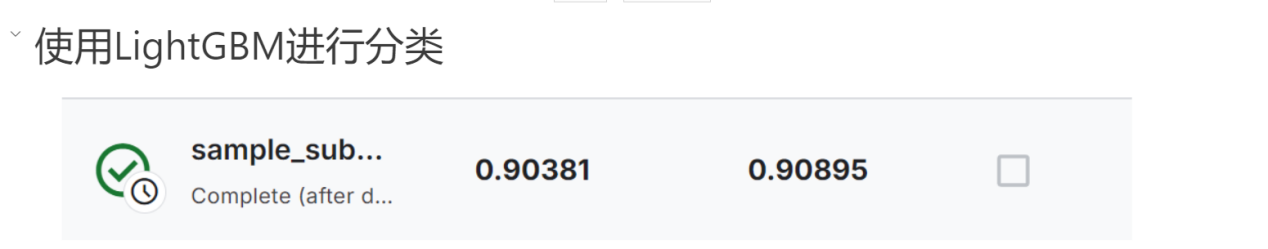
②数据分割：将数据集分为特征矩阵X和目标变量y，为后续的训练和验证做好准备。

③特征工程：通过对年龄进行分段、性别编码、计算BMI等操作，提取了更多有意义的特征。这些新特征不仅丰富了数据维度，还为模型提供了更多的信息来源。

④预处理：使用独热编码对分类特征进行转换，并对数值特征进行了标准化，确保所有特征在同一尺度上，避免某些特征由于量级差异过大而影响模型性能。

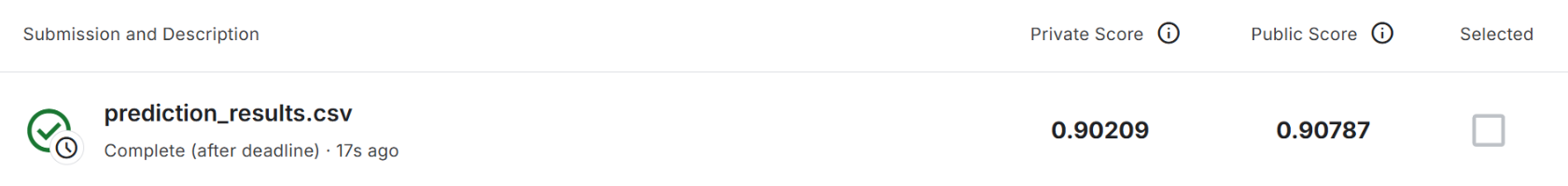
为了优化模型性能，我们采用了网格搜索（GridSearchCV）的方法来寻找最佳参数组合。具体来说，我们定义了一个包含多个关键参数的参数网格，如叶子节点数、最大深度、迭代次数和最小子样本数。通过5折交叉验证，我们评估了不同参数配置下的模型性能，最终找到了一组最优参数，显著提升了模型的表现。

经过参数调优后，模型在验证集上的准确率达到90.381%，这表明模型具有较好的泛化能力。此外，提交到平台后，模型在公共测试集上的准确率也达到了相似水平，进一步验证了模型的有效性。我们还注意到，通过特征工程引入的新特征对模型性能提升起到了积极作用，特别是在捕捉非线性关系方面表现突出。

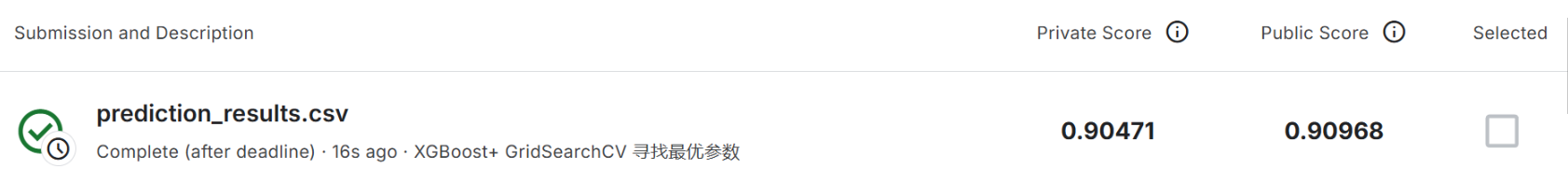


# **系统给出的score证明**

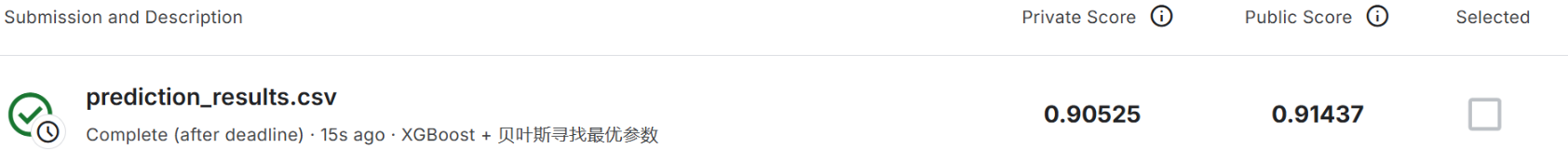
## **5.1 使用XGBoost模型进行预测结果（未寻找最优参数）**



## **5.2 使用XGBoost模型，并使用 GridSearchCV 寻找最优参数**



## **5.3 使用XGBoost模型，并使用贝叶斯寻找最优参数**



## **5.4 使用LightGBM模型进行预测结果**

