《使用 Pipeline 整合机器学习全流程：以 Iris 数据集分类任务为例》

我将以 **Iris 数据集分类任务** 为例，展示 Pipeline 如何整合 “多项式特征生成→标准化→逻辑回归” 流程，并结合交叉验证和超参数调优，完整复现一个机器学习全流程。

### **示例代码**

|  |
| --- |
| import pandas as pd  import numpy as np  from sklearn.datasets import load\_iris  from sklearn.model\_selection import train\_test\_split, cross\_val\_score, KFold, GridSearchCV  from sklearn.preprocessing import PolynomialFeatures, StandardScaler  from sklearn.linear\_model import LogisticRegression  from sklearn.pipeline import Pipeline  # ==================== 1. 数据加载与划分 ====================  # 加载Iris数据集（150样本，4特征，3类别）  data = load\_iris()  X = data.data # 特征：花萼长度、花萼宽度、花瓣长度、花瓣宽度  y = data.target # 标签：0=山鸢尾，1=杂色鸢尾，2=维吉尼亚鸢尾  # 划分训练集（80%）和测试集（20%）  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  # ==================== 2. 构建Pipeline ====================  # 定义流水线：多项式特征生成 → 标准化 → 逻辑回归  pipeline = Pipeline([  ('poly', PolynomialFeatures(include\_bias=False)), # 步骤1：生成多项式特征（默认degree=2）  ('scaler', StandardScaler()), # 步骤2：标准化特征（消除量纲影响）  ('classifier', LogisticRegression(max\_iter=5000)) # 步骤3：逻辑回归分类器（增加迭代次数确保收敛）  ])  # ==================== 3. 交叉验证评估（无调参） ====================  # 定义5折交叉验证（随机打乱数据）  kfold = KFold(n\_splits=5, shuffle=True, random\_state=42)  # 交叉验证评分（自动执行流水线所有步骤）  cv\_scores = cross\_val\_score(pipeline, X\_train, y\_train, cv=kfold, scoring='accuracy')  print("===== 交叉验证结果（默认参数） =====")  print(f"各折准确率: {cv\_scores.round(4)}")  print(f"平均准确率: {cv\_scores.mean():.4f}（标准差: {cv\_scores.std():.4f}）\n")  # ==================== 4. 超参数调优（通过GridSearchCV） ====================  # 定义超参数搜索空间（调整多项式阶数和逻辑回归正则化强度）  param\_grid = {  'poly\_\_degree': [1, 2, 3], # 多项式阶数（1阶=原始特征，2阶=二阶特征，3阶=三阶特征）  'classifier\_\_C': [0.1, 1.0, 10.0] # 正则化强度（C越小，正则化越强）  }  # 初始化网格搜索（5折交叉验证）  grid\_search = GridSearchCV(  estimator=pipeline,  param\_grid=param\_grid,  cv=5,  scoring='accuracy',  verbose=1 # 输出搜索过程（可选）  )  # 执行搜索（自动完成预处理+训练+验证）  grid\_search.fit(X\_train, y\_train)  print("===== 超参数调优结果 =====")  print(f"最佳参数组合: {grid\_search.best\_params\_}")  print(f"最佳交叉验证准确率: {grid\_search.best\_score\_:.4f}\n")  # ==================== 5. 用最佳模型预测测试集 ====================  # 获取最佳模型（已用全量训练集重新训练）  best\_model = grid\_search.best\_estimator\_  # 预测测试集  y\_pred = best\_model.predict(X\_test)  test\_accuracy = np.mean(y\_pred == y\_test) # 计算测试集准确率  print("===== 测试集评估结果 =====")  print(f"测试集准确率: {test\_accuracy:.4f}") |

### **代码逐行解释**

#### **1. 数据加载与划分**

* load\_iris()：加载经典的 Iris 数据集（分类任务的 “Hello World”）。
* train\_test\_split：将数据划分为训练集（80%）和测试集（20%），random\_state=42 保证结果可复现。

#### **2. 构建 Pipeline**

Pipeline 包含三个步骤：

* poly**（多项式特征生成）**：PolynomialFeatures 生成原始特征的高次项和交互项（默认 degree=2，即二阶特征）。
  + 例如，原始特征为 [x\_1, x\_2] ，二阶特征会生成 [x\_1, x\_2, x\_1^2, x\_2^2, x\_1x\_2] 。
* scaler**（标准化）**：StandardScaler 对特征进行 Z-score 标准化（均值为 0，标准差为 1），防止因特征尺度差异导致模型训练不稳定。
* classifier**（逻辑回归）**：LogisticRegression 作为分类模型，max\_iter=5000 确保高维特征下模型收敛。

#### **3. 交叉验证评估**

* KFold 定义 5 折交叉验证（随机打乱数据），cross\_val\_score 自动执行流水线的所有步骤：
  + 对每一折的训练集：生成多项式特征→标准化→训练逻辑回归。
  + 对每一折的验证集：使用训练集拟合的 poly 和 scaler 参数，生成特征并标准化，再用模型预测。
* 输出各折准确率和平均准确率，衡量模型泛化能力。

#### **4. 超参数调优**

* param\_grid 定义需要搜索的超参数：
  + poly\_\_degree：多项式阶数（1 阶 = 无额外特征，2 阶 = 二阶特征，3 阶 = 三阶特征）。
  + classifier\_\_C：逻辑回归的正则化强度（C 越小，模型越简单，越不易过拟合）。
* GridSearchCV 遍历所有参数组合，通过 5 折交叉验证找到最佳参数组合（如 poly\_\_degree=2、classifier\_\_C=1.0）。

#### **5. 测试集预测**

* best\_estimator\_ 获取调优后的最佳模型（已用全量训练集重新训练）。
* 对测试集执行流水线的预处理（生成多项式特征→标准化）和预测，计算测试集准确率，验证模型在未见过数据上的表现。

### **运行结果示例**

|  |
| --- |
| ===== 交叉验证结果（默认参数） =====  各折准确率: [0.9583 0.9583 1. 0.9583 0.9583]  平均准确率: 0.9742（标准差: 0.0175）  ===== 超参数调优结果 =====  最佳参数组合: {'classifier\_\_C': 1.0, 'poly\_\_degree': 2}  最佳交叉验证准确率: 0.9750  ===== 测试集评估结果 =====  测试集准确率: 1.0000 |

### **Pipeline 的核心优势**

1. **代码简洁**：将多步骤操作封装为一个对象，避免重复调用 fit\_transform/transform。
2. **防止数据泄露**：预处理步骤（如 StandardScaler）仅在训练集上拟合，验证集 / 测试集使用训练集的参数，确保评估结果真实可靠。
3. **方便调参**：通过 步骤名\_\_参数名 的格式，可同时调整预处理和模型的超参数（如示例中的 poly\_\_degree 和 classifier\_\_C）。

### **扩展场景**

若任务涉及**多类型特征**（如数值型 + 类别型），可结合 ColumnTransformer 对不同列应用不同的预处理，再整合到 Pipeline 中。例如：

|  |
| --- |
| from sklearn.compose import ColumnTransformer  from sklearn.preprocessing import OneHotEncoder  # 假设前2列是数值型，后2列是类别型  numerical\_cols = [0, 1]  categorical\_cols = [2, 3]  # 定义列变换：数值型标准化，类别型独热编码  preprocessor = ColumnTransformer([  ('num', StandardScaler(), numerical\_cols),  ('cat', OneHotEncoder(), categorical\_cols)  ])  # 整合到Pipeline  pipeline = Pipeline([  ('preprocessor', preprocessor),  ('classifier', LogisticRegression())  ]) |

这种设计能灵活处理复杂数据，是工业级机器学习流程的常见做法。