### 案例 10：基于 LightGBM-Stacking 模型的互联网企业用户流失预测

* **问题背景**：用户是互联网企业的核心资源，用户流失会导致企业收入下降和市场份额萎缩。准确预测用户流失风险，有助于企业采取针对性挽留措施，降低流失率。用户流失受用户使用频率、产品体验、竞争对手吸引、价格因素等多方面影响，且不同用户群体的流失原因差异较大。
* **问题描述**：某互联网社交平台需要预测未来 1 个月内可能流失的用户（定义为连续 15 天未登录的用户）。要求模型能够集成多种基模型的优势，提高预测的稳健性和准确性，识别出高流失风险用户，为平台制定个性化挽留策略提供支持。
* **数据情况**：提供平台过去 2 年的用户行为数据，包括用户的登录频率、在线时长、互动次数、好友数量、付费情况等 30 余项特征，以及用户是否流失的标签。数据量约 100 万用户，存在部分用户行为数据缺失的情况。

### 案例 10：LightGBM-Stacking 模型用户流失预测代码

|  |
| --- |
| import pandas as pd  import numpy as np  from sklearn.preprocessing import LabelEncoder, StandardScaler  from sklearn.model\_selection import train\_test\_split, KFold  from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier  from lightgbm import LGBMClassifier  from sklearn.linear\_model import LogisticRegression  from sklearn.metrics import roc\_auc\_score, classification\_report  import joblib  # 数据加载与预处理  data = pd.read\_csv('user\_churn.csv')  data = data.dropna()  # 类别特征编码  le = LabelEncoder()  for col in ['gender', 'subscription\_type']:  data[col] = le.fit\_transform(data[col])  # 特征与目标变量  X = data.drop(['user\_id', 'churn'], axis=1)  y = data['churn']  # 数据标准化  scaler = StandardScaler()  X\_scaled = scaler.fit\_transform(X)  # 划分训练集和测试集  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)  # 定义基模型  base\_models = [  ('lightgbm', LGBMClassifier(n\_estimators=100, random\_state=42)),  ('rf', RandomForestClassifier(n\_estimators=100, random\_state=42)),  ('gbt', GradientBoostingClassifier(n\_estimators=100, random\_state=42))  ]  # Stacking第一层：生成基模型预测  def stacking\_train(X, y, base\_models, n\_folds=5):  kf = KFold(n\_splits=n\_folds, shuffle=True, random\_state=42)  meta\_features = np.zeros((X.shape[0], len(base\_models)))    for i, (name, model) in enumerate(base\_models):  for train\_idx, val\_idx in kf.split(X):  X\_tr, X\_val = X[train\_idx], X[val\_idx]  y\_tr = y[train\_idx]  model.fit(X\_tr, y\_tr)  meta\_features[val\_idx, i] = model.predict\_proba(X\_val)[:, 1]  return meta\_features  # 生成训练集元特征  meta\_train = stacking\_train(X\_train, y\_train, base\_models)  # 训练元模型（逻辑回归）  meta\_model = LogisticRegression()  meta\_model.fit(meta\_train, y\_train)  # 生成测试集元特征  def stacking\_predict(X\_train, X\_test, y\_train, base\_models):  meta\_test = np.zeros((X\_test.shape[0], len(base\_models)))  for i, (name, model) in enumerate(base\_models):  model.fit(X\_train, y\_train)  meta\_test[:, i] = model.predict\_proba(X\_test)[:, 1]  return meta\_test  meta\_test = stacking\_predict(X\_train, X\_test, y\_train, base\_models)  # 最终预测  y\_pred\_proba = meta\_model.predict\_proba(meta\_test)[:, 1]  y\_pred = meta\_model.predict(meta\_test)  # 评估模型  print(f'AUC: {roc\_auc\_score(y\_test, y\_pred\_proba)}')  print(classification\_report(y\_test, y\_pred))  # 保存模型  joblib.dump(base\_models, 'base\_models\_stacking.pkl')  joblib.dump(meta\_model, 'meta\_model\_stacking.pkl')  joblib.dump(scaler, 'scaler\_churn.pkl') |