GDBT-sklearn实验

利用sklearn库中的gdbt进行对训练集分类，然后进行调参。

GBDT类库boosting框架参数

首先，我们来看boosting框架相关的重要参数。由于GradientBoostingClassifier和GradientBoostingRegressor的参数绝大部分相同，我们下面会一起来讲，不同点会单独指出。

1) **n_estimators**: 也就是弱学习器的最大迭代次数，或者说最大的弱学习器的个数。一般来说n_estimators太小，容易欠拟合，n_estimators太大，又容易过拟合，一般选择一个适中的数值。默认是100。在实际调参的过程中，我们常常将n_estimators和下面介绍的参数learning_rate一起考虑。

2) **learning_rate**: 即每个弱学习器的权重缩减系数v，也称作步长，在原理篇的正则化章节我们也讲到了，加上了正则化项，我们的强学习器的迭代公式为$fk(x) = f_{k-1}(x) + \nu h_k(x)$。v的取值范围为0<v≤10<v≤1。对于同样的训练集拟合效果，较小的vv意味着我们需要更多的弱学习器的迭代次数。通常我们用步长和迭代最大次数一起来决定算法的拟合效果。所以这两个参数n_estimators和learning_rate要一起调参。一般来说，可以从一个小一点的vv开始调参，默认是1。

3) **subsample**: 即我们在原理篇的正则化章节讲到的子采样，取值为(0,1]。注意这里的子采样和随机森林不一样，随机森林使用的是放回抽样，而这里是不放回抽样。如果取值为1，则全部样本都使用，等于没有使用子采样。如果取值小于1，则只有一部分样本会去做GBDT的决策树拟合。选择小于1的比例可以减少方差，即防止过拟合，但是会增加样本拟合的偏差，因此取值不能太低。推荐在[0.5, 0.8]之间，默认是1.0，即不使用子采样。

4) **init**: 即我们的初始化的时候的弱学习器，拟合对应原理篇里面的f0(x)f0(x)，如果不输入，则用训练集样本来做样本集的初始化分类回归预测。否则用init参数提供的学习器做初始化分类回归预测。一般用在我们对数据有先验知识，或者之前做过一些拟合的时候，如果没有的话就不用管这个参数了。

5) **loss:** 即我们GBDT算法中的损失函数。分类模型和回归模型的损失函数是不一样的。

对于分类模型，有对数似然损失函数"deviance"和指数损失函数"exponential"两者输入选择。默认是对数似然损失函数"deviance"。在原理篇中对这些分类损失函数有详细的介绍。一般来说，推荐使用默认的"deviance"。它对二元分离和多元分类各自都有比较好的优化。而指数损失函数等于把我们带到了Adaboost算法。

对于回归模型，有均方差"ls", 绝对损失"lad", Huber损失"huber"和分位数损失“quantile”。默认是均方差"ls"。一般来说，如果数据的噪音点不多，用默认的均方差"ls"比较好。如果是噪音点较多，则推荐用抗噪音的损失函数"huber"。而如果我们需要对训练集进行分段预测的时候，则采用“quantile”。

6) **alpha**：这个参数只有GradientBoostingRegressor有，当我们使用Huber损失"huber"和分位数损失“quantile”时，需要指定分位数的值。默认是0.9，如果噪音点较多，可以适当降低这个分位数的值。

GBDT类库弱学习器参数

这里我们再对GBDT的类库弱学习器的重要参数做一个总结。由于GBDT使用了CART回归决策树，因此它的参数基本来源于决策树类，也就是说，和DecisionTreeClassifier和DecisionTreeRegressor的参数基本类似。如果你已经很熟悉决策树算法的调参，那么这一节基本可以跳过。不熟悉的朋友可以继续看下去。

1) 划分时考虑的最大特征数max_features: 可以使用很多种类型的值，默认是"None",意味着划分时考虑所有的特征数；如果是"log2"意味着划分时最多考虑log2N个特征；如果是"sqrt"或者"auto"意味着划分时最多考虑√N个特征。如果是整数，代表考虑的特征绝对数。如果是浮点数，代表考虑特征百分比，即考虑（百分比xN）取整后的特征数。其中N为样本总特征数。一般来说，如果样本特征数不多，比如小

于50，我们用默认的"None"就可以了，如果特征数非常多，我们可以灵活使用刚才描述的其他取值来控制划分时考虑的最大特征数，以控制决策树的生成时间。

2) 决策树最大深度**max_depth**: 默认可以不输入，如果不输入的话，默认值是3。一般来说，数据少或者特征少的时候可以不管这个值。如果模型样本量多，特征也多的情况下，推荐限制这个最大深度，具体的取值取决于数据的分布。常用的可以取值10-100之间。

3) 内部节点再划分所需最小样本数**min_samples_split**: 这个值限制了子树继续划分的条件，如果某节点的样本数少于min_samples_split，则不会继续再尝试选择最优特征来进行划分。 默认是2.如果样本量不大，不需要管这个值。如果样本量数量级非常大，则推荐增大这个值。

4) 叶子节点最少样本数**min_samples_leaf**: 这个值限制了叶子节点最少的样本数，如果某叶子节点数目小于样本数，则会和兄弟节点一起被剪枝。 默认是1,可以输入最少的样本数的整数，或者最少样本数占样本总数的百分比。如果样本量不大，不需要管这个值。如果样本量数量级非常大，则推荐增大这个值。

5）叶子节点最小的样本权重和**min_weight_fraction_leaf**：这个值限制了叶子节点所有样本权重和的最小值，如果小于这个值，则会和兄弟节点一起被剪枝。 默认是0，就是不考虑权重问题。一般来说，如果我们有较多样本有缺失值，或者分类树样本的分布类别偏差很大，就会引入样本权重，这时我们就要注意这个值了。

6) 最大叶子节点数**max_leaf_nodes**: 通过限制最大叶子节点数，可以防止过拟合，默认是"None”，即不限制最大的叶子节点数。如果加了限制，算法会建立在最大叶子节点数内最优的决策树。如果特征不多，可以不考虑这个值，但是如果特征分成多的话，可以加以限制，具体的值可以通过交叉验证得到。

7) 节点划分最小不纯度**min_impurity_split:** 这个值限制了决策树的增长，如果某节点的不纯度(基于基尼系数，均方差)小于这个阈值，则该节点不再生成子节点。即为叶子节点 。一般不推荐改动默认值1e-7。

基本gdbt实现

```
import pandas as pd
import numpy as np
from sklearn.ensemble import GradientBoostingClassifier
from sklearn import metrics
from sklearn.model_selection import cross_validate
from sklearn.model_selection import GridSearchCV

import matplotlib.pylab as plt

train = pd.read_csv('train_modified.csv')
# 第一列是二元分类的输出
target = 'Disbursed'
IDcol = 'ID'
# 查看Disbursed列有多少重复值 pandas.value_counts()
print(train['Disbursed'].value_counts())

x_columns = [x for x in train.columns if x not in [target,IDcol]]
X = train[x_columns]
y = train['Disbursed']

# 固定random_state，使每次相同参数，数据集，测试集输出一样。
gbm0 = GradientBoostingClassifier(random_state=10)
gbm0.fit(X,y)
y_pred = gbm0.predict(X)
y_predprob = gbm0.predict_proba(X)[:,1]
```

```
print("Accuracy:%.4g" % metrics.accuracy_score(y.values,y_pred))
print("AUC Score(Train): %f" % metrics.roc_auc_score(y,y_predprob))
```

Accuracy : 0.9852
AUC Score (Train): 0.900531

网格搜索调参

1.gridSearchCV网格搜索介绍

它存在的意义就是自动调参，只要把参数输进去，就能给出最优化的结果和参数。但是这个方法适合于小数据集，一旦数据的量级上去了，很难得出结果。

```
class sklearn.model_selection.GridSearchCV(estimator, param_grid, scoring=None,
fit_params=None, n_jobs=1, iid=True, refit=True, cv=None, verbose=0,
pre_dispatch='2*n_jobs', error_score='raise', return_train_score='warn')
```

- estimator: 使用的分类器，并且传入除需要确定最佳的参数之外的其他参数。每一个分类器都需要一个scoring参数，或者score方法：estimator=RandomForestClassifier(min_samples_split=100,min_samples_leaf=20,max_depth=8,max_features='sqrt',random_state=10)
- param_grid： 需要最优化的参数的取值，值为字典或者列表，例如：param_grid =param_test1，param_test1 = {'n_estimators':range(10,71,10)}。
- scoring=None： 模型评价标准，默认None,这时需要使用score函数；或者如scoring='roc_auc'，根据所选模型不同，评价准则不同。字符串（函数名），或是可调用对象，需要其函数签名形如：scorer(estimator, X, y)；如果是None，则使用estimator的误差估计函数。
- fit_params=None
- n_jobs=1：n_jobs: 并行数，int：个数，-1：跟CPU核数一致, 1:默认值
- iid=True： iid:默认True,为True时，默认为各个样本fold概率分布一致，误差估计为所有样本之和，而非各个fold的平均。
- refit=True：默认为True,程序将会以交叉验证训练集得到的最佳参数，重新对所有可用的训练集与开发集进行，作为最终用于性能评估的最佳模型参数。即在搜索参数结束后，用最佳参数结果再次fit一遍全部数据集。
- cv=None: 交叉验证参数，默认None，使用三折交叉验证。指定fold数量，默认为3，也可以是yield训练/测试数据的生成器。
- verbose=0, scoring=None :verbose：日志冗长度，int：冗长度，0：不输出训练过程，1：偶尔输出，>1：对每个子模型都输出。
- pre_dispatch='2*n_jobs': 指定总共分发的并行任务数。当n_jobs大于1时，数据将在每个运行点进行复制，这可能导致OOM，而设置pre_dispatch参数，则可以预先划分总共的job数量，使数据最多被复制pre_dispatch次
- error_score='raise':
- return_train_score='warn': 如果"False"，cv_results_属性将不包括训练分数

2.学习率（lr），迭代次数(n_estimators)

```
param_test1 = {'n_estimators':range(20,81,10)}
gsearch1 = GridSearchCV(estimator =
GradientBoostingClassifier(learning_rate=0.1, min_samples_split=300,
min_samples_leaf=20,max_depth=8,max_features='sqrt',subsample=0.8,random_state=1
0), param_grid = param_test1, scoring='roc_auc',iid=False,cv=5)
gsearch1.fit(X,y)
gsearch1.grid_scores_, gsearch1.best_params_, gsearch1.best_score_
```

结果显示60的时候效果最好。

3.决策树最大深度(max_depth),内部节点再划分最小样本数(min_samples_split)

固定了上一步结果的lr，和迭代次数

```
param_test2 = {'max_depth':range(3,14,2),
'min_samples_split':range(100,801,200)}
gsearch2 = GridSearchCV(estimator =
GradientBoostingClassifier(learning_rate=0.1, n_estimators=60,
min_samples_leaf=20,
      max_features='sqrt', subsample=0.8, random_state=10),
   param_grid = param_test2, scoring='roc_auc',iid=False, cv=5)
gsearch2.fit(X,y)
print(gsearch2.cv_results_)
print(gsearch2.best_params_)
print(gsearch2.best_score_)
```

结果：最大树深7，内部节点再划分最小样本数300

{'max_depth': 7, 'min_samples_split': 300}
0.8213724275914632

4.内部节点再划分所需最小样本数min_samples_split和叶子节点最少样本数min_samples_leaf

内部节点再划分所需最小样本数min_samples_split和其他参数——叶子节点最少样本数min_samples_leaf有关联，所以需要一起调参

```
param_test3 = {'min_samples_split':range(800,1900,200),
'min_samples_leaf':range(60,101,10)}
gsearch3 = GridSearchCV(estimator =
GradientBoostingClassifier(learning_rate=0.1, n_estimators=60,max_depth=7,
                                     max_features='sqrt', subsample=0.8,
random_state=10),
                        param_grid = param_test3, scoring='roc_auc',iid=False,
cv=5)
gsearch3.fit(X,y)
print(gsearch3.cv_results_)
print(gsearch3.best_params_)
print(gsearch3.best_score_)
```

```
{'min_samples_leaf': 60, 'min_samples_split': 1200},
 0.8222032996697154)
```

综上，再实验：

```
gbm1 = GradientBoostingClassifier(learning_rate=0.1,
n_estimators=60,max_depth=7, min_samples_leaf =60,
               min_samples_split =1200, max_features='sqrt', subsample=0.8,
random_state=10)
gbm1.fit(X,y)
y_pred = gbm1.predict(X)
y_predprob = gbm1.predict_proba(X)[:,1]
print("Accuracy:%.4g" % metrics.accuracy_score(y.values,y_pred))
print("AUC Score(Train): %f" % metrics.roc_auc_score(y,y_predprob))
```

Accuracy:0.984
AUC Score(Train): 0.908099

精度略有下降，主要原理是我们使用了0.8的子采样，20%的数据没有参与拟合。

5.最大特征数max_features

```
param_test4 = {'max_features':range(7,20,2)}
gsearch4 = GridSearchCV(estimator =
GradientBoostingClassifier(learning_rate=0.1, n_estimators=60,max_depth=7,
min_samples_leaf =60,
               min_samples_split =1200, subsample=0.8, random_state=10),
                       param_grid = param_test4, scoring='roc_auc',iid=False,
cv=5)
gsearch4.fit(X,y)
print(gsearch4.cv_results_)
print(gsearch4.best_params_)
print(gsearch4.best_score_)
```

{'max_features': 9},
0.822412506351626)

6.子采样比例

```
param_test5 = {'subsample':[0.6,0.7,0.75,0.8,0.85,0.9]}
gsearch5 = GridSearchCV(estimator =
GradientBoostingClassifier(learning_rate=0.1, n_estimators=60,max_depth=7,
min_samples_leaf =60,
               min_samples_split =1200, max_features=9, random_state=10),
                       param_grid = param_test5, scoring='roc_auc',iid=False,
cv=5)
gsearch5.fit(X,y)
print(gsearch5.cv_results_)
print(gsearch5.best_params_)
print(gsearch5.best_score_)
```

{'subsample': 0.7},
0.8234378969766262)

再实验：

```
gbm2 = GradientBoostingClassifier(learning_rate=0.05,
n_estimators=120,max_depth=7, min_samples_leaf =60,
               min_samples_split =1200, max_features=9, subsample=0.7,
random_state=10)
gbm2.fit(X,y)
y_pred = gbm2.predict(X)
y_predprob = gbm2.predict_proba(X)[:,1]
print("Accuracy:%.4g" % metrics.accuracy_score(y.values,y_pred))
print("AUC Score(Train): %f" % metrics.roc_auc_score(y,y_predprob))
```

Accuracy:0.984
AUC Score(Train): 0.905324

可以修改lr和n_estimators，约数：乘积保持不变

图形绘制

```
dot_data = StringIO()
tree.export_graphviz(gbm2.estimators_[0, 0],
                     out_file=dot_data,
                     node_ids=True,
                     filled=True,
                     rounded=True,
                     special_characters=True)

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())

graph.write_pdf("gbdt.pdf")
```

生成的可视化图