成年人死亡率预测 - 程序报告

吴天宇 12334125

1 实验概要

1.1 实验内容

本实验旨在利用机器学习算法对成年人死亡率（Adult Mortality）进行预测。成年人死亡率是衡量一个国家或地区健康状况的重要指标，指每1000名15至60岁人口中死亡的概率。通过分析影响成年人死亡率的各种因素，建立回归模型，可以为公共卫生政策制定提供数据支持。

1.2 实验结果概要

在本实验中，我们对提供的训练数据进行了预处理，包括缺失值填补和特征归一化。然后，使用了四种回归模型（线性回归、随机森林、梯度提升、XGBoost）对成年人死亡率进行了预测。通过评估各模型的性能，最终选择了表现最好的模型进行保存和预测。实验结果表明，随机森林模型在训练集上取得了最高的R^2分数，说明其具有较强的预测能力。

- 线性回归（Linear Regression）
  - 核心思想：线性回归试图找到输入特征和目标变量之间的线性关系。通过最小化预测值与实际值之间的均方误差，线性回归模型确定最佳拟合直线。
- 随机森林（Random Forest）
  - 核心思想：随机森林是由多棵决策树组成的集成模型。每棵树在训练时都使用了数据的不同子集和特征的随机子集。最终预测结果是所有树预测结果的平均值（回归问题）或多数投票（分类问题）。
  - 优点：通过集成多棵树，随机森林可以减少过拟合，提高模型的泛化能力。
- 梯度提升（Gradient Boosting）
  - 核心思想：梯度提升是一种迭代的集成方法，通过逐步添加新的弱学习器（通常是决策树）来纠正前一轮模型的错误。每一轮的新模型都是在前一轮模型的残差上进行训练的。
  - 优点：通过不断优化模型的残差，梯度提升可以构建出强大的预测模型。
- XGBoost
  - 核心思想：XGBoost（Extreme Gradient Boosting）是梯度提升的改进版本，具有更高的效率和更强的性能。它通过并行计算、正则化和处理缺失值等技术来提升模型的训练速度和预测准确性。
  - 优点：XGBoost在处理大规模数据和复杂模型时表现出色，常用于各种机器学习竞赛中。

2 回归模型实现

本实验通过数据预处理、模型训练、预测和评估等步骤实现了成年人死亡率的回归预测。数据预处理环节填补了缺失值并归一化特征，确保数据一致性；模型训练阶段使用了线性回归、随机森林、梯度提升和XGBoost四种算法，依次拟合数据并保存模型参数；在性能评估中，选用均方误差（MSE）和决定系数（R^2 Score）评估模型，最终选择随机森林为最佳模型。

2.1 导入必要的库

In [8]:
```
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, r2_score
from xgboost import XGBRegressor
import joblib
import shutil
import os
```

2.2 数据预处理函数

- 使用 pd.read_csv 函数读取了训练数据集 train_data.csv，并将其存储在 train_data 变量中。
- preprocess_data 函数用于对数据进行预处理，包括缺失值填补和归一化。
- 定义数值型特征列表 numeric_columns，这些特征需要进行填补和归一化。
- 删除了与预测无关的列 Country 和 Status，因为它们是非数值型特征，无法直接用于模型训练。
- 使用 SimpleImputer 填补缺失值，默认使用均值填补。
- 使用 MinMaxScaler 对数值型特征进行归一化处理，将特征值缩放到 [0,1] 区间。
- 删除了 Year 列，因为年份对于预测成年人死亡率的影响较小，且可能引入时间相关的偏差。
- 最后，返回预处理后的数据，以及用于填补和归一化的 imputer 和 scaler，以便在测试数据预处理中使用相同的参数。

In [9]:
```python
# ==================== 数据预处理函数 ====================
train_data = pd.read_csv('./data/train_data.csv')

def preprocess_data(data, imputer=None, scaler=None):
    """
    预处理数据：填补缺失值并归一化数值型列。
    :param data: 待处理的数据
    :param imputer: 缺失值填补器（默认为None，使用均值填补）
    :param scaler: 归一化器（默认为None，使用MinMaxScaler）
    :return: 预处理后的数据、imputer 和 scaler
    """
    numeric_columns = [
        'Year', 'Life expectancy ', 'infant deaths', 'Alcohol',
        'percentage expenditure', 'Hepatitis B', 'Measles ', ' BMI ',
        'under-five deaths ', 'Polio', 'Total expenditure', 'Diphtheria ',
        ' HIV/AIDS', 'GDP', 'Population', ' thinness  1-19 years',
        ' thinness 5-9 years', 'Income composition of resources', 'Schooling'
    ]

    # 删除无关列
    data = data.drop(["Country", "Status"], axis=1)

    # 填补缺失值
    if imputer is None:
        imputer = SimpleImputer(strategy='mean', missing_values=np.nan)
        imputer.fit(data[numeric_columns])
    data[numeric_columns] = imputer.transform(data[numeric_columns])

    # 归一化处理
    if scaler is None:
        scaler = MinMaxScaler()
        scaler.fit(data[numeric_columns])
    data[numeric_columns] = scaler.transform(data[numeric_columns])

    # 删除非必要列
    data = data.drop(['Year'], axis=1)

    return pd.DataFrame(data, columns=data.columns), imputer, scaler
```

2.3 模型训练函数

- model_fit 函数用于训练指定类型的回归模型。
- 将 train_data 的最后一列 Adult Mortality 作为标签 train_y，其余列作为特征。
- 调用之前定义的 preprocess_data 函数对特征进行预处理，得到归一化后的训练数据 train_x。
- 根据 model_type 参数选择不同的回归模型，包括线性回归、随机森林、梯度提升和 XGBoost。
- 训练模型后，创建 ./results 目录，并将模型、imputer 和 scaler 保存到该目录，以便后续加载和预测。

In [10]:
```python
# ==================== 模型训练函数 ====================
def model_fit(train_data, model_type="linear"):
    """
    训练指定类型的回归模型并保存至 ./results 目录。
    :param train_data: 训练数据
    :param model_type: 模型类型（默认为线性回归）
    :return: 训练好的模型
    """
    train_y = train_data.iloc[:, -1].values
    train_data = train_data.drop(["Adult Mortality"], axis=1)
```

```
    train_data_norm, imputer, scaler = preprocess_data(train_data)
    train_x = train_data_norm.values

    if model_type == "linear":
        model = LinearRegression()
    elif model_type == "random_forest":
        model = RandomForestRegressor(n_estimators=200, random_state=42)
    elif model_type == "gradient_boosting":
        model = GradientBoostingRegressor(n_estimators=500, random_state=42)
    elif model_type == "xgboost":
        model = XGBRegressor(n_estimators=20, random_state=42)
    else:
        raise ValueError("未知模型类型")

    model.fit(train_x, train_y)

    # 创建结果目录并保存模型和预处理器
    os.makedirs('./results', exist_ok=True)
    joblib.dump(model, f"./results/{model_type}_model.pkl")
    joblib.dump(imputer, "./results/imputer.pkl")
    joblib.dump(scaler, "./results/scaler.pkl")

    return model
```

2.4 定义预测函数

- predict_model 函数用于使用训练好的模型对测试数据进行预测。
- 加载之前保存的模型、imputer 和 scaler，确保测试数据与训练数据使用相同的预处理参数。
- 对测试数据进行预处理，包括缺失值填补和归一化。
- 使用加载的模型对预处理后的测试数据进行预测，返回预测结果。

In [11]:
```
# ==================== 预测函数 ====================
def predict_model(test_data, model_type="linear"):
    """
    使用训练好的模型对测试数据进行预测。
    :param test_data: 测试数据
    :param model_type: 使用的模型类型
    :return: 预测结果
    """
    loaded_model = joblib.load(f"./results/{model_type}_model.pkl")
    imputer = joblib.load("./results/imputer.pkl")
    scaler = joblib.load("./results/scaler.pkl")

    test_data_norm, _, _ = preprocess_data(test_data, imputer, scaler)
    test_x = test_data_norm.values

    predictions_model = loaded_model.predict(test_x)

    return predictions_model
```

2.5 定义模型性能评估函数

- evaluate_model 函数用于评估模型在训练集上的性能。
- 使用 predict_model 函数对训练数据进行预测，得到预测值 y_pred。
- 计算均方误差（MSE）和决定系数（R^2 Score），衡量模型的回归性能。
- 打印并返回模型的 MSE 和 R^2 分数。

In [12]:
```
# ==================== 模型性能评估函数 ====================
def evaluate_model(train_data, model_type="linear"):
    """
    评估模型性能并打印结果。
    :param train_data: 训练数据
    :param model_type: 模型类型
    :return: 模型的 MSE 和 R2 分数
    """
    label = train_data['Adult Mortality']
    train_data = train_data.drop(columns=['Adult Mortality'])
    y_pred = predict_model(train_data, model_type)

    mse = mean_squared_error(label, y_pred)
    r2 = r2_score(label, y_pred)
```

```
    print(f"模型: {model_type}, MSE: {mse}, R2 Score: {r2}")

    return mse, r2
```

2.6 训练模型并选择最佳模型

- 定义了 best_model_type 和 best_score 变量，用于记录最佳模型类型和最高的 R^2 分数。
- 使用一个循环，遍历四种模型类型，依次进行模型训练和评估。
- 对于每种模型，调用 model_fit 进行训练，然后调用 evaluate_model 进行性能评估。
- 如果当前模型的 R^2 分数高于 best_score，则更新 best_score 和 best_model_type。
- 最后，打印最佳模型的信息，并将最佳模型复制为 best_model.pkl，方便后续使用。

In [13]:
```
# ==================== 训练并选择最佳模型 ====================
best_model_type = None
best_score = float('-inf')

for model_type in ["linear", "random_forest", "gradient_boosting", "xgboost"]:
    model_fit(train_data, model_type)
    mse, r2 = evaluate_model(train_data, model_type)

    if r2 > best_score:
        best_score = r2
        best_model_type = model_type

# 复制最佳模型为 best_model.pkl
print(f"最佳模型: {best_model_type}, MSE: {best_score}, R2 Score: {r2}")
best_model_path = f"./results/{best_model_type}_model.pkl"
best_model_destination = "./results/best_model.pkl"
shutil.copy(best_model_path, best_model_destination)
print(f"最佳模型已保存至: {best_model_destination}")
```

```
模型: linear, MSE: 7451.44129449904, R2 Score: 0.5207060487947698
模型: random_forest, MSE: 890.7511780607877, R2 Score: 0.9427048224900266
模型: gradient_boosting, MSE: 1298.166788331046, R2 Score: 0.9164989074312473
模型: xgboost, MSE: 1648.7380992654453, R2 Score: 0.8939493298530579
最佳模型: random_forest, MSE: 0.9427048224900266, R2 Score: 0.8939493298530579
最佳模型已保存至: ./results/best_model.pkl
```

2.7 创建平台测试-预测函数

- 该部分代码定义了一个用于平台测试的预测函数 predict，供平台调用。
- 函数加载了之前保存的最佳模型，以及对应的 imputer 和 scaler。
- 对输入的测试数据进行预处理，然后使用模型进行预测，返回预测结果。
- 注意该部分代码被注释掉了，实际使用时需要去掉注释。

In [14]:
```
# ==================== 平台测试-预测函数 ====================
def predict(test_data):
    """
    使用训练好的模型对测试数据进行预测。
    :param test_data: 测试数据
    :return: 预测结果
    """
    loaded_model = joblib.load(f"./results/best_model.pkl")
    imputer = joblib.load("./results/imputer.pkl")
    scaler = joblib.load("./results/scaler.pkl")

    test_data_norm, _, _ = preprocess_data(test_data, imputer, scaler)
    test_x = test_data_norm.values

    predictions = loaded_model.predict(test_x)

    return predictions
```