1. import pandas as pd
2. from sklearn.model\_selection import train\_test\_split, RandomizedSearchCV
3. from sklearn.preprocessing import StandardScaler
4. def load\_data(fpath):
5. # 读取数据
6. data = pd.read\_csv(fpath)
7. # 数据清洗，删除包含空字段的行
8. data.dropna(inplace=True)
9. # 特征工程
10. features = ['Age', 'Gender', 'Sleep duration', 'REM sleep percentage', 'Deep sleep percentage',
11. 'Light sleep percentage', 'Awakenings', 'Caffeine consumption',
12. 'Alcohol consumption', 'Smoking status', 'Exercise frequency']
13. # 将性别、吸烟状态转换为数值
14. data['Gender'] = data['Gender'].map({'Male': 0, 'Female': 1})
15. data['Smoking status'] = data['Smoking status'].map({'No': 0, 'Yes': 1})
16. # 提取特征和目标变量
17. X = data[features].values
18. y = data['Sleep efficiency'].values
19. # 标准化数据
20. scaler = StandardScaler()
21. X\_scaled = scaler.fit\_transform(X)
22. # 随机划分训练集和测试集
23. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)
24. # 计算默认值（使用训练集的均值）
25. default\_values = data[features].mean().to\_dict()
26. return X\_train, X\_test, y\_train, y\_test, scaler, default\_values
27. import torch
28. from torch import nn, optim
29. from sklearn.metrics import mean\_squared\_error, r2\_score
30. from torch.utils.data import DataLoader, TensorDataset
31. from draw\_plot import plot\_loss\_curve, plot\_results
32. class LinearRegressionModel(nn.Module):
33. # input\_dim 输入特征的维度
34. def \_\_init\_\_(self, input\_dim):
35. super(LinearRegressionModel, self).\_\_init\_\_()
36. # 定义线性层，输入维度为 input\_dim，输出维度为 1
37. self.linear = nn.Linear(input\_dim, 1)
38. # 定义ReLU激活函数缓解过拟合问题， 能够在x>0时保持梯度不衰减，从而缓解梯度消失问题
39. self.relu = nn.ReLU()
40. # 定义前向传播逻辑
41. def forward(self, x):
42. # 线性层对输入数据进行线性变换后传递给ReLU激活函数，得到非线性变换后的输出
43. return self.relu(self.linear(x))
44. # 多元线性回归算法
45. def linear\_regression(X\_train\_lr, X\_test\_lr, y\_train\_lr, y\_test\_lr):
46. # 创建模型实例 维度为 1
47. input\_dim = X\_train\_lr.shape[1]
48. model\_lr = LinearRegressionModel(input\_dim)
49. # 定义损失函数均方误差，预测与实际之间的平均平方差
50. criterion = nn.MSELoss()
51. # 优化器 SGD=随机梯度下降 lr=学习率 momentum=动量加速，跳过局部最小值
52. optimizer = optim.SGD(model\_lr.parameters(), lr=0.01, momentum=0.9)
53. # 学习率指数衰减 gamma=衰减率
54. scheduler = optim.lr\_scheduler.ExponentialLR(optimizer, gamma=0.95)
55. X\_train\_tensor\_lr = torch.tensor(X\_train\_lr, dtype=torch.float32)
56. y\_train\_tensor\_lr = torch.tensor(y\_train\_lr, dtype=torch.float32).view(-1, 1)
57. train\_dataset\_lr = TensorDataset(X\_train\_tensor\_lr, y\_train\_tensor\_lr)
58. train\_loader\_lr = DataLoader(train\_dataset\_lr, batch\_size=32, shuffle=True)
59. # 训练模型
60. num\_epochs = 500 # 迭代100次
61. epoch\_loss = [] # 存储每个 epoch 的损失
62. for epoch in range(num\_epochs):
63. model\_lr.train()
64. total\_loss = 0
65. for inputs, targets in train\_loader\_lr:
66. # 前向传播
67. outputs = model\_lr(inputs)
68. loss = criterion(outputs, targets)
69. # 反向传播和优化
70. optimizer.zero\_grad()
71. loss.backward()
72. optimizer.step()
73. total\_loss += loss.item()
74. avg\_loss = total\_loss / len(train\_loader\_lr)
75. epoch\_loss.append(avg\_loss)
76. # 更新学习率
77. scheduler.step()
78. # if (epoch + 1) % 10 == 0:
79. # print(f'Epoch [{epoch + 1}/{num\_epochs}], Loss: {avg\_loss:.4f}')
80. plot\_loss\_curve(epoch\_loss, '多元线性回归 训练损失变化曲线')
81. # 评估模型
82. model\_lr.eval()
83. y\_test\_pred = model\_lr(torch.tensor(X\_test\_lr, dtype=torch.float32)).detach().numpy()
84. mse = mean\_squared\_error(y\_test\_lr, y\_test\_pred)
85. r2 = r2\_score(y\_test\_lr, y\_test\_pred)
86. # print(f"均方误差 (MSE): {mse}")
87. # print(f"决定系数 (R²): {r2}")
88. plot\_results(y\_test\_lr, y\_test\_pred, '多元线性回归 真实值与预测值对比')return mse, r2
89. from sklearn.metrics import mean\_squared\_error, r2\_score
90. import torch
91. import torch.nn as nn
92. import torch.optim as optim
93. from torch.utils.data import DataLoader, TensorDataset
94. from draw\_plot import plot\_loss\_curve, plot\_results
95. # 定义神经网络模型
96. class NeuralNetworkModel(nn.Module):
97. # input\_dim控制输入层的神经元数量
98. def \_\_init\_\_(self, input\_dim):
99. super(NeuralNetworkModel, self).\_\_init\_\_()
100. # 第一线性层 64个
101. self.fc1 = nn.Linear(input\_dim, 64)
102. # 第二线性层 64-32
103. self.fc2 = nn.Linear(64, 32)
104. # 第三线性层 32-1
105. self.fc3 = nn.Linear(32, 1)
106. # ReLU 非线性激活函数 提高能力
107. self.relu = nn.ReLU()
108. # 前向传播
109. def forward(self, x):
110. # 输出第一线性层的激活值
111. x = self.relu(self.fc1(x))
112. # 输出第二线性层的激活值
113. x = self.relu(self.fc2(x))
114. # 第三层不用输出，已经是最终值
115. x = self.fc3(x)
116. return x
117. # 神经网络回归算法
118. def neural\_network\_regression(X\_train\_nn, X\_test\_nn, y\_train\_nn, y\_test\_nn):
119. # 创建模型实例 维度为 1
120. input\_dim = X\_train\_nn.shape[1]
121. model\_nn = NeuralNetworkModel(input\_dim)
122. # 定义损失函数均方误差，预测与实际之间的平均平方差
123. criterion = nn.MSELoss()
124. # 优化器 Adam
125. optimizer = optim.Adam(model\_nn.parameters(), lr=0.01)
126. # 转换为 PyTorch 张量
127. X\_train\_tensor\_nn = torch.tensor(X\_train\_nn, dtype=torch.float32)
128. # 转换为一个二维张量，使其形状变为 [样本数量, 1]
129. y\_train\_tensor\_nn = torch.tensor(y\_train\_nn, dtype=torch.float32).view(-1, 1)
130. # 设置数据集
131. train\_dataset\_nn = TensorDataset(X\_train\_tensor\_nn, y\_train\_tensor\_nn)
132. # DataLoader创建训练加载器 batch\_size每个批次包含32个样本 shuffle打乱数据以提高模型的泛化能力
133. train\_loader\_nn = DataLoader(train\_dataset\_nn, batch\_size=32, shuffle=True)
134. # 训练模型
135. num\_epochs = 500 # 迭代200次
136. epoch\_loss = [] # 存储每个 epoch 的损失
137. for epoch in range(num\_epochs):
138. model\_nn.train()
139. total\_loss = 0
140. for inputs, targets in train\_loader\_nn:
141. # 前向传播
142. outputs = model\_nn(inputs)
143. loss = criterion(outputs, targets)
144. # 反向传播和优化
145. optimizer.zero\_grad()
146. loss.backward()
147. optimizer.step()
148. total\_loss += loss.item()
149. avg\_loss = total\_loss / len(train\_loader\_nn)
150. epoch\_loss.append(avg\_loss)
151. # if (epoch + 1) % 10 == 0:
152. # print(f'Epoch [{epoch + 1}/{num\_epochs}], Loss: {avg\_loss:.4f}')
153. plot\_loss\_curve(epoch\_loss, '神经网络回归 训练损失变化曲线')
154. # 评估模型
155. model\_nn.eval()
156. y\_test\_pred\_nn = model\_nn(torch.tensor(X\_test\_nn, dtype=torch.float32)).detach().numpy()
157. mse = mean\_squared\_error(y\_test\_nn, y\_test\_pred\_nn)
158. r2 = r2\_score(y\_test\_nn, y\_test\_pred\_nn)
159. # print(f"均方误差 (MSE): {mse}")
160. # print(f"决定系数 (R²): {r2}")
161. plot\_results(y\_test\_nn, y\_test\_pred\_nn, '神经网络回归 真实值与预测值对比')
162. return mse, r2
163. import xgboost as xgb
164. from sklearn.metrics import mean\_squared\_error, r2\_scorefrom draw\_plot import plot\_results, plot\_loss\_curve
165. # XGBoost算法
166. def xgboost\_regression(X\_train\_xgb, X\_test\_xgb, y\_train\_xgb, y\_test\_xgb):
167. # 定义和训练XGBoost回归模型 objective=回归问题 n\_estimators=200棵树 learning\_rate=学习率0.01 max\_depth=树的最大深度3
168. model\_xgb = xgb.XGBRegressor(objective='reg:squarederror', n\_estimators=200, learning\_rate=0.01, max\_depth=3)
169. model\_xgb.fit(X\_train\_xgb, y\_train\_xgb, eval\_set=[(X\_train\_xgb, y\_train\_xgb)], verbose=False)# 进行预测
170. y\_train\_pred\_xgb = model\_xgb.predict(X\_train\_xgb)
171. y\_test\_pred\_xgb = model\_xgb.predict(X\_test\_xgb)# 计算评估指标
172. mse = mean\_squared\_error(y\_train\_xgb, y\_train\_pred\_xgb)
173. r2 = r2\_score(y\_train\_xgb, y\_train\_pred\_xgb)# print(f"均方误差 (MSE): {mse}")
174. # print(f"决定系数 (R²): {r2}")plot\_results(y\_test\_xgb, y\_test\_pred\_xgb, 'XGBoost回归 真实值与预测值对比')# 绘制训练损失变化曲线
175. results = model\_xgb.evals\_result()
176. rmse\_values = results['validation\_0']['rmse']
177. plot\_loss\_curve(rmse\_values, 'XGBoost回归 训练损失变化')return mse, r2
178. import numpy as np
179. from sklearn.ensemble import RandomForestRegressor
180. from sklearn.model\_selection import RandomizedSearchCV
181. from sklearn.metrics import mean\_squared\_error, r2\_score
182. from draw\_plot import plot\_results
183. # 随机森林算法
184. def random\_forest\_regression(X\_train\_rf, X\_test\_rf, y\_train\_rf, y\_test\_rf):
185. # 定义参数搜索空间
186. """
187. criterion: 均方误差和绝对误差的决策树分裂标准。
188. n\_estimators: 森林中的200 到 2000 的 10 个不同树木数量。
189. max\_features: 每次分裂时考虑平方根和对数两种最大特征数量。
190. max\_depth: 树的最大深度。定义了从 10 到 100 的 10 个不同深度，并不限制深度。
191. min\_samples\_split: 内部节点再划分所需的最小样本数。
192. min\_samples\_leaf: 叶节点所需的最小样本数。
193. bootstrap: 是否使用自助法抽样。
194. """
195. criterion = ['squared\_error', 'absolute\_error']
196. n\_estimators = [int(x) for x in np.linspace(start=200, stop=2000, num=10)]
197. max\_features = ['sqrt', 'log2']
198. max\_depth = [int(x) for x in np.linspace(10, 100, num=10)]
199. max\_depth.append(None)
200. min\_samples\_split = [2, 5, 10]
201. min\_samples\_leaf = [1, 2, 4]
202. bootstrap = [True, False]
203. random\_grid = {
204. 'criterion': criterion,
205. 'n\_estimators': n\_estimators,
206. 'max\_features': max\_features,
207. 'max\_depth': max\_depth,
208. 'min\_samples\_split': min\_samples\_split,
209. 'min\_samples\_leaf': min\_samples\_leaf,
210. 'bootstrap': bootstrap
211. }# 构建随机森林模型
212. clf = RandomForestRegressor(n\_estimators=200, random\_state=42)
213. # 10 次随机搜索，进行 3 折交叉验证 verbose=0不打印
214. clf\_random = RandomizedSearchCV(estimator=clf, param\_distributions=random\_grid,
215. n\_iter=10, cv=3, verbose=0, random\_state=42, n\_jobs=1)# 训练集上进行参数搜索
216. clf\_random.fit(X\_train\_rf, y\_train\_rf)
217. print(clf\_random.best\_params\_)# 使用最佳参数构建最终模型并训练
218. best\_params = clf\_random.best\_params\_
219. model\_rf = RandomForestRegressor(\*\*best\_params)
220. model\_rf.fit(X\_train\_rf, y\_train\_rf)
221. y\_test\_pred\_rf = model\_rf.predict(X\_test\_rf)mse = mean\_squared\_error(y\_test\_rf, y\_test\_pred\_rf)
222. r2 = r2\_score(y\_test\_rf, y\_test\_pred\_rf)print(f"均方误差 (MSE): {mse}")
223. print(f"决定系数 (R²): {r2}")plot\_results(y\_test\_rf, y\_test\_pred\_rf, '随机森林回归 真实值与预测值对比')return model\_rf, mse, r2
224. import matplotlib
225. import matplotlib.pyplot as plt# 支持中文
226. matplotlib.rcParams['font.sans-serif'] = ['SimHei']
227. matplotlib.rcParams['axes.unicode\_minus'] = False
228. def plot\_results(y\_test\_contrast, y\_test\_pred\_contrast, title):
229. # plt.figure(figsize=(8, 6))
230. # plt.plot(y\_test\_contrast, label='真实值', marker='o', linestyle='-', color='green')
231. # plt.plot(y\_test\_pred\_contrast, label='预测值', marker='x', linestyle=':', color='blue')
232. # plt.xlabel('测试样本索引')
233. # plt.ylabel('睡眠效率')
234. # plt.title(title)
235. # plt.xticks(rotation=45)
236. # plt.grid(True)
237. # plt.legend()
238. # plt.tight\_layout()
239. # plt.show()plt.figure(figsize=(10, 6))
240. plt.scatter(y\_test\_contrast, y\_test\_pred\_contrast, edgecolor='k', alpha=0.7)
241. plt.plot([y\_test\_contrast.min(), y\_test\_contrast.max()], [y\_test\_contrast.min(), y\_test\_contrast.max()], 'k--', lw=3)
242. plt.xlabel('真实值')
243. plt.ylabel('预测值')
244. plt.title(title)
245. plt.savefig(f'pic/{title}.png', bbox\_inches='tight')
246. plt.show()
247. def plot\_loss\_curve(epoch\_loss, title):
248. plt.figure(figsize=(8, 6))
249. plt.plot(range(len(epoch\_loss)), epoch\_loss, label='训练损失')
250. plt.xlabel('Epoch')
251. plt.ylabel('Loss')
252. plt.title(title)
253. plt.grid(True)
254. plt.legend()
255. plt.tight\_layout()
256. plt.savefig(f'pic/{title}.png', bbox\_inches='tight')
257. plt.show()
258. import joblib
259. import numpy as np
260. from matplotlib import pyplot as plt
261. def model\_save(model, scaler, default\_values, filename='model/mf\_model.pkl'):
262. """
263. 保存模型、标准化器和默认值到文件。参数:
     * model: 训练好的模型
     * scaler: 用于数据标准化的标准化器
     * default\_values: 特征的默认值（均值）
     * filename: 保存文件的名称，默认 'mf\_model.pkl'
264. """
265. data\_to\_save = {
266. 'model': model,
267. 'scaler': scaler,
268. 'default\_values': default\_values
269. }
270. joblib.dump(data\_to\_save, filename)
271. print(f"模型和标准化器已保存到 {filename}")
272. def model\_load(filename='model/mf\_model.pkl'):
273. """
274. 从文件加载模型、标准化器和默认值。参数:
     * filename: 保存文件的名称，默认 'mf\_model.pkl'返回:
     * model: 加载的模型
     * scaler: 加载的标准化器
     * default\_values: 加载的特征默认值
275. """
276. data = joblib.load(filename)
277. print(f"从 {filename} 加载模型和标准化器")
278. return data['model'], data['scaler'], data['default\_values']
279. #
280. #
281. #
282. def predict\_sleep\_efficiency(age, gender, model, scaler, default\_values):
283. """
284. 使用模型预测特定年龄和性别的睡眠效率。参数:
     * age: 年龄
     * gender: 性别 (0: 男性, 1: 女性)
     * model: 训练好的xgb模型
     * scaler: 标准化器
     * default\_values: 特征默认值返回:
     * 预测的睡眠效率
285. """
286. new\_data = {
287. 'Age': age,
288. 'Gender': gender,
289. 'Sleep duration': default\_values['Sleep duration'],
290. 'REM sleep percentage': default\_values['REM sleep percentage'],
291. 'Deep sleep percentage': default\_values['Deep sleep percentage'],
292. 'Light sleep percentage': default\_values['Light sleep percentage'],
293. 'Awakenings': default\_values['Awakenings'],
294. 'Caffeine consumption': default\_values['Caffeine consumption'],
295. 'Alcohol consumption': default\_values['Alcohol consumption'],
296. 'Smoking status': default\_values['Smoking status'],
297. 'Exercise frequency': default\_values['Exercise frequency']
298. }new\_data\_values = np.array(list(new\_data.values())).reshape(1, -1)
299. new\_data\_scaled = scaler.transform(new\_data\_values)
300. predicted\_efficiency = model.predict(new\_data\_scaled)
301. return predicted\_efficiency[0]#
302. # def plot\_age\_vs\_efficiency(model, scaler, default\_values):
303. # ages = range(10, 71)
304. # male\_efficiency = [predict\_sleep\_efficiency(age, 0, model, scaler, default\_values) for age in ages]
305. # female\_efficiency = [predict\_sleep\_efficiency(age, 1, model, scaler, default\_values) for age in ages]
306. #
307. # plt.figure(figsize=(10, 6))
308. # plt.plot(ages, male\_efficiency, label='男性', color='blue')
309. # plt.plot(ages, female\_efficiency, label='女性', color='red')
310. # plt.xlabel('年龄')
311. # plt.ylabel('预测的睡眠效率')
312. # plt.title('年龄与睡眠效率预测')
313. # plt.legend()
314. # plt.grid(True)
315. # plt.savefig(f'pic/年龄与睡眠效率预测.png', bbox\_inches='tight')
316. # plt.show()
317. #
318. #
319. # plot\_age\_vs\_efficiency(model, scaler, default\_values)
320. from flask import Flask, request, render\_template, jsonify
321. import joblib
322. import numpy as npfrom SleepEfficiency.static.machine\_learning.model\_manage import predict\_sleep\_efficiency
323. from machine\_learning.linear\_torch import linear\_regression
324. from machine\_learning.main import load\_data
325. from machine\_learning.model\_manage import model\_save, model\_load
326. from machine\_learning.neural\_network import neural\_network\_regression
327. from machine\_learning.random\_forest import random\_forest\_regression
328. from machine\_learning.xgb\_re import xgboost\_regressionapp = Flask(\_\_name\_\_)
329. file\_path = 'static/machine\_learning/Sleep\_Efficiency.csv'
330. X\_train, X\_test, y\_train, y\_test, scaler, default\_values = load\_data(file\_path)# print("多元线性回归")# print("神经网络回归")# print("XGBoost回归")# print("随机森林回归")# 加载训练好的模型和标准化器
331. model, scaler, default\_values = model\_load('static/machine\_learning/model/mf\_model.pkl')
332. @app.route('/')
333. def index():
334. return render\_template('index.html')
335. @app.route('/predict', methods=['POST'])
336. def predict():
337. try:
338. age = int(request.form['age'])
339. gender = int(request.form['gender'])# 预测
340. prediction = predict\_sleep\_efficiency(age, gender, model, scaler, default\_values)return jsonify({'prediction': prediction})
341. except Exception as e:
342. return jsonify({'error': str(e)})
343. @app.route('/linear-regression')
344. def linearre():
345. linear\_mse, linear\_regression\_r2 = linear\_regression(X\_train, X\_test, y\_train, y\_test)
346. return render\_template('linear-regression.html', mse=linear\_mse, r2=linear\_regression\_r2)
347. @app.route('/neural-network')
348. def neuraln():
349. neural\_mse, neural\_r2 = neural\_network\_regression(X\_train, X\_test, y\_train, y\_test)
350. return render\_template('neural-network.html', mse=neural\_mse, r2=neural\_r2)
351. @app.route('/xgboost')
352. def xgbo():
353. xgboost\_mse, xgboost\_r2 = model\_xgb = xgboost\_regression(X\_train, X\_test, y\_train, y\_test)
354. return render\_template('xgboost.html', mse=xgboost\_mse, r2=xgboost\_r2)
355. @app.route('/random-forest')
356. def rf():
357. model\_rf, rf\_mse, rf\_r2 = random\_forest\_regression(X\_train, X\_test, y\_train, y\_test)
358. model\_save(model\_rf, scaler, default\_values)
359. return render\_template('random-forest.html', mse=rf\_mse, r2=rf\_r2)
360. @app.route('/others')
361. def other():
362. return render\_template('others.html')
363. if \_\_name\_\_ == '\_\_main\_\_':
364. app.run(debug=True)