#!/usr/bin/env python  
# coding: utf-8  
  
# 导入需要的各种包  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
from sklearn import preprocessing, metrics, svm  
from sklearn.model\_selection import train\_test\_split, GridSearchCV  
import xgboost as xgb  
  
# 读取数据  
data\_train = pd.read\_csv(r"C:\Users\阿韩想养二哈\Desktop\数模经典数据集\train.csv")  
  
# 数据统计  
print(data\_train['Survived'].describe())  
# 画图  
plt.rcParams['font.sans-serif'] = ['SimHei'] # 用来正常显示中文标签  
plt.rcParams['axes.unicode\_minus'] = False # 用来正常显示负号  
  
sns.distplot(data\_train['Survived'], color="r")  
plt.title("Survived直方图")  
plt.show()  
  
print("Skewness: %f" % data\_train['Survived'].skew())  
print("Kurtosis: %f" % data\_train['Survived'].kurt())  
  
var = 'Pclass'  
data = pd.concat([data\_train['Survived'], data\_train[var]], axis=1)  
fig = sns.boxplot(x=var, y="Survived", data=data)  
fig.axis(ymin=0, ymax=800000)  
plt.title("Pclass 总体评价箱型图")  
plt.show()  
  
var = 'Age'  
data = pd.concat([data\_train['Survived'], data\_train[var]], axis=1)  
data.plot.scatter(x=var, y="Survived", ylim=(0, 800000))  
plt.title("Age 散点图")  
plt.show()  
# 相关性分析  
df\_tmp1 = data\_train[  
 ['Pclass','Age','SibSp','Parch','Fare','Survived']]  
plt.rcParams['font.sans-serif'] = ['SimHei'] # 指定默认字体  
plt.rcParams['axes.unicode\_minus'] = False # 解决保存图像是负号'-'显示为方块的问题  
sns.heatmap(df\_tmp1.corr(), cmap="YlGnBu", annot=True)  
plt.title("相关性分析图")  
plt.show()  
# 特征工程  
cols = ['Pclass','Age','SibSp','Parch','Fare']  
x = data\_train[cols].values  
y = data\_train['Survived'].values  
X\_train, X\_validation, y\_train, y\_validation = train\_test\_split(x, y, test\_size=0.2, random\_state=42)  
ss\_X = preprocessing.StandardScaler()# 标准化处理  
ss\_Y = preprocessing.StandardScaler()  
X\_train\_scaled = ss\_X.fit\_transform(X\_train)  
y\_train\_scaled = ss\_Y.fit\_transform(y\_train.reshape(-1, 1))  
print(X\_train\_scaled)  
X\_validation\_scaled = ss\_X.transform(X\_validation)  
y\_validation\_scaled = ss\_Y.transform(y\_validation.reshape(-1, 1))  
  
import optuna  
from xgboost import XGBRegressor  
from sklearn.model\_selection import cross\_val\_score  
import matplotlib.pyplot as plt  
  
  
# 定义贝叶斯优化目标函数  
def objective(trial):  
 param = {  
 'learning\_rate': trial.suggest\_uniform('learning\_rate', 0.01, 0.3),  
 'n\_estimators': trial.suggest\_int('n\_estimators', 100, 800),  
 'max\_depth': trial.suggest\_int('max\_depth', 2, 5),  
 'reg\_alpha': trial.suggest\_uniform('reg\_alpha', 0.00001, 5),  
 'reg\_lambda': trial.suggest\_uniform('reg\_lambda', 0.00001, 5),  
 'min\_child\_weight': trial.suggest\_uniform('min\_child\_weight', 1, 100),  
 'subsample': trial.suggest\_uniform('subsample', 0.6, 1.0),  
 'colsample\_bytree': trial.suggest\_uniform('colsample\_bytree', 0.6, 1.0),  
 'colsample\_bynode': trial.suggest\_uniform('colsample\_bynode', 0.6, 1.0),  
 'colsample\_bylevel': trial.suggest\_uniform('colsample\_bylevel', 0.6, 1.0),  
 'gamma': trial.suggest\_uniform('gamma', 0.0, 0.3),  
 'scale\_pos\_weight': trial.suggest\_uniform('scale\_pos\_weight', 1.0, 1.5)  
 }  
  
 # 使用贝叶斯优化参数创建XGBoost模型  
 xgb\_model = XGBRegressor(\*\*param, nthread=10, n\_jobs=10, objective='reg:squarederror', booster='gbtree',  
 random\_state=7)  
  
 # 在训练集上进行交叉验证  
 xgb\_train = xgb.DMatrix(X\_train, y\_train)  
 result\_train = cross\_val\_score(xgb\_model, X\_train, y\_train, cv=5, scoring='neg\_root\_mean\_squared\_error')  
 result\_test = cross\_val\_score(xgb\_model, X\_validation, y\_validation, cv=5, scoring='neg\_root\_mean\_squared\_error')  
 return -result\_train.mean()-result\_test.mean() # 可以根据需要调整优化目标，例如只考虑训练误差或测试误差等  
  
  
# 设置贝叶斯优化参数和样本数  
study = optuna.create\_study()  
study.optimize(objective, n\_trials=100) # 根据需要调整样本数  
  
# 分析结果  
print("Best parameters:", study.best\_params)  
print("Best value:", study.best\_value)  
  
# 训练最优模型  
best\_param = study.best\_params  
best\_model = XGBRegressor(\*\*best\_param, nthread=10, n\_jobs=10, objective='reg:squarederror', booster='gbtree', random\_state=7)  
best\_model.fit(X\_train, y\_train)  
  
# 拟合  
best\_model.fit(X\_train\_scaled, y\_train\_scaled)  
y\_validation\_pred = best\_model.predict(X\_validation\_scaled) # 预测  
  
# 画图  
plt.plot(range(y\_validation\_scaled.shape[0]), y\_validation\_scaled, color="blue", linewidth=1.5, linestyle="-")  
plt.plot(range(y\_validation\_pred.shape[0]), y\_validation\_pred, color="red", linewidth=1.5, linestyle="-.")  
plt.legend(['真实值', '预测值'])  
plt.title("真实值与预测值比对图")  
plt.show() #显示图片  
# 模型评估  
print('可解释方差值：{}'.format(round(metrics.explained\_variance\_score(y\_validation\_scaled, y\_validation\_pred), 2)))  
print('平均绝对误差：{}'.format(round(metrics.mean\_absolute\_error(y\_validation\_scaled, y\_validation\_pred), 2)))  
print('均方误差：{}'.format(round(metrics.mean\_squared\_error(y\_validation\_scaled, y\_validation\_pred), 2)))  
print('R方值：{}'.format(round(metrics.r2\_score(y\_validation\_scaled, y\_validation\_pred), 2)))  
  
# 显示重要特征  
importances = list(best\_model.feature\_importances\_)  
data\_tmp=data\_train.drop(columns='Survived')  
feature\_list = list(data\_tmp.columns)  
  
feature\_importances = [(feature, round(importance, 2)) for feature, importance in zip(feature\_list, importances)]  
feature\_importances = sorted(feature\_importances, key=lambda x: x[1], reverse=True)  
  
import matplotlib.pyplot as plt  
  
x\_values = list(range(len(importances)))  
print(x\_values)  
plt.bar(x\_values, importances, orientation='vertical')  
plt.xticks(x\_values, feature\_list, rotation=6)  
plt.ylabel('Importance')  
plt.xlabel('Variable')  
plt.title('Variable Importances')  
plt.show()  
# 对测试数据进行预测  
cols = ['Pclass','Age','SibSp','Parch','Fare']  
  
data\_test = pd.read\_csv(r"C:\Users\阿韩想养二哈\Desktop\数模经典数据集\test.csv")  
print(data\_test[cols].isnull().sum())  
'''  
mean\_GarageCars = data\_test['B'].mean()  
mean\_TotalBsmtSF = data\_test['C'].mean()  
data\_test['B'].fillna(mean\_GarageCars, inplace=True)  
data\_test['C'].fillna(mean\_TotalBsmtSF, inplace=True)  
'''  
  
x\_test = data\_test.values  
x\_test\_scaled = ss\_X.transform(x\_test)  
y\_test\_pred = best\_model.predict(x\_test\_scaled)  
y\_test\_pred = y\_test\_pred.reshape(-1, 1)  
data\_test['Survived\_Pred'] = ss\_Y.inverse\_transform(y\_test\_pred)  
# data\_test.to\_excel('test\_pred.xlsx')  
print('Predict:')  
print(data\_test)