

# kaggle竞赛大杀器：XGBoost图解

数据派THU 2023-03-20 17:00 发表于北京

清华大数据软件团队官方微信公众号

DataPi THU, Share and Study

来源：机器学习算法那些事

本文**约8400字**，建议阅读**10分钟**  
本文详细讲解 XGBoost 的工程应用方法。

图解机器学习实战以案例和代码驱动的方式展示机器学习算法的应用流程&链条各环节，掌握构建场景建模解决方案并进行效果调优的能力。

本文详细讲解 XGBoost 的工程应用方法。XGBoost 是一个非常强大的 Boosting 算法工具包，是很多大厂机器学习方案的模型首选，在并行计算效率、缺失值处理、控制过拟合等能力上都表现非常优秀。



<https://www.showmeai.tech/article-detail/204>

XGBoost 是 eXtreme Gradient Boosting 的缩写称呼，它是一个非常强大的 Boosting 算法工具包，优秀的性能（效果与速度）让其在很长时间内霸屏数据科学比赛解决方案榜首，现在很多大厂的机器学习方案依旧会首选这个模型。XGBoost 在并行计算效率、缺失值处理、控制过拟合、预测泛化能力上都表现非常优秀。

## 1.XGBoost安装

XGBoost作为常见的强大Python机器学习工具库，安装也比较简单。

### Python与IDE环境设置



Python 环境与 IDE 设置可以参考ShowMeAI文章 图解python | 安装与环境设置[2] 进行设置。

## 工具库安装

### (1) Linux/Mac等系统

这些系统下的XGBoost安装，大家只要基于pip就可以轻松完成了，在命令行端输入命令如下命令即可等待安装完成。

```
pip install xgboost
```

大家也可以选择国内的pip源，以获得更好的安装速度

```
pip install -i https://pypi.tuna.tsinghua.edu.cn/simple xgboost
```



### (2) Windows系统

对于windows系统而言，比较高效便捷的安装方式是：在网址 <http://www.lfd.uci.edu/~gohlke/pythonlibs/> 中去下载对应版本的XGBoost安装包，再通过如下命令安装。

```
pip install xgboost-1.5.1-cp310-cp310-win32.whl
```

## 2.XGBoost数据读取

应用 XGBoost 的第一步，需要加载所需的数据成为工具库所能支持的格式形态。XGBoost 可以加载多种数据格式的数据用于训练建模：

- libsvm 格式的文本数据。
- Numpy 的二维数组。
- XGBoost 的二进制的缓存文件。加载的数据存储在对象 DMatrix 中。

XGBoost 的 SKLearn 接口也支持对于 Dataframe 格式的数据(参考ShowMeAI的文章 [Python数据分析|Pandas核心操作函数大全\[3\]](#) 进行处理。

下面是不同格式的数据，XGBoost 的加载方式。

- 加载libsvm格式的数据

```
dtrain1 = xgb.DMatrix('train.svm.txt')
```

- 加载二进制的缓存文件

```
dtrain2 = xgb.DMatrix('train.svm.buffer')
```

- 加载 numpy 的数组

```
data = np.random.rand(5,10) # 5 entities, each contains 10 features
label = np.random.randint(2, size=5) # binary target
dtrain = xgb.DMatrix( data, label=label)
```

- 将 **scipy.sparse** 格式的数据转化为 **DMatrix** 格式

```
csr = scipy.sparse.csr_matrix( (dat, (row,col)) )
dtrain = xgb.DMatrix( csr )
```

- 将 **DMatrix** 格式的数据保存成 XGBoost 的二进制格式，在下次加载时可以提高加载速度，使用方式如下

```
dtrain = xgb.DMatrix('train.svm.txt')
dtrain.save_binary("train.buffer")
```

- 可以用如下方式处理 **DMatrix** 中的缺失值

```
dtrain = xgb.DMatrix( data, label=label, missing = -999.0)
```

- 当需要给样本设置权重时，可以用如下方式

```
w = np.random.rand(5,1)
dtrain = xgb.DMatrix( data, label=label, missing = -999.0, weight=w)
```

### 3.XGBoost不同建模方式

内置建模方式：libsvm格式数据源

XGBoost 内置了建模方式，有如下的数据格式与核心训练方法：

- 基于DMatrix格式的数据
- 基于xgb.train接口训练

下面是官方的一个简单示例，演示了读取libsvm格式数据(成DMatrix格式)并指定参数建模的过程。

```
# 导入工具箱
import numpy as np
import scipy.sparse
import pickle
import xgboost as xgb

# 从Libsvm文件中读取数据，做二分类
# 数据是Libsvm的格式，如下样本格式
#1 3:1 10:1 11:1 21:1 30:1 34:1 36:1 40:1 41:1 53:1 58:1 65:1 69:1 77:1 86:1 88:1 92:1 95:1 1
#0 3:1 10:1 20:1 21:1 23:1 34:1 36:1 39:1 41:1 53:1 56:1 65:1 69:1 77:1 86:1 88:1 92:1 95:1 1
#0 1:1 10:1 19:1 21:1 24:1 34:1 36:1 39:1 42:1 53:1 56:1 65:1 69:1 77:1 86:1 88:1 92:1 95:1 1

dtrain = xgb.DMatrix('./data/agaricus.txt.train')
dtest = xgb.DMatrix('./data/agaricus.txt.test')

# 超参数设定
# 主要是树深、学习率、目标函数
```

```

param = {'max_depth':2, 'eta':1, 'silent':1, 'objective':'binary:logistic' }

# 设定watchlist用于建模过程中观测模型状态
watchlist = [(dtest,'eval'), (dtrain,'train')]

num_round = 2
bst = xgb.train(param, dtrain, num_round, watchlist)

# 使用模型预测
preds = bst.predict(dtest)

# 判断准确率
labels = dtest.get_label()
print('错误率为%f' % \
      (sum(1 for i in range(len(preds)) if int(preds[i]>0.5)!=labels[i]) /float(len(preds))))

# 模型存储
bst.save_model('./model/0001.model')

```



**XGBoost**

工具库建模应用详解

**不同建模方式**

内置建模方式-libsvm

<http://www.showmeai.tech/>

```

dtrain = xgb.DMatrix('./data/agaricus.txt.train')
dtest = xgb.DMatrix('./data/agaricus.txt.test')

# 超参数设定
# 主要是树深、学习率、目标函数
param = {'max_depth':2, 'eta':1, 'silent':1, 'objective':'binary:logistic' }

# 设定watchlist用于建模过程中观测模型状态
watchlist = [(dtest,'eval'), (dtrain,'train')]
num_round = 2
bst = xgb.train(param, dtrain, num_round, watchlist)

# 使用模型预测
preds = bst.predict(dtest)

```

ShowMeAI

[搜索 | 微信 ShowMeAI 研究中心](#)

```

[0] eval-error:0.042831 train-error:0.046522
[1] eval-error:0.021726 train-error:0.022263
错误率为0.021726

```

### 内置建模方式：csv格式数据源


下面的例子，输入的数据源是csv文件，我们使用大家熟悉的 Pandas 工具库(参考ShowMeAI教程 数据分析系列教程[4] 与 数据科学工具速查 | Pandas使用指南[5])把数据读取为 Dataframe 格式，再构建Dmatrix格式输入，后续使用内置建模方式进行训练。

```

# 皮马印第安人糖尿病数据集 包含很多字段：怀孕次数 口服葡萄糖耐量试验中血浆葡萄糖浓度 舒张压(mm Hg) 三头肌
# 2小时血清胰岛素(μU/ mL) 体重指数(kg/(身高(m)^2) 糖尿病系统功能 年龄(岁)

import pandas as pd
data = pd.read_csv('./data/Pima-Indians-Diabetes.csv')
data.head()

```




## XGBoost

### 工具库建模应用详解

不同建模方式

内置建模方式-csv

 <http://www.showmeai.tech/>

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

ShowMeAI 研究中心

```
# 导入工具库
import numpy as np
import pandas as pd
import pickle
import xgboost as xgb
from sklearn.model_selection import train_test_split

# 用pandas读入数据
data = pd.read_csv('./data/Pima-Indians-Diabetes.csv')

# 做数据切分
train, test = train_test_split(data)

# 转换成Dmatrix格式
feature_columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']
target_column = 'Outcome'

# 取出Dataframe的numpy数组值去初始化DMatrix对象
xgtrain = xgb.DMatrix(train[feature_columns].values, train[target_column].values)
xgtest = xgb.DMatrix(test[feature_columns].values, test[target_column].values)

# 参数设定
param = {'max_depth':5, 'eta':0.1, 'silent':1, 'subsample':0.7, 'colsample_bytree':0.7, 'objective':'binary:logistic'}

# 设定watchlist用于查看模型状态
watchlist = [(xgtest,'eval'), (xgtrain,'train')]
num_round = 10
bst = xgb.train(param, xgtrain, num_round, watchlist)

# 使用模型预测
preds = bst.predict(xgtest)

# 判断准确率
labels = xgtest.get_label()
print('错误类为%f' % \
      (sum(1 for i in range(len(preds)) if int(preds[i]>0.5)!=labels[i]) /float(len(preds))))

# 模型存储
bst.save_model('./model/0002.model')
```



**XGBoost**

工具库建模应用详解

**不同建模方式**

内置建模方式-csv

<http://www.showmeai.tech/>

```
# 取出numpy array去初始化DMatrix对象
xgtrain = xgb.DMatrix(train[feature_columns].values, train[target_column].values)
xgtest = xgb.DMatrix(test[feature_columns].values, test[target_column].values)

# 参数设定
param = {'max_depth':5, 'eta':0.1, 'silent':1, 'subsample':0.7, \
         'colsample_bytree':0.7, 'objective':'binary:logistic' }

# 设定watchlist用于查看模型状态
watchlist = [(xgtest,'eval'), (xgtrain,'train')]
num_round = 10
bst = xgb.train(param, xgtrain, num_round, watchlist)

# 使用模型预测
preds = bst.predict(xgtest)
```

 搜索 | 微信 ShowMeAI 研究中心

```
[0] eval-error:0.354167 train-error:0.194444
[1] eval-error:0.34375 train-error:0.170139
[2] eval-error:0.322917 train-error:0.170139
[3] eval-error:0.28125 train-error:0.161458
[4] eval-error:0.302083 train-error:0.147569
[5] eval-error:0.286458 train-error:0.138889
[6] eval-error:0.296875 train-error:0.142361
[7] eval-error:0.291667 train-error:0.144097
[8] eval-error:0.302083 train-error:0.130208
[9] eval-error:0.291667 train-error:0.130208
错误类为0.291667
```

### 预估器建模方式：SKLearn接口+Dataframe

XGBoost 也支持用 SKLearn 中统一的预估器形态接口进行建模如下为典型的参考案例，对于读取为 Dataframe 格式的训练集和测试集，可以直接使用 XGBoost初始化 XGBClassifier 进行 fit 拟合训练。使用方法与接口，和 SKLearn 中其他预估器一致。

```
# 导入工具库
import numpy as np
import pandas as pd
import pickle
import xgboost as xgb
from sklearn.model_selection import train_test_split

# 用pandas读入数据
data = pd.read_csv('./data/Pima-Indians-Diabetes.csv')

# 做数据切分
train, test = train_test_split(data)

# 特征列
feature_columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']

# 标签列
target_column = 'Outcome'

# 初始化模型
xgb_classifier = xgb.XGBClassifier(n_estimators=20, \
                                  max_depth=4, \
                                  learning_rate=0.1, \
                                  subsample=0.7, \
                                  colsample_bytree=0.7, \
                                  eval_metric='error')

# Dataframe格式数据拟合模型
xgb_classifier.fit(train[feature_columns], train[target_column])
```

```
# 使用模型预测
preds = xgb_classifier.predict(test[feature_columns])

# 判断准确率
print('错误类为%f' %((preds!=test[target_column]).sum()/float(test_y.shape[0])))

# 模型存储
joblib.dump(xgb_classifier, './model/0003.model')
```



XGBoost

工具库建模应用详解

不同建模方式

预估建模方式

<http://www.showmeai.tech/>

```
# 用pandas读入数据
data = pd.read_csv('./data/Pima-Indians-Diabetes.csv')

# 做数据切分
train, test = train_test_split(data)
...

# 初始化模型
xgb_classifier = xgb.XGBClassifier(n_estimators=20,max_depth=4,learning_rate=0.1,\
                                  subsample=0.7, olsample_bytree=0.7, eval_metric='error')

# Dataframe格式数据拟合模型
xgb_classifier.fit(train[feature_columns], train[target_column])

# 使用模型预测
preds = xgb_classifier.predict(test[feature_columns])
```

搜索 | 微信 ShowMeAI 研究中心

错误类为0.265625

[ './model/0003.model' ]

4. 模型调参与高级功能

XGBoost参数详解



XGBoost

工具库建模应用详解

模型调参与高级功能

XGBoost参数详解

<http://www.showmeai.tech/>

通用参数 General parameters	控制在提升(boosting)过程中使用哪种booster
提升器参数 Booster parameters	包含树模型booster和线性booster参数
任务参数 Task parameters	控制学习的场景

搜索 | 微信 ShowMeAI 研究中心

在运行XGBoost之前，必须设置三种类型参数：General parameters、Booster parameters和Task parameters：

通用参数：General parameters

该参数控制在提升(boosting)过程中使用哪种booster，常用的booster有树模型(tree)和线性模型(linear model)。

提升器参数：Booster parameters

这取决于使用哪种booster，包含树模型booster和线性booster参数。

任务参数：Task parameters

控制学习的场景，例如在回归问题中会使用不同的参数控制排序。

(1) 通用参数



**XGBoost**

工具库建模应用详解

模型调参与高级功能

XGBoost参数详解

<http://www.showmeai.tech/>

**通用参数**

<b>booster</b>	<b>silent</b>	<b>nthread</b>
决定使用基于树的模型或线性模型进行提升计算	0时表示打印出运行时信息 1表示以缄默方式运行	XGBoost运行时的线程数 缺省值是可获得最大线程数
<b>num_pbuffer</b>	<b>num_feature</b>	
预测缓冲区大小，通常设置为训练实例的数目。	Boosting过程中用到的特征维数，设置为特征个数。	

**MORE** → XGBoost工具库建模应用详解  
<http://www.showmeai.tech/article-detail/204>

搜索 | 微信 ShowMeAI 研究中心

booster [default=gbtree]

有两种模型可以选择 gbtree 和 gblinear。gbtree 使用基于树的模型进行提升计算，gblinear 使用线性模型进行提升计算。缺省值为 gbtree。

silent [default=0]

取 0 时表示打印出运行时信息，取 1 时表示以缄默方式运行，不打印运行时信息。缺省值为 0。

nthread

XGBoost 运行时的线程数。缺省值是当前系统可以获得的最大线程数。

num\_pbuffer

预测缓冲区大小，通常设置为训练实例的数目。缓冲用于保存最后一步提升的预测结果，无需人为设置。

num\_feature

Boosting 过程中用到的特征维数，设置为特征个数。XGBoost 会自动设置，无需人为设置。

(2) 树模型booster参数





eta [default=0.3]

为了防止过拟合，更新过程中用到的收缩步长。在每次提升计算之后，算法会直接获得新特征的权重。

eta 通过缩减特征的权重使提升计算过程更加保守。缺省值为 0.3 取值范围为： $[0, 1]$ 。

gamma [default=0]

树要进一步分裂生长所需的最小 loss 减小值。the larger, the more conservative the algorithm will be. 取值范围为  $[0, \infty]$ 。

max\_depth [default=6]

树的最大深度。缺省值为 6，取值范围为  $[1, \infty]$ 。

min\_child\_weight [default=1]

孩子节点中最小的样本权重和。如果一个叶子节点的样本权重和小于min\_child\_weight则拆分过程结束。

在现行回归模型中，这个参数是指建立每个模型所需要的最小样本数。该成熟越大算法越 conservative 取值范围为  $[0, \infty]$ 。

max\_delta\_step [default=0]

我们允许每个树的权重被估计的值。如果它的值被设置为 0，意味着没有约束；如果它被设置为一个正值，它能够使得更新的步骤更加保守。

通常这个参数是没有必要的，但是如果在逻辑回归中类极其不平衡这时候他有可能会起到帮助作用。把它范围设置为1-10之间也许能控制更新。取值范围为  $[0, \infty]$ 。

subsample [default=1]

用于训练模型的子样本占整个样本集合的比例。如果设置为 0.5 则意味着 XGBoost 将随机的从整个样本集合中随机的抽取 50% 的子样本建立树模型，这能够防止过拟合。取值范围为  $(0, 1]$ 。

colsample\_bytree [default=1]

在建树时对特征采样的比例。缺省值为 1 取值范围为  $(0, 1]$ 。

(3) 线性Booster参数



XGBoost

工具库建模应用详解

模型调参与高级功能

XGBoost参数详解

 <http://www.showmeai.tech/>

线性booster参数

MORE → XGBoost工具库建模应用详解  
<http://www.showmeai.tech/article-detail/204>

lambda

L2正则的惩罚系数

alpha

L1正则的惩罚系数

lambda\_bias

在偏置上的L2正则

dmlc

XGBoost

搜索 | 微信

ShowMeAI 研究中心

lambda [default=0]

L2 正则的惩罚系数。

alpha [default=0]

L1 正则的惩罚系数。

lambda\_bias

在偏置上的 L2 正则。缺省值为 0 (在 L1 上没有偏置项的正则，因为L1时偏置不重要)。

(4) 任务参数



XGBoost

工具库建模应用详解

模型调参与高级功能

XGBoost参数详解

 <http://www.showmeai.tech/>

任务参数

MORE → XGBoost工具库建模应用详解  
<http://www.showmeai.tech/article-detail/204>

base\_score

有实例的初始化预测分数  
全局偏置

objective

定义学习任务  
及相应的学习目标

seed

随机数的种子  
缺省值为0

eval\_metric

校验数据所需要的评价指标  
不同的目标函数将会有缺省的评价指标

dmlc

XGBoost

搜索 | 微信

ShowMeAI 研究中心

objective [ default=reg:linear ]

定义学习任务及相应的学习目标。

可选的目标函数如下：

- reg:linear ：线性回归。
- reg:logistic: 逻辑回归。
- binary:logistic: 二分类的逻辑回归问题，输出为概率。
- binary:logitraw: 二分类的逻辑回归问题，输出的结果为wTx。

<https://mp.weixin.qq.com/s/9cMW0Vcw0fuXUDysR8Yz2A>

10/20

- count:poisson: 计数问题的poisson回归，输出结果为poisson分布。在poisson回归中，max\_delta\_step的缺省值为0.7。(used to safeguard optimization)。
- multi:softmax : 让XGBoost采用softmax目标函数处理多分类问题，同时需要设置参数num\_class(类别个数)。
- multi:softprob: 和softmax一样，但是输出的是ndata \* nclass的向量，可以将该向量reshape成ndata行nclass列的矩阵。没行数据表示样本所属于每个类别的概率。
- rank:pairwise: set XGBoost to do ranking task by minimizing the pairwise loss。

base\_score [ default=0.5 ]

- 所有实例的初始化预测分数，全局偏置；
- 为了足够的迭代次数，改变这个值将不会有太大的影响。

eval\_metric [ default according to objective ]

校验数据所需要的评价指标，不同的目标函数将会有缺省的评价指标(rmse for regression, and error for classification, mean average precision for ranking)

用户可以添加多种评价指标，对于 Python 用户要以 list 传递参数给程序，而不是 map 参数 list 参数不会覆盖 eval\_metric

可供的选择如下：

- rmse: root mean square error
- logloss: negative log-likelihood
- error: Binary classification error rate. It is calculated as  $\frac{\#(\text{wrong cases})}{\#(\text{all cases})}$ . For the predictions, the evaluation will regard the instances with prediction value larger than 0.5 as positive instances, and the others as negative instances.
- merror: Multiclass classification error rate. It is calculated as  $\frac{\#(\text{wrong cases})}{\#(\text{all cases})}$ .
- mlogloss: Multiclass logloss
- auc: Area under the curve for ranking evaluation.
- ndcg: Normalized Discounted Cumulative Gain
- map: Mean average precision
- ndcg@n,map@n: n can be assigned as an integer to cut off the top positions in the lists for evaluation.
- ndcg-,map-,ndcg@n-,map@n-: In XGBoost, NDCG and MAP will evaluate the score of a list without any positive samples as 1. By adding - in the evaluation metric XGBoost will evaluate these score as 0 to be consistent under some conditions. training repeatedly

seed [ default=0 ]


随机数的种子。缺省值为 0。

## 内置调参优化

### (1) 交叉验证

XGBoost自带实验与调参的一些方法，如下为交叉验证方法xgb.cv。

```
xgb.cv(param, dtrain, num_round, nfold=5,metrics={'error'}, seed = 0)
```




XGBoost

工具库建模应用详解

模型调参与高级功能


交叉验证



<http://www.showmeai.tech/>

```
xgb.cv(param, dtrain, num_round, nfold=5, metrics={'error'}, seed = 0)
```

	train-error-mean	train-error-std	test-error-mean	test-error-std
0	0.008329	0.004488	0.009829	0.005859
1	0.003838	0.001788	0.005067	0.001857
2	0.003263	0.001780	0.004606	0.003534
3	0.001881	0.001397	0.002456	0.003123
4	0.001574	0.000809	0.001842	0.001977
5	0.001305	0.000372	0.001382	0.001228
6	0.001228	0.000260	0.001228	0.001041
7	0.001228	0.000260	0.001228	0.001041
8	0.001228	0.000260	0.001228	0.001041
9	0.001228	0.000260	0.001228	0.001041

 搜索 | 微信 ShowMeAI 研究中心

(2) 添加预处理

我们可以把数据建模过程中的一些设置加到交叉验证环节里，比如对于不同类别的样本加权，可以参考下列代码示例：

```
# 计算正负样本比，调整样本权重
def fpreproc(dtrain, dtest, param):
    label = dtrain.get_label()
    ratio = float(np.sum(label == 0)) / np.sum(label==1)

    param['scale_pos_weight'] = ratio
    return (dtrain, dtest, param)

# 先做预处理，计算样本权重，再做交叉验证
xgb.cv(param, dtrain, num_round, nfold=5,
        metrics={'auc'}, seed = 0, fpreproc = fpreproc)
```



XGBoost

工具库建模应用详解

模型调参与高级功能

添加预处理的交叉验证




<http://www.showmeai.tech/>

```
# 计算正负样本比，调整样本权重
def fpreproc(dtrain, dtest, param):
    label = dtrain.get_label()
    ratio = float(np.sum(label == 0)) / np.sum(label==1)
    param['scale_pos_weight'] = ratio
    return (dtrain, dtest, param)

# 先做预处理，计算样本权重，再做交叉验证
xgb.cv(param, dtrain, num_round, nfold=5,
        metrics={'auc'}, seed = 0, fpreproc = fpreproc)
```

 搜索 | 微信 ShowMeAI 研究中心



XGBoost

工具库建模应用详解

模型调参与高级功能

添加预处理的交叉验证



<http://www.showmeai.tech/>

	train-auc-mean	train-auc-std	test-auc-mean	test-auc-std
0	0.995562	0.001265	0.994036	0.001449
1	0.999546	0.000749	0.999860	0.000121
2	0.999764	0.000372	0.999900	0.000113
3	0.999774	0.000378	0.999930	0.000116
4	0.999969	0.000037	0.999903	0.000160
5	0.999987	0.000012	0.999955	0.000058
6	0.999991	0.000006	0.999978	0.000020
7	0.999994	0.000006	0.999977	0.000021
8	0.999996	0.000004	0.999980	0.000022
9	0.999997	0.000004	0.999987	0.000019

 搜索 | 微信 ShowMeAI 研究中心

(3) 自定义损失函数与评估准则

XGBoost支持在训练过程中，自定义损失函数和评估准则，其中损失函数的定义需要返回损失函数一阶和二阶导数的计算方法，评估准则部分需要对数据的label和预估值进行计算。其中损失函数用于训练过程中的树结构学习，而评估准则很多时候是用在验证集上进行效果评估。

```
print('使用自定义损失函数进行交叉验证')
# 自定义损失函数，需要提供损失函数的一阶导和二阶导
def logregobj(preds, dtrain):
    labels = dtrain.get_label()
    preds = 1.0 / (1.0 + np.exp(-preds))
    grad = preds - labels
    hess = preds * (1.0-preds)
    return grad, hess

# 自定义评估准则，评估预测值和标准答案之间的差距
def evalerror(preds, dtrain):
    labels = dtrain.get_label()
    return 'error', float(sum(labels != (preds > 0.0))) / len(labels)

watchlist = [(dtest,'eval'), (dtrain,'train')]
param = {'max_depth':3, 'eta':0.1, 'silent':1}
num_round = 5

# 自定义损失函数训练
bst = xgb.train(param, dtrain, num_round, watchlist, logregobj, evalerror)

# 交叉验证
xgb.cv(param, dtrain, num_round, nfold = 5, seed = 0, obj = logregobj, feval=evalerror)
```




**XGBoost**  
工具库建模应用详解  
**模型调参与高级功能**  
自定义损失函数与评估准则  
<http://www.showmeai.tech/>

```
print('使用自定义损失函数进行交叉验证')
# 自定义损失函数，需要提供损失函数的一阶导和二阶导
def logregobj(preds, dtrain):
    labels = dtrain.get_label()
    preds = 1.0 / (1.0 + np.exp(-preds))
    grad = preds - labels
    hess = preds * (1.0-preds)
    return grad, hess

# 自定义评估准则，评估预测值和标准答案之间的差距
def evalerror(preds, dtrain):
    labels = dtrain.get_label()
    return 'error', float(sum(labels != (preds > 0.0))) / len(labels)

watchlist = [(dtest,'eval'), (dtrain,'train')]
param = {'max_depth':3, 'eta':0.1, 'silent':1}
num_round = 5
```

搜索 | 微信 ShowMeAI 研究中心



**XGBoost**  
工具库建模应用详解  
**模型调参与高级功能**  
自定义损失函数与评估准则  
<http://www.showmeai.tech/>

```
# 自定义损失函数训练
bst = xgb.train(param, dtrain, num_round, watchlist, logregobj, evalerror)

# 交叉验证
xgb.cv(param, dtrain, num_round, nfold = 5, seed = 0, obj = logregobj, feval=evalerror)
```

	train-error-mean	train-error-std	train-rmse-mean	train-rmse-std	test-error-mean	test-error-std	test-rmse-mean	test-rmse-std
0	0.517887	0.001085	0.308875	0.005171	0.517886	0.004343	0.309039	0.005206
1	0.517887	0.001085	0.176503	0.002046	0.517886	0.004343	0.177803	0.003767
2	0.014433	0.000223	0.172680	0.003719	0.014433	0.000892	0.174890	0.009391
3	0.014433	0.000223	0.275762	0.001776	0.014433	0.000892	0.276689	0.005918
4	0.014433	0.000223	0.399885	0.003371	0.014433	0.000892		

搜索 | 微信 ShowMeAI 研究中心

```
使用自定义损失函数进行交叉验证

[0] eval-rmse:0.306901 train-rmse:0.306164 eval-error:0.518312 train-error:0.517887
[1] eval-rmse:0.179189 train-rmse:0.177278 eval-error:0.518312 train-error:0.517887
[2] eval-rmse:0.172565 train-rmse:0.171728 eval-error:0.016139 train-error:0.014433
[3] eval-rmse:0.269612 train-rmse:0.27111 eval-error:0.016139 train-error:0.014433
[4] eval-rmse:0.396903 train-rmse:0.398256 eval-error:0.016139 train-error:0.014433
```

(4) 只用前n颗树预测

对于 boosting 模型来说，最后会训练得到很多基学习器(在XGBoost中很多时候是很多棵树)，我们可以一次完整训练，只用前n棵树的集成来完成预测。

```
#!/usr/bin/python
import numpy as np
import pandas as pd
import pickle
import xgboost as xgb
from sklearn.model_selection import train_test_split

# 基本例子，从csv文件中读取数据，做二分类

# 用pandas读入数据
data = pd.read_csv('./data/Pima-Indians-Diabetes.csv')

# 做数据切分
train, test = train_test_split(data)

# 转换成Dmatrix格式
feature_columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigree', 'Age']
target_column = 'Outcome'
xgtrain = xgb.DMatrix(train[feature_columns].values, train[target_column].values)
xgtest = xgb.DMatrix(test[feature_columns].values, test[target_column].values)

# 参数设定
param = {'max_depth':5, 'eta':0.1, 'silent':1, 'subsample':0.7, 'colsample_bytree':0.7, 'objective':'binary:logistic'}

# 设定watchlist用于查看模型状态
watchlist = [(xgtest,'eval'), (xgtrain,'train')]
num_round = 10
bst = xgb.train(param, xgtrain, num_round, watchlist)

# 只用第1颗树预测
ypred1 = bst.predict(xgtest, ntree_limit=1)
# 用前9颗树预测
ypred2 = bst.predict(xgtest, ntree_limit=9)
label = xgtest.get_label()
print('用前1颗树预测的错误率为 %f' % (np.sum((ypred1>0.5)!=label) /float(len(label))))
print('用前9颗树预测的错误率为 %f' % (np.sum((ypred2>0.5)!=label) /float(len(label))))
```

```
[0] eval-error:0.255208 train-error:0.196181
[1] eval-error:0.234375 train-error:0.175347
[2] eval-error:0.25 train-error:0.163194
[3] eval-error:0.229167 train-error:0.149306
[4] eval-error:0.213542 train-error:0.154514
[5] eval-error:0.21875 train-error:0.152778
[6] eval-error:0.21875 train-error:0.154514
[7] eval-error:0.213542 train-error:0.138889
[8] eval-error:0.1875 train-error:0.147569
[9] eval-error:0.1875 train-error:0.144097
```

用前1颗树预测的错误率为 0.255208  
用前9颗树预测的错误率为 0.187500

## 预估器调参优化

### (1) SKLearn形态接口实验评估

XGBoost 有 SKLearn 预估器形态的接口，整体使用方法和 SKLearn 中其他预估器一致，如下是手动对数据做交叉验证，注意到这里直接使用XGBClassifier对 Dataframe 数据进行 fit 拟合和评估。

```
import pickle
import xgboost as xgb

import numpy as np
from sklearn.model_selection import KFold, train_test_split, GridSearchCV
from sklearn.metrics import confusion_matrix, mean_squared_error
from sklearn.datasets import load_iris, load_digits, load_boston

rng = np.random.RandomState(31337)

# 二分类：混淆矩阵
print("数字0和1的二分类问题")
digits = load_digits(2)
y = digits['target']
X = digits['data']
# 数据切分对象
kf = KFold(n_splits=2, shuffle=True, random_state=rng)
print("在2折数据上的交叉验证")
# 2折交叉验证
for train_index, test_index in kf.split(X):
    xgb_model = xgb.XGBClassifier().fit(X[train_index], y[train_index])
    predictions = xgb_model.predict(X[test_index])
    actuals = y[test_index]
    print("混淆矩阵:")
    print(confusion_matrix(actuals, predictions))

# 多分类：混淆矩阵
print("\nIris: 多分类")
iris = load_iris()
y = iris['target']
X = iris['data']
kf = KFold(n_splits=2, shuffle=True, random_state=rng)
print("在2折数据上的交叉验证")
for train_index, test_index in kf.split(X):
    xgb_model = xgb.XGBClassifier().fit(X[train_index], y[train_index])
    predictions = xgb_model.predict(X[test_index])
    actuals = y[test_index]
    print("混淆矩阵:")
    print(confusion_matrix(actuals, predictions))

# 回归问题：MSE
print("\n波士顿房价回归预测问题")
boston = load_boston()
y = boston['target']
X = boston['data']
kf = KFold(n_splits=2, shuffle=True, random_state=rng)
print("在2折数据上的交叉验证")
for train_index, test_index in kf.split(X):
    xgb_model = xgb.XGBRegressor().fit(X[train_index], y[train_index])
```

```

predictions = xgb_model.predict(X[test_index])
actuals = y[test_index]
print("MSE:", mean_squared_error(actuals, predictions))

```

数字0和1的二分类问题

在2折数据上的交叉验证

混淆矩阵:

```
[[87  0]
 [ 1 92]]
```

混淆矩阵:

```
[[91  0]
 [ 3 86]]
```

Iris: 多分类

在2折数据上的交叉验证

混淆矩阵:

```
[[19  0  0]
 [ 0 31  3]
 [ 0  1 21]]
```

混淆矩阵:

```
[[31  0  0]
 [ 0 16  0]
 [ 0  3 25]]
```

波士顿房价回归预测问题

在2折数据上的交叉验证

MSE: 9.860776812557337

MSE: 15.942418468446029

## (2) 网格搜索调参

上面提到 XGBoost 的预估器接口，整体使用方法和 SKLearn 中其他预估器一致，所以我们可以使用 SKLearn 中的超参数调优方法来进行模型调优。

如下是一个典型的网格搜索交叉验证调优超参数的代码示例，我们会给出候选参数列表字典，通过 GridSearchCV 进行交叉验证实验评估，选出 XGBoost 在候选参数中最优的超参数。

```

print("参数最优化: ")
y = boston['target']
X = boston['data']
xgb_model = xgb.XGBRegressor()
clf = GridSearchCV(xgb_model,
                   {'max_depth': [2,4,6],
                    'n_estimators': [50,100,200]}, verbose=1)

clf.fit(X,y)
print(clf.best_score_)
print(clf.best_params_)

```

参数最优化:

Fitting 3 folds for each of 9 candidates, totalling 27 fits



```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.  
  
0.6001029721598573  
{'max_depth': 4, 'n_estimators': 100}  
  
[Parallel(n_jobs=1)]: Done 27 out of 27 | elapsed: 1.3s finished
```

### (3) early-stopping早停

XGBoost 模型有时候会因为不停叠加新的树(修正训练集上拟合尚不正确的一些样本)，可能会因为对于训练集过度学习而导致模型过拟合。

early stopping 早停止是一个有效的策略，具体的做法是，在训练集不断追加树学习的过程中，对验证集上的表现进行监控，如果出现一定轮次评估准则都没有优化提升的情况，则回溯到历史上验证集最好的点，保存为最佳模型。

下面是对应的代码示例，其中参数early\_stopping\_rounds设定了验证集上能接受的效果不提升的最多轮次数，eval\_set指定了验证数据集。

```
# 在训练集上学习模型，一颗一颗树添加，在验证集上看效果，当验证集效果不再提升，停止树的添加与生长  
X = digits['data']  
y = digits['target']  
  
X_train, X_val, y_train, y_val = train_test_split(X, y, random_state=0)  
clf = xgb.XGBClassifier()  
clf.fit(X_train, y_train, early_stopping_rounds=10, eval_metric="auc",  
        eval_set=[(X_val, y_val)])
```

```
[0] validation_0-auc:0.999497  
Will train until validation_0-auc hasn't improved in 10 rounds.  
[1] validation_0-auc:0.999497  
[2] validation_0-auc:0.999497  
[3] validation_0-auc:0.999749  
[4] validation_0-auc:0.999749  
[5] validation_0-auc:0.999749  
[6] validation_0-auc:0.999749  
[7] validation_0-auc:0.999749  
[8] validation_0-auc:0.999749  
[9] validation_0-auc:0.999749  
[10] validation_0-auc:1  
[11] validation_0-auc:1  
[12] validation_0-auc:1  
[13] validation_0-auc:1  
[14] validation_0-auc:1  
[15] validation_0-auc:1  
[16] validation_0-auc:1
```

```
[17] validation_0-auc:1
[18] validation_0-auc:1
[19] validation_0-auc:1
[20] validation_0-auc:1
Stopping. Best iteration:
[10] validation_0-auc:1

XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
              max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
              n_jobs=1, nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=True, subsample=1)
```

#### (4) 特征重要度

XGBoost 建模过程中，还可以学习到对应的特征重要度信息，并保存在模型的 `feature_importances_` 属性中。如下为绘制特征重要度的可视化代码：

```
iris = load_iris()
y = iris['target']
X = iris['data']
xgb_model = xgb.XGBClassifier().fit(X,y)

print('特征排序: ')
feature_names=['sepal_length', 'sepal_width', 'petal_length', 'petal_width']
feature_importances = xgb_model.feature_importances_
indices = np.argsort(feature_importances)[::-1]

for index in indices:
    print("特征 %s 重要度为 %f" %(feature_names[index], feature_importances[index]))

%matplotlib inline
import matplotlib.pyplot as plt
plt.figure(figsize=(16,8))
plt.title("feature importances")
plt.bar(range(len(feature_importances)), feature_importances[indices], color='b')
plt.xticks(range(len(feature_importances)), np.array(feature_names)[indices], color='b')
```

```
特征排序:
特征 petal_length 重要度为 0.415567
特征 petal_width 重要度为 0.291557
特征 sepal_length 重要度为 0.179420
特征 sepal_width 重要度为 0.113456
```

## (5) 并行训练加速

在多资源的情况下，XGBoost可以实现并行训练加速，示例代码如下：

```
import os

if __name__ == "__main__":
    try:
        from multiprocessing import set_start_method
    except ImportError:
        raise ImportError("Unable to import multiprocessing.set_start_method."
                           " This example only runs on Python 3.4")
    #set_start_method("forkserver")

    import numpy as np
    from sklearn.model_selection import GridSearchCV
    from sklearn.datasets import load_boston
    import xgboost as xgb

    rng = np.random.RandomState(31337)

    print("Parallel Parameter optimization")
    boston = load_boston()

    os.environ["OMP_NUM_THREADS"] = "2" # or to whatever you want
    y = boston['target']
    X = boston['data']
    xgb_model = xgb.XGBRegressor()
    clf = GridSearchCV(xgb_model, {'max_depth': [2, 4, 6],
                                   'n_estimators': [50, 100, 200]}, verbose=1,
                       n_jobs=2)

    clf.fit(X, y)
    print(clf.best_score_)
    print(clf.best_params_)
```

```
Parallel Parameter optimization
Fitting 3 folds for each of 9 candidates, totalling 27 fits

[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done 24 out of 27 | elapsed: 2.2s remaining: 0.3s

0.6001029721598573
{'max_depth': 4, 'n_estimators': 100}

[Parallel(n_jobs=2)]: Done 27 out of 27 | elapsed: 2.4s finished
```

### 参考资料

- [1]图解机器学习|XGBoost模型详解: <https://www.showmeai.tech/article-detail/194>
- [2]图解python | 安装与环境设置: <https://www.showmeai.tech/article-detail/65>

[3]Python数据分析|Pandas核心操作函数大全: <https://www.showmeai.tech/article-detail/146>

[4]数据分析系列教程: <https://www.showmeai.tech/tutorials/33>

[5]数据科学工具速查 | Pandas使用指南: <https://www.showmeai.tech/article-detail/101>

编辑：王菁

校对：邱婷婷

[阅读原文](#)