XGBoost参数调优小结

原创 石头 机器学习算法那些事 2019-01-15

XGBoost在Kaggle比赛大放异彩，在之前的文章已介绍XGBoost算法原理和XGBoost切分算法，网上对XGBoost参数的解释大部分只停留在表面，对刚入门机器学习算法的人极其不友好，本文在解释某些重要参数的同时会参考数学公式以增加对XGBoost算法原理的理解，并通过分类实例阐述XGBoost调参思想 。

> 本文的框架和实例来自于https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/的翻译，并根据自己的理解对内容和代码进行了修改，本文实例只处理训练数据，并以五折交叉验证率作为衡量模型性能的标准 。
> 代码链接：https://github.com/zhangleiszu/xgboost-

目录

1. XGBoost算法原理简单回顾

XGBoost算法的每颗树都是对前一个模型损失函数的二阶导展开式拟合，然后结合多棵树给出分类或回归结果。因此我们只要知道每棵树的构建过程，就能很好的理解XGBoost的算法原理。

假设对于前t-1棵树的预测结果为 $\hat{y}^{(t-1)}$ ，真实结果为y，用L表示损失函数，那么当前模型的损失函数为：

$$\sum_{i=1}^{n} L(y_i, \hat{y}_i^{(t-1)}) \quad (1.1)$$

XGBoost法构建树的同时考虑了正则化，定义每棵树的复杂度为：

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \quad (1.2)$$

因此，包含正则化项的损失函数为：

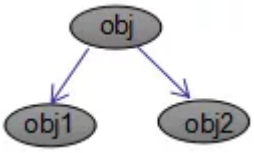
$$obj(\theta) = \sum_{i=1}^{n} L(y_i, \hat{y}_i^{(t-1)}) + \sum_{k=1}^{t} \Omega(f_k) \quad (1.3)$$

最小化（1.3）式得到最终的目标函数：

$$obj(\theta^*) = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j+\lambda} + \gamma T \qquad (1.4)$$

目标函数也称为打分函数，它是衡量树结构好坏的标准，值越小代表树的结构越好。因此树的节点切分规则是选择树目标函数值下降最大的点作为切分点 。

如下图，对于某一节点切分前后的目标函数之差记为Gain。



$$Gain = obj - obj1 - obj2$$

得：

$$Gain = \frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right] - \gamma \qquad (1.5)$$

选择Gain下降最大的切分点对树分裂，以此类推，得到完整的树 。

因此，只要知道（1.5）式的参数值就能基本确定树模型，其中G_L表示左子节点损失函数的二阶导之和，G_R表示右子节点损失函数的二阶导之和，H_L表示左子节点损失函数的一阶导之和，H_R表示右子节点损失函数的一阶导之和 。λ表示正则化系数，γ表示切分节点难度，λ和γ定义了树的复杂度。再简单一点，我们只要设置了损失函数L，正则化系数λ和切分节点难度γ，就基本确定了树模型 。XGBoost参数择要重点考虑这三个参数 。

2. XGBoost的优点

1. 正则化

XGBoost切分节点时考虑了正则化项（如图1.3），减少了过拟合。实际上，XGBoost也称为“正则化提升”技术。

2. 并行处理

虽然XGBoost是串行迭代生成各决策树，但是我们在切分节点时可以做到并行处理 ，因此XGBoost支持Hadoop实现 。

3. 高度灵活性

XGBoost支持自定义目标函数和评价函数，评价函数是衡量模型好坏的标准，目标函数是损失函数，如上节讨论，只要知道了损失函数，然后求其一阶导和二阶导，就能确定节点的切分规则。

4. 缺失值处理

XGBoost内置了处理缺失值的节点切分规则 。用户需要提供一个和其他样本不一样的值（如-999）， 然后把该值作为参数的缺失值 。

5. 内置交叉验证

XGBoost允许每一轮迭代中使用交叉验证，因此，可以很方便的获得交叉验证率，得到最优boosting迭代次数 ， 并不需要传统的网格搜索法来获取最优迭代次数 。

6. 在已有的模型基础上继续

XGBoost可以从上一轮的结果上继续训练，从而节省了运行时间，通过设置模型参数“process_type”=update来实现 。

3. XGBoost参数解释

XGBoost参数把所有的参数分为三类：

1. 通用参数：控制了模型的宏观功能 。
2. Booster参数：控制每一轮迭代的树生成。
3. 学习目标参数：决定了学习场景，如损失函数和评价函数 。

3.1 通用参数

- booster [default = gbtree]

供选择的模型有gbtree和gblinear，gbtree使用基于树的模型进行提升，gblinear使用线性模型进行提升，默认是gbtree 。这里只介绍tree booster，因为它的表现远远胜过linear booster

- silent [default = 0]

  取0时表示打印运行信息，取1时表示不打印运行信息。

- nthread

  XGBoost运行时的线程数，默认是当前系统可运行的最大线程数。

- num_pbuffer

  预测数据的缓冲区大小，一般设置成训练样本个数的大小 。缓冲区保留最后一次迭代后的预测结果，系统自动设置。

- num_feature

  设置特征维数来构建树模型，系统自动设置 。

3.2 booster参数

- eta [default = 0.3]

控制树模型的权重，与AdaBoost算法的学习率（learning rate）含义类似，缩小权重以避免模型处于过拟合，在对eta进行参数择优时，需要与迭代次数（num_boosting_rounding）结合在一起考虑 ，如减小学习率，那么增加最大迭代次数；反之，则减小最大迭代次数 。

范围：[0,1]

模型迭代公式：

$$f^t(x) = f^{t-1}(x) + \alpha * g^t(x)$$

其中$f^t(x)$表示前t棵树结合模型的输出结果，$g^t(x)$表示第t棵树模型，α表示学习率，控制模型更新的权重 。

- gamma [default = 0]

控制模型切分节点的最小损失函数值，对应（1.5）式的γ，若γ设置的过大（1.5）式小于零，则不进行节点切分，因此降低了模型的复杂度。

- lambda [default = 1]

表示L2正则化参数，对应（1.5）式中的λ，增大λ的值使模型更保守，避免过拟合 。

- alpha [default = 0]

表示L1正则化参数，L1正则化的意义在于可以降维，增大alpha值使模型更保守，避免过拟合

范围：[0,∞]

- max_depth [default=6]

表示树的最大深度，增加树的深度提高了树的复杂度，很有可能会导致过拟合，0表示对树的最大深度无限制 。

范围：[0,∞]

- min_child_weight [default=1]

表示最小叶子节点的样本权重和，和之前介绍的min_child_leaf参数有差别，min_child_weight表示样本权重的和，min_child_leaf表示样本总数和。

范围：[0,∞]

- subsample [default=1]

  表示随机采样一定比例的样本来构建每棵树，减小比例参数subsample值，算法会更加保守，避免过拟合

  范围：[0,1]

- colsample_bytree,colsample_bylevel,colsample_bynode

  这个三个参数表示对特征进行随机抽样，且具有累积效应 。

  colsample_bytree，表示每棵树划分的特征比重

  colsample_bylevel，表示树的每层划分的特征比重

  colsample_bynode，表示树的每个节点划分的特征比重 。

  比如，样本共有64个特征，设置{'colsample_bytree' : 0.5 , 'colsample_bylevel' : 0.5 , 'colsample_bynode' : 0.5 }，那么随机采样4个特征应用于树的每一个节点切分 。

  范围：[0,1]

- tree_method string [default = auto]

  表示树的构建方法，确切来说是切分点的选择算法，包括贪心算法，近似贪心算法，直方图算法

  exact：贪心算法

  aprrox：近似贪心算法，选择特征的分位点进行切分

  hist：直方图切分算法，LightGBM算法也采用该切分算法 。

- scale_pos_weight [default = 1]

  当正负样本不平衡时，设置该参数为正值 ，可以使算法更快收敛 。

  典型的值可设置为：（负样本个数）/（正样本个数）

- process_type [default = default]

  default：正常的构建提升树过程

  update: 从已有的模型构建提升树

3.3 学习任务参数

根据任务和目的设置参数

objective，训练目标，分类还是回归

reg : linear，线性回归

reg : logistic，逻辑回归

binary : logistic，使用LR二分类，输出概率

binary : logitraw，使用LR二分类，输出Logistic转换之前的分类得分 。

eval_metric，评价验证集的指标，默认根据目标函数设置，默认情况下，最小均方差用于回归，错误率用于分类，平均精确率用于排序 。

rmse：均方差

mae：平均绝对值误差

error ： 错误率

logloss：负的对数损失

auc ： roc曲线下的面积

seed [default=0]

随机数种子，设置它可以复现随机数据结果 。

4. 参数调优实例

原始数据集和经过特征工程处理后的数据集在开头的链接部分进行下载，算法在jupter - notebook交互界面开发 。

定义模型评价函数，并根据一定的学习率得到最优迭代次数 ，函数定义如下：

```
# 构建xgb模型并执行交叉验证，该模型可以验证更新后的模型好坏
# 定义modelfit函数，根据一定的学习率得到最优迭代次数
def modelfit(alg, dtrain, predictors, useTrainCV=True, cv_folds=5, early_stopping_rounds=50):
    if useTrainCV:
        xgb_param = alg.get_xgb_params()
        xgtrain = xgb.DMatrix(dtrain[predictors].values, label=dtrain[target].values)
        cvresult = xgb.cv(xgb_param, xgtrain, num_boost_round=alg.get_params()['n_estimators'], nfold=cv_folds,
                          metrics='auc', early_stopping_rounds=early_stopping_rounds, verbose_eval=True)
        alg.set_params(n_estimators=cvresult.shape[0])

    # Fit the algorithm on the data
    alg.fit(dtrain[predictors].values, dtrain[target].values, eval_metric='auc')

    # Predict training set:
    dtrain_predictions = alg.predict(dtrain[predictors].values)
    dtrain_predprob = alg.predict_proba(dtrain[predictors].values)[:, 1]

    # Print model report:
    print("\nModel Report")
    print("Accuracy : %.4g" % metrics.accuracy_score(dtrain[target].values, dtrain_predictions))
    print("AUC Score (Train): %f" % metrics.roc_auc_score(dtrain[target].values, dtrain_predprob))

    feat_imp = pd.Series(alg.get_booster().get_fscore()).sort_values(ascending=False)  #降序查看特征重要性程度
```

Step1:

根据经验设置默认的参数，调用modelfit函数得到最优迭代次数是198 。

```
[196]   train-auc:0.921243+0.00179222   test-auc:0.836213+0.0116077
[197]   train-auc:0.921403+0.00189843   test-auc:0.836146+0.0116368
[198]   train-auc:0.921868+0.00170905   test-auc:0.835853+0.0115561
```

Step2: 树最大深度和最小叶子节点权重调优

根据第一步得到的最优迭代次数更新模型参数n_estimators，然后调用model_selection类的GridSearchCV设置交叉验证模型，最后调用XGBClassifier类的fit函数得到最优树的最大深度和最小叶子节点权重。

a) 设置树最大深度和最小叶子节点权重范围：

```
param_test1 = {
    'max_depth':range(3,10,2),
    'min_child_weight':range(1,6,2)
}
```

b) GridSearchCV设置交叉验证模型：

```
gsearch1 = GridSearchCV(estimator = XGBClassifier( learning_rate =0.1, n_estimators=198, max_depth=5,
 min_child_weight=1, gamma=0, subsample=0.8, colsample_bytree=0.8,
 objective= 'binary:logistic', nthread=4, scale_pos_weight=1, seed=27),
 param_grid = param_test1, scoring='roc_auc',n_jobs=4,iid=False, cv=5)
```

c) XGBClassfier类的fit函数更新参数:

```
gsearch1.fit(train[predictors].values,train[target].values)
gsearch1.best_params_, gsearch1.best_score_
```

Step3: gamma参数调优

Step4: subsample and colsample_bytree参数调优

Step5: 正则化参数调优

> **Step3，Step4和Step5参数调优思想Step2一致，这里不再展开，可参考代码去理解。**

Step6: 减小学习率，增大对应的最大迭代次数。参数调优思想与Step1一致，得到最优的迭代次数，并输出交叉验证率结果。

```
# 第六步：学习率参数调优,减小学习率，最优迭代次数也会相应的改变，增大最大迭代数 。由第四步得到最优的reg_alpha是1
xgb4 = XGBClassifier(
 learning_rate =0.01,
 n_estimators=5000,
 max_depth=3,
 min_child_weight=5,
 gamma=0,
 subsample=0.8,
 colsample_bytree=0.9,
 reg_alpha=1,
 objective= 'binary:logistic',
 nthread=4,
 scale_pos_weight=1,
 seed=27)
modelfit(xgb4, train, predictors)
```

迭代次数为2346时，得到最优分类结果(如下图)。

```
[2343]  train-auc:0.873217+0.00254332   test-auc:0.838402+0.0133809
[2344]  train-auc:0.873227+0.00254477   test-auc:0.838408+0.0133855
[2345]  train-auc:0.873239+0.00254333   test-auc:0.838412+0.0133698
[2346]  train-auc:0.873257+0.00254121   test-auc:0.838416+0.0133724
```

5. XGBoost参数调优小结

XGBoost是一个高性能的学习模型算法，当你不知道怎么对模型进行参数调优时，可参考上一节的步骤去进行参数调优 。本节根据上节的内容，以及自己的项目经验阐述一下本人对XGBoost算法参数调优的一些理解，若有错误之处，还忘指正 。

（1）模型初始化 。在初始化模型参数时，我们尽量让模型的复杂度较高，然后一步一步的通过参数调优来降低模型复杂度 。如上节初始化参数：叶子节点最小权重为0，最大树深度为5，最小损失函数下降值为0，这些参数初始化都是复杂化模型 。

(2) 学习率和最大迭代次数 。这两个参数的调优一定是联系在一起的，学习率越大，达到相同性能的模型所需要的最大迭代次数越小；学习率越小，达到相同性能的模型所需要的最大迭代次数越大 。XGBoost每个参数的更新都需要进行多次迭代，因此，学习率和最大迭代次数是首先需要考虑的参数 ，且学习率和最大迭代参数的重点不是提高模型的分类准确率，而是提高模型的泛化能力，因此，当模型的分类准确率很高时，我们最后一步应减小学习率的大小，以提高模型的泛化能力 。

(3) 逐步降低模型复杂度 。树的最大深度，叶子节点最小权重等参数都是影响模型复杂度的因素，这个看自己经验，第四节的调参顺序是：树的最大深度和叶子节点最小权重->最小损失函数下降值->行采样和列采样->正则化参数 。实际项目中，我一般按照这个顺序去调参 ，若有不同的理解，欢迎交流。

(4) 特征工程：若模型的准确率在很低的范围，那么我建议先不用去调参了，去重点关注特征和训练数据。特征和数据决定模型上限，模型只是逼近这个上限 。

参考：

https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/

https://xgboost.readthedocs.io/en/latest/

推荐阅读

XGBoost算法原理小结

XGBoost切分点算法