【机器学习基础】XGBoost、LightGBM与CatBoost算法对比与调参

机器学习初学者 1周前

以下文章来源于机器学习实验室 ， 作者louwill

机器学习实验室

统计学出身的深度学习算法工程师。进击的Coder。

机器学习

Author：louwill

Machine Learning Lab

虽然现在深度学习大行其道，但以XGBoost、LightGBM和CatBoost为代表的Boosting算法仍有其广阔的用武之地。抛开深度学习适用的图像、文本、语音和视频等非结构化的数据应用，Boosting算法对于训练样本较少的结构化数据领域仍然是第一选择。本文先对前述章节的三大Boosting的联系与区别进行简单阐述，并一个实际数据案例来对三大算法进行对比。然后对常用的Boosting算法超参数调优方法进行介绍，包括随机调参法、网格搜索法和贝叶斯调参法，并给出相应的代码示例。

三大Boosting算法对比

首先，XGBoost、LightGBM和CatBoost都是目前经典的SOTA（state of the art）Boosting算法，都可以归类到梯度提升决策树算法系列。三个模型都是以决策树为支撑的集成学习框架，其中XGBoost是对原始版本的GBDT算法的改进，而LightGBM和CatBoost则是在XGBoost基础上做了进一步的优化，在精度和速度上都有各自的优点。

三大模型的原理细节我们本文不做叙述，可参考【原创首发】机器学习公式推导与代码实现30讲.pdf。那么这三大Boosting算法又有哪些大的方面的区别呢？主要有两个方面。第一个是三个模型树的构造方式有所不同，XGBoost使用按层生长（level-wise）的决策树构建策略，LightGBM则是使用按叶子生长（leaf-wise）的构建策略，而CatBoost使用了对称树结构，其决策树都是完全二叉树。第二个有较大区别的方面是对于类别特征的处理。XGBoost本身不具备自动处理类别特征的能力，对于数据中的类别特征，需要我们手动处理变换成数值后才能输入到模型中；LightGBM中则需要指定类别特征名称，算法即可对其自动进行处理；

CatBoost以处理类别特征而闻名，通过目标变量统计等特征编码方式也能实现类别特征的高效处理。

下面我们以kaggle 2015年航班延误数据集为示例，分别用XGBoost、LightGBM和CatBoost模型进行实验。图1是flights数据集简介。

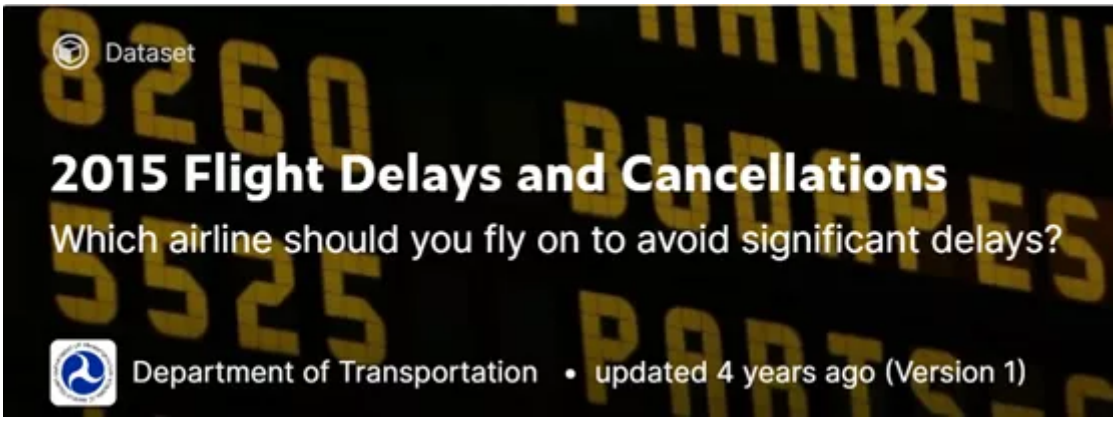

图2 flights数据集

该数据集完整数据量有500多万条航班记录数据，特征有31个，仅作演示用的情况下，我们采用抽样的方式从原始数据集中抽样1%的数据，并筛选11个特征，经过预处理后重新构建训练数据集，目标是构建对航班是否延误的二分类模型。数据读取和简单预处理过程如代码1所示。

代码1 数据处理

```
# 导入pandas和sklearn数据划分模块
import pandas as pd
from sklearn.model_selection import train_test_split
# 读取flights数据集
flights = pd.read_csv('flights.csv')
# 数据集抽样1%
flights = flights.sample(frac=0.01, random_state=10)
# 特征抽样，获取指定的11个特征
flights = flights[["MONTH", "DAY", "DAY_OF_WEEK", "AIRLINE",
"FLIGHT_NUMBER","DESTINATION_AIRPORT", "ORIGIN_AIRPORT","AIR_TIME",
"DEPARTURE_TIME", "DISTANCE", "ARRIVAL_DELAY"]]
# 对标签进行离散化，延误10分钟以上才算延误
flights["ARRIVAL_DELAY"] = (flights["ARRIVAL_DELAY"]>10)*1
# 类别特征
cat_cols = ["AIRLINE", "FLIGHT_NUMBER", "DESTINATION_AIRPORT",
"ORIGIN_AIRPORT"]
# 类别特征编码
for item in cat_cols:
```

```
        flights[item] = flights[item].astype("category").cat.codes +1
    # 数据集划分
X_train, X_test, y_train, y_test = train_test_split(
        flights.drop(["ARRIVAL_DELAY"], axis=1),
        flights["ARRIVAL_DELAY"],
        random_state=10, test_size=0.3)
# 打印划分后的数据集大小
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
```

输出：

```
(39956, 10) (39956,) (17125, 10) (17125,)
```

在代码1中，我们先读取了flights原始数据集，因为原始数据集量太大，我们对其进行抽样1%，并筛选了11个特征，构建有57081条1、11个特征的航班记录数据集。然后对抽样数据集进行简单的预处理，先对训练标签进行二值离散化，延误大于10分钟的转化为1（延误），延误小于10分钟的转化为0（不延误），然后对“航线”、“航班号”、“目的地机场”、“出发地机场”等类别特征进行类别编码处理。最后划分数据集，得到有39956条训练样本，17125条测试样本。

XGBoost

下面我们开始来测试三个模型在该数据集上的效果。先来看XGBoost，如代码2所示。

代码2 XGBoost

```
# 导入xgboost模块
import xgboost as xgb
# 导入模型评估auc函数
from sklearn.metrics import roc_auc_score
# 设置模型超参数
params = {
    'booster': 'gbtree',
    'objective': 'binary:logistic',
    'gamma': 0.1,
    'max_depth': 8,
    'lambda': 2,
    'subsample': 0.7,
    'colsample_bytree': 0.7,
    'min_child_weight': 3,
```

```
    'eta': 0.001,
    'seed': 1000,
    'nthread': 4,
}
# 封装xgboost数据集
dtrain = xgb.DMatrix(X_train, y_train)
# 训练轮数，即树的棵数
num_rounds = 500
# 模型训练
model_xgb = xgb.train(params, dtrain, num_rounds)
# 对测试集进行预测
dtest = xgb.DMatrix(X_test)
y_pred = model_xgb.predict(dtest)
print('AUC of testset based on XGBoost: ', roc_auc_score(y_test, y_pred))
```

输出：

```
AUC of testset based on XGBoost: 0.6845368959487046
```

在代码15-2中，我们测试了XGBoost在flights数据集上的表现，导入相关模块并设置模型超参数，便可基于训练集进行XGBoost模型拟合，最后将训练好的模型用于测试集预测，可得到测试集AUC为0.6845。

LightGBM

LightGBM在flights数据集上的测试过程如代码3所示。

代码3 LightGBM

```
# 导入lightgbm模块
import lightgbm as lgb
dtrain = lgb.Dataset(X_train, label=y_train)
params = {
"max_depth": 5,
"learning_rate" : 0.05,
"num_leaves": 500,
"n_estimators": 300
}

# 指定类别特征
cate_features_name = ["MONTH","DAY","DAY_OF_WEEK","AIRLINE",
```

```
"DESTINATION_AIRPORT", "ORIGIN_AIRPORT"]
  # lightgbm模型拟合
model_lgb = lgb.train(params, d_train,
categorical_feature = cate_features_name)
  # 对测试集进行预测
y_pred = model_lgb.predict(X_test)
print('AUC of testset based on XGBoost: 'roc_auc_score(y_test, y_pred))
```

输出：

```
AUC of testset based on XGBoost: 0.6873707383550387
```

在代码3中，我们测试了LightGBM在flights数据集上的表现，导入相关模块并设置模型超参数，便可基于训练集进行LightGBM模型拟合，最后将训练好的模型用于测试集预测，可得到测试集AUC为0.6873，跟XGBoost效果差不多。

CatBoost

CatBoost在flights数据集上的测试过程如代码4所示。

代码4 CatBoost

```
# 导入lightgbm模块
import catboost as cb
# 类别特征索引
cat_features_index = [0,1,2,3,4,5,6]
# 创建catboost模型实例
model_cb = cb.CatBoostClassifier(eval_metric="AUC",
  one_hot_max_size=50, depth=6, iterations=300, l2_leaf_reg=1,
  learning_rate=0.1)
# catboost模型拟合
model_cb.fit(X_train, y_train, cat_features=cat_features_index)
# 对测试集进行预测
y_pred = model_cb.predict(X_test)
print('AUC of testset based on CatBoost: 'roc_auc_score(y_test, y_pred))
```

输出：

```
AUC of testset based on CatBoost: 0.5463773041667715
```

在代码4中，我们测试了CatBoost在flights数据集上的表现，导入相关模块并设置模型超参数，便可基于训练集进行CatBoost模型拟合，最后将训练好的模型用于测试集预测，可得到测试集AUC为0.54，相较于XGBoost和LightGBM，CatBoost在该数据集上的效果要差不少。表1是针对flights数据集三大模型的综合对比结果。

表 1 三大模型性能对比结果

	XGBoost	LightGBM	CatBoost
基本超参数	max_depth: 8, lambda: 2, subsample: 0.7, colsample_bytree: 0.7, min_child_weight: 3 n_estimator: 500	max_depth: 5, learning_rate: 0.05, num_leaves: 500, n_estimators: 300	one_hot_max_size=10, depth=4, iterations=300, l2_leaf_reg=1, learning_rate=0.1
训练集 AUC	0.7516	0.8812	0.5735
测试集 AUC	0.6845	0.6874	0.5464
训练时间/(s)	21.95	2.18	37.28
测试时间/(s)	0.23	0.51	0.28

从表1的综合对比结果来看，LightGBM无论是在精度上还是速度上，都要优于XGBoost和CatBoost。当然了，我们只是在数据集上直接用三个模型做了比较，没有做进一步的数据特征工程和超参数调优，表1的结果均可做进一步的优化。

常用的超参数调优方法

机器学习模型中有大量需要事先进行人为设定的参数，比如说神经网络训练的batch-size，XGBoost等集成学习模型的树相关参数，我们将这类不是经过模型训练得到的参数叫做超参数（hyperparameter）。人为的对超参数调整的过程也就是我们熟知的调参。机器学习中常用的调参方法包括网格搜索法（grid search）、随机搜索法（random search）和贝叶斯优化（bayesian optimization）。

网格搜索法

网格搜索是一项常用的超参数调优方法，常用于优化三个或者更少数量的超参数，本质是一种穷举法。对于每个超参数，使用者选择一个较小的有限集去探索。然后，这些超参数笛卡尔乘积得到若干组超参数。网格搜索使用每组超参数训练模型，挑选验证集误差最小的超参数作为最好的超参数。

例如，我们有三个需要优化的超参数a,b,c，候选的取值分别是{1,2}，{3,4}，{5,6}。则所有可能的参数取值组合组成了一个8个点的3维空间网格如下：{（1,3,5），（1,3,6），（1,4,5），（1,4,6），（2,3,5），（2,3,6），（2,4,5），（2,4,6）}，网格搜索就是通过遍历这8个可能的参数取值组合，进行训练和验证，最终得到最优超参数。

Sklearn中通过model_selection模块下的GridSearchCV来实现网格搜索调参，并且这个调参过程是加了交叉验证的。我们同样以前述flights数据集为例，展示XGBoost的网格搜索代码示例。

代码5 网格搜索

```
### 基于XGBoost的GridSearch搜索范例
# 导入GridSearch模块
from sklearn.model_selection import GridSearchCV
# 创建xgb分类模型实例
model = xgb.XGBClassifier()
# 待搜索的参数列表空间
param_lst = {"max_depth": [3,5,7],
             "min_child_weight" : [1,3,6],
             "n_estimators": [100,200,300],
             "learning_rate": [0.01, 0.05, 0.1]
            }
# 创建网格搜索
grid_search = GridSearchCV(model, param_grid=param_lst, cv=3,
                                  verbose=10, n_jobs=-1)
# 基于flights数据集执行搜索
grid_search.fit(X_train, y_train)
# 输出搜索结果
print(grid_search.best_estimator_)
```

输出：

```
XGBClassifier(max_depth=5, min_child_weight=6, n_estimators=300)
```

代码5给出了基于XGBoost的网格搜索范例。我们先创建XGBoost分类模型实例，然后给出需要搜索的参数和对应的参数范围列表，并基于GridSearch创建网格搜索对象，最后拟合训练数据，输出网格搜索的参数结果。可以看到，当树最大深度为5、最小子树权重取6以及树的棵数为300时，模型能达到相对最优的效果。

随机搜索

随机搜索，顾名思义，即在指定的超参数范围或者分布上随机搜索和寻找最优超参数。相较于网格搜索方法，给定超参数分布内并不是所有的超参数都会进行尝试，而是会从给定分布中抽样一个固定数量的参数，实际仅对这些抽样到的超参数进行实验。相较于网格搜索，随机搜索有时候会是一种更高效的调参方法。Sklearn中通过model_selection模块下

RandomizedSearchCV方法进行随机搜索。基于XGBoost的随机搜索调参示例如代码6所示。

代码6 随机搜索

```
### 基于XGBoost的GridSearch搜索范例
# 导入GridSearch模块
from sklearn.model_selection import GridSearchCV
# 创建xgb分类模型实例
model = xgb.XGBClassifier()
# 待搜索的参数列表空间
param_lst = {"max_depth": [3,5,7],
             "min_child_weight" : [1,3,6],
             "n_estimators": [100,200,300],
             "learning_rate": [0.01, 0.05, 0.1]
            }
# 创建网格搜索
grid_search = GridSearchCV(model, param_grid=param_lst, cv=3,
                                  verbose=10, n_jobs=-1)
# 基于flights数据集执行搜索
grid_search.fit(X_train, y_train)
# 输出搜索结果
print(grid_search.best_estimator_)
```

输出：

```
XGBClassifier(max_depth=5, min_child_weight=6, n_estimators=300)
```

代码6给出了随机搜索的使用示例，模式上跟网格搜索基本一致，可以看到，随机搜索的结果认为树的棵树取300，最小子树权重为6，最大深度为5，学习率取0.1的时候模型达到最优。

贝叶斯调参

除了上述两种调参方法外，本小节介绍第三种，也有可能是最好的一种调参方法，即贝叶斯优化。贝叶斯优化是一种基于高斯过程（gaussian process）和贝叶斯定理的参数优化方法，近年来被广泛用于机器学习模型的超参数调优。这里不详细探讨高斯过程和贝叶斯优化的数学原理，仅展示贝叶斯优化的基本用法和调参示例。

贝叶斯优化其实跟其他优化方法一样，都是为了为了求目标函数取最大值时的参数值。作为一个序列优化问题，贝叶斯优化需要在每一次迭代时选取一个最佳观测值，这是贝叶斯优化的关键问题。而这个关键问题正好被上述的高斯过程完美解决。关于贝叶斯优化的大量数学原理，

包括高斯过程、采集函数、Upper Confidence Bound（UCB）和Expectation Improvements（EI）等概念原理，本节限于篇幅不做展开描述。贝叶斯优化可直接借用现成的第三方库BayesianOptimization来实现。使用示例如代码7所示。

代码7 贝叶斯优化

```
### 基于XGBoost的BayesianOptimization搜索范例
# 导入xgb模块
import xgboost as xgb
# 导入贝叶斯优化模块
from bayes_opt import BayesianOptimization
# 定义目标优化函数
def xgb_evaluate(min_child_weight,
                 colsample_bytree,
                 max_depth,
                 subsample,
                 gamma,
                 alpha):
    # 指定要优化的超参数
    params['min_child_weight'] = int(min_child_weight)
    params['cosample_bytree'] = max(min(colsample_bytree, 1), 0)
    params['max_depth'] = int(max_depth)
    params['subsample'] = max(min(subsample, 1), 0)
    params['gamma'] = max(gamma, 0)
    params['alpha'] = max(alpha, 0)
    # 定义xgb交叉验证结果
    cv_result = xgb.cv(params, dtrain, num_boost_round=num_rounds, nfold=5,
                 seed=random_state,
                 callbacks=[xgb.callback.early_stop(50)])
    return cv_result['test-auc-mean'].values[-1]

# 定义相关参数
num_rounds = 3000
random_state = 2021
num_iter = 25
init_points = 5
params = {
    'eta': 0.1,
    'silent': 1,
    'eval_metric': 'auc',
    'verbose_eval': True,
```

```
    'seed': random_state
}
# 创建贝叶斯优化实例
# 并设定参数搜索范围
xgbBO = BayesianOptimization(xgb_evaluate,
                             {'min_child_weight': (1, 20),
                              'colsample_bytree': (0.1, 1),
                              'max_depth': (5, 15),
                              'subsample': (0.5, 1),
                              'gamma': (0, 10),
                              'alpha': (0, 10),
                               })
# 执行调优过程
xgbBO.maximize(init_points=init_points, n_iter=num_iter)
```

代码7给出了基于XGBoost的贝叶斯优化示例，在执行贝叶斯优化前，我们需要基于XGBoost的交叉验证xgb.cv定义一个待优化的目标函数，获取xgb.cv交叉验证结果，并以测试集AUC为优化时的精度衡量指标。最后将定义好的目标优化函数和超参数搜索范围传入贝叶斯优化函数BayesianOptimization中，给定初始化点和迭代次数，即可执行贝叶斯优化。

```
| 22       | 0.7069   | 5.509    | 1.0      | 0.0      | 15.0     | 1.0      | 1.0      |
Multiple eval metrics have been passed: 'test-auc' will be used for early stopping.

Will train until test-auc hasn't improved in 50 rounds.
Stopping. Best iteration:
[313]   train-auc:0.844097+0.00169582   test-auc:0.717143+0.00497509

| 23       | 0.7171   | 4.099    | 0.1      | 0.0      | 5.0      | 5.377    | 1.0      |
Multiple eval metrics have been passed: 'test-auc' will be used for early stopping.

Will train until test-auc hasn't improved in 50 rounds.
Stopping. Best iteration:
[121]   train-auc:0.861537+0.000673411  test-auc:0.704227+0.00621222

| 24       | 0.7042   | 10.0     | 0.1      | 0.0      | 15.0     | 20.0     | 0.5      |
Multiple eval metrics have been passed: 'test-auc' will be used for early stopping.
```

图2 贝叶斯优化结果

部分优化过程如图2所示，可以看到，贝叶斯优化在第23次迭代时达到最优，当alpha参数取4.099、列抽样比例为0.1、gamma参数为0、树最大深度为5、最小子树权重取5.377以及子抽样比例为1.0时，测试集AUC达到最优的0.72。

总结

本章是在前述几章集成学习内容基础上的一个简单综合对比，并给出了集成学习常用的超参数调优方法和示例。我们针对常用的三大Boosting集成学习模型：XGBoost、LightGBM和

CatBoost，以具体的数据实例做了一个精度和速度上的性能对比，但限于具体的数据集和调优差异，对比结果仅作为演示说明使用，并不能真正代表LightGBM模型一定就要优于CatBoost模型。

三大常用的超参数调优方法：网格搜索法、随机搜索法和贝叶斯优化法。本章也基于同样的数据集给出了三大超参数调优方法的使用示例，但限于篇幅，并没有太多深入每个方法的数学原理阐述。