1. Random Forest and Gradient Tree Boosting parameters

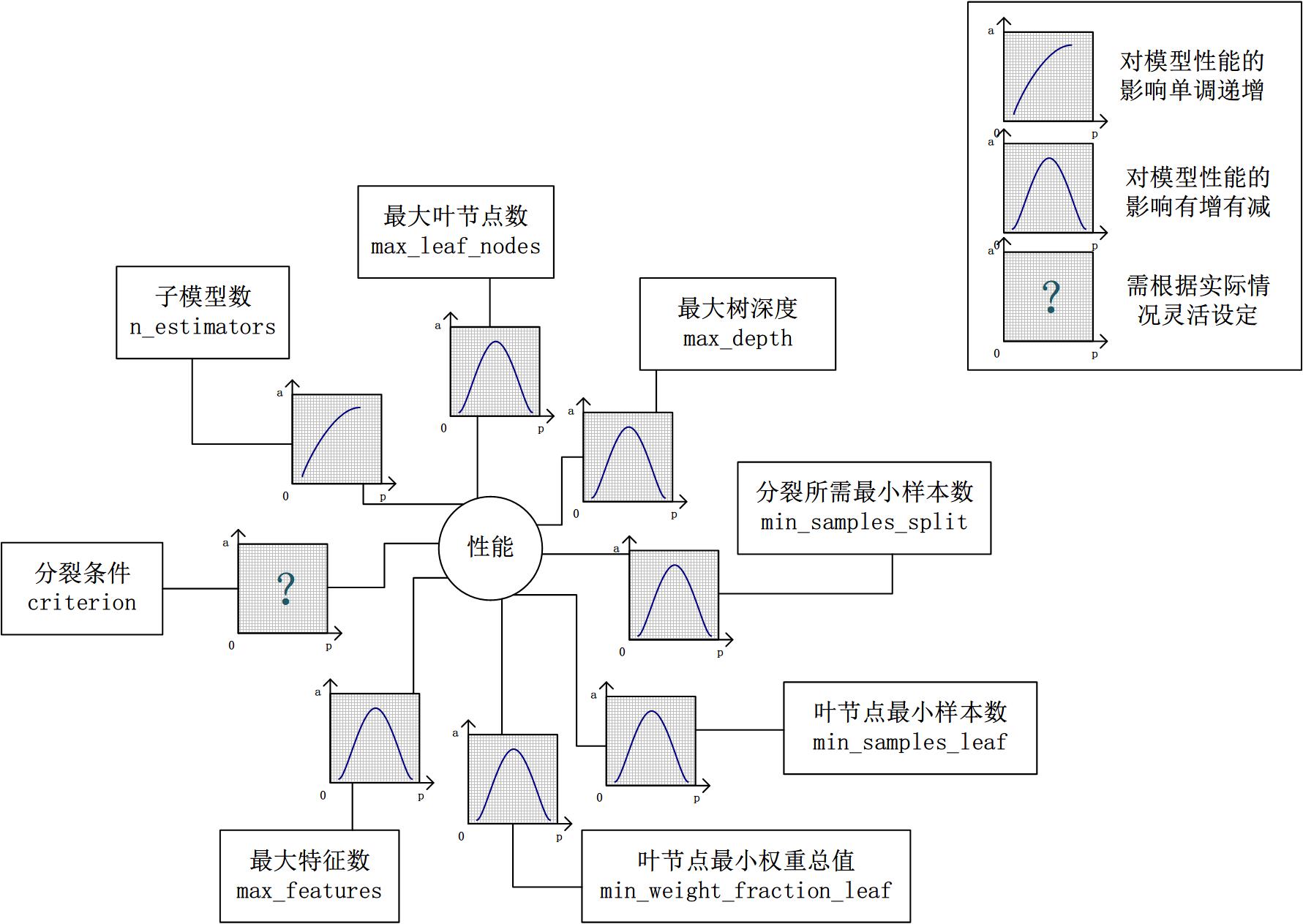
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **参数** | **类型** | **RandomForestClassifier** | **RandomForestRegressor** | **GradientBoostingClassifier** | **GradientBoostingRegressor** |
| loss | 目标 |  |  | 损失函数  ● exponential：模型等同AdaBoost  ★ deviance：和Logistic Regression的损失函数一致 | 损失函数  ● exponential：模型等同AdaBoost  ★ deviance：和Logistic Regression的损失函数一致 |
| alpha | 目标 |  |  | 损失函数为huber或quantile的时，alpha为损失函数中的参数 | 损失函数为huber或quantile的时，alpha为损失函数中的参数 |
| class\_weight | 目标 | 类别的权值 |  |  |  |
| n\_estimators | 性能 | 子模型的数量  ● int：个数  ★ 10：默认值 | 子模型的数量  ● int：个数  ★ 10：默认值 | 子模型的数量  ● int：个数  ★ 100：默认值 | 子模型的数量  ● int：个数  ★ 100：默认值 |
| learning\_rate  对于同样的训练集拟合效果，较小的νν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νν开始调参，默认是1。 | 性能 |  |  | 学习率（缩减） | 学习率（缩减） |
| criterion | 性能 | 判断节点是否继续分裂采用的计算方法  ● entropy  ★ gini | 判断节点是否继续分裂采用的计算方法  ★ mse |  |  |
| max\_features  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。 | 性能 | 节点分裂时参与判断的最大特征数  ● int：个数  ● float：占所有特征的百分比  ★ auto：所有特征数的开方  ● sqrt：所有特征数的开方  ● log2：所有特征数的log2值  ● None：等于所有特征数 | 节点分裂时参与判断的最大特征数  ● int：个数  ● float：占所有特征的百分比  ★ auto：所有特征数的开方  ● sqrt：所有特征数的开方  ● log2：所有特征数的log2值  ● None：等于所有特征数 | 节点分裂时参与判断的最大特征数  ● int：个数  ● float：占所有特征的百分比  ● auto：所有特征数的开方  ● sqrt：所有特征数的开方  ● log2：所有特征数的log2值  ★ None：等于所有特征数 | 节点分裂时参与判断的最大特征数  ● int：个数  ● float：占所有特征的百分比  ● auto：所有特征数的开方  ● sqrt：所有特征数的开方  ● log2：所有特征数的log2值  ★ None：等于所有特征数 |
| max\_depth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 | 性能 | 最大深度，如果max\_leaf\_nodes参数指定，则忽略  ● int：深度  ★ None：树会生长到所有叶子都分到一个类，或者某节点所代表的样本数已小于min\_samples\_split | 最大深度，如果max\_leaf\_nodes参数指定，则忽略  ● int：深度  ★ None：树会生长到所有叶子都分到一个类，或者某节点所代表的样本数已小于min\_samples\_split | 最大深度，如果max\_leaf\_nodes参数指定，则忽略  ● int：深度  ★ 3：默认值 | 最大深度，如果max\_leaf\_nodes参数指定，则忽略  ● int：深度  ★ 3：默认值 |
| min\_samples\_split  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。 | 性能 | 分裂所需的最小样本数  ● int：样本数  ★ 2：默认值 | 分裂所需的最小样本数  ● int：样本数  ★ 2：默认值 | 分裂所需的最小样本数  ● int：样本数  ★ 2：默认值 | 分裂所需的最小样本数  ● int：样本数  ★ 2：默认值 |
| min\_samples\_leaf | 性能 | 叶节点最小样本数  ● int：样本数  ★ 1：默认值 | 叶节点最小样本数  ● int：样本数  ★ 1：默认值 | 叶节点最小样本数  ● int：样本数  ★ 1：默认值 | 叶节点最小样本数  ● int：样本数  ★ 1：默认值 |
| min\_weight\_fraction\_leaf | 性能 | 叶节点最小样本权重总值  ● float：权重总值  ★ 0：默认值 | 叶节点最小样本权重总值  ● float：权重总值  ★ 0：默认值 | 叶节点最小样本权重总值  ● float：权重总值  ★ 0：默认值 | 叶节点最小样本权重总值  ● float：权重总值  ★ 0：默认值 |
| max\_leaf\_nodes  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。 | 性能 | 最大叶节点数  ● int：个数  ★ None：不限制叶节点数 | 最大叶节点数  ● int：个数  ★ None：不限制叶节点数 | 最大叶节点数  ● int：个数  ★ None：不限制叶节点数 | 最大叶节点数  ● int：个数  ★ None：不限制叶节点数 |
| bootstrap | 性能 | 是否bootstrap对样本抽样  ● False：子模型的样本一致，子模型间强相关  ★ True：默认值 | 是否bootstrap对样本抽样  ● False：子模型的样本一致，子模型间强相关  ★ True：默认值 |  |  |
| Subsample  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] | 性能 |  |  | 子采样率  ● float：采样率  ★ 1.0：默认值 | 子采样率  ● float：采样率  ★ 1.0：默认值 |
| Init | 性能 |  |  | 初始子模型 | 初始子模型 |
| n\_jobs | 效率 | 并行数  ● int：个数  ● -1：跟CPU核数一致  ★ 1:默认值 | 并行数  ● int：个数  ● -1：跟CPU核数一致  ★ 1:默认值 |  |  |
| warm\_start | 效率 | 是否热启动，如果是，则下一次训练是以追加树的形式进行  ● bool：热启动  ★ False：默认值 | 是否热启动，如果是，则下一次训练是以追加树的形式进行  ● bool：热启动  ★ False：默认值 | 是否热启动，如果是，则下一次训练是以追加树的形式进行  ● bool：热启动  ★ False：默认值 | 是否热启动，如果是，则下一次训练是以追加树的形式进行  ● bool：热启动  ★ False：默认值 |
| presort | 效率 |  |  | 是否预排序,预排序可以加速查找最佳分裂点，对于稀疏数据不管用  ● Bool  ★ auto：非稀疏数据则预排序，若稀疏数据则不预排序 | 是否预排序,预排序可以加速查找最佳分裂点，对于稀疏数据不管用  ● Bool  ★ auto：非稀疏数据则预排序，若稀疏数据则不预排序 |
| oob\_score | 附加 | 是否计算[袋外得分](http://scikit-learn.org/stable/auto_examples/ensemble/plot_ensemble_oob.html)  ★ False：默认值 | 是否计算[袋外得分](http://scikit-learn.org/stable/auto_examples/ensemble/plot_ensemble_oob.html)  ★ False：默认值 |  |  |
| random\_state | 附加 | 随机器对象 | 随机器对象 | 随机器对象 | 随机器对象 |
| verbose | 附加 | 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 | 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 | 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 | 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 |

## Random forest – low bias, aim to decrease variance

1. smaller number of estimators ( increase n\_estimator helps )
2. sub-trees are not weak learners (no restrictions on the max\_depth)
   1. max depth and max\_leaf\_nodes (deeper trees or more leaf nodes mean lower variance and higher bias) – broader tuning
   2. min\_samples\_split, min\_samples\_leaf and min\_weight\_fraction\_leaf

(smaller min sample split or smaller min sample leaf mean more complicated models)

1. Decorrelate trees would help decrease the variance (max\_features set at auto, choose decrease max\_features can decrease variance but increase bias)
2. Criterion may have different effects



print rf0.oob\_score\_

我们首先对n\_estimators进行网格搜索：

[复制代码](javascript:void(0);)

param\_test1 = {'n\_estimators':range(10,71,10)}

gsearch1 = GridSearchCV(estimator = RandomForestClassifier(min\_samples\_split=100,

min\_samples\_leaf=20,max\_depth=8,max\_features='sqrt' ,random\_state=10),

param\_grid = param\_test1, scoring='roc\_auc',cv=5)

gsearch1.fit(X,y)

gsearch1.grid\_scores\_, gsearch1.best\_params\_, gsearch1.best\_score\_

接着我们对决策树最大深度max\_depth和内部节点再划分所需最小样本数min\_samples\_split进行网格搜索。

param\_test2 = {'max\_depth':range(3,14,2), 'min\_samples\_split':range(50,201,20)}

gsearch2 = GridSearchCV(estimator = RandomForestClassifier(n\_estimators= 60,

min\_samples\_leaf=20,max\_features='sqrt' ,oob\_score=True, random\_state=10),

param\_grid = param\_test2, scoring='roc\_auc',iid=False, cv=5)

gsearch2.fit(X,y)

gsearch2.grid\_scores\_, gsearch2.best\_params\_, gsearch2.best\_score\_

对于内部节点再划分所需最小样本数min\_samples\_split，我们暂时不能一起定下来，因为这个还和决策树其他的参数存在关联。下面我们再对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一起调参。

param\_test3 = {'min\_samples\_split':range(80,150,20), 'min\_samples\_leaf':range(10,60,10)}

gsearch3 = GridSearchCV(estimator = RandomForestClassifier(n\_estimators= 60, max\_depth=13,

max\_features='sqrt' ,oob\_score=True, random\_state=10),

param\_grid = param\_test3, scoring='roc\_auc',iid=False, cv=5)

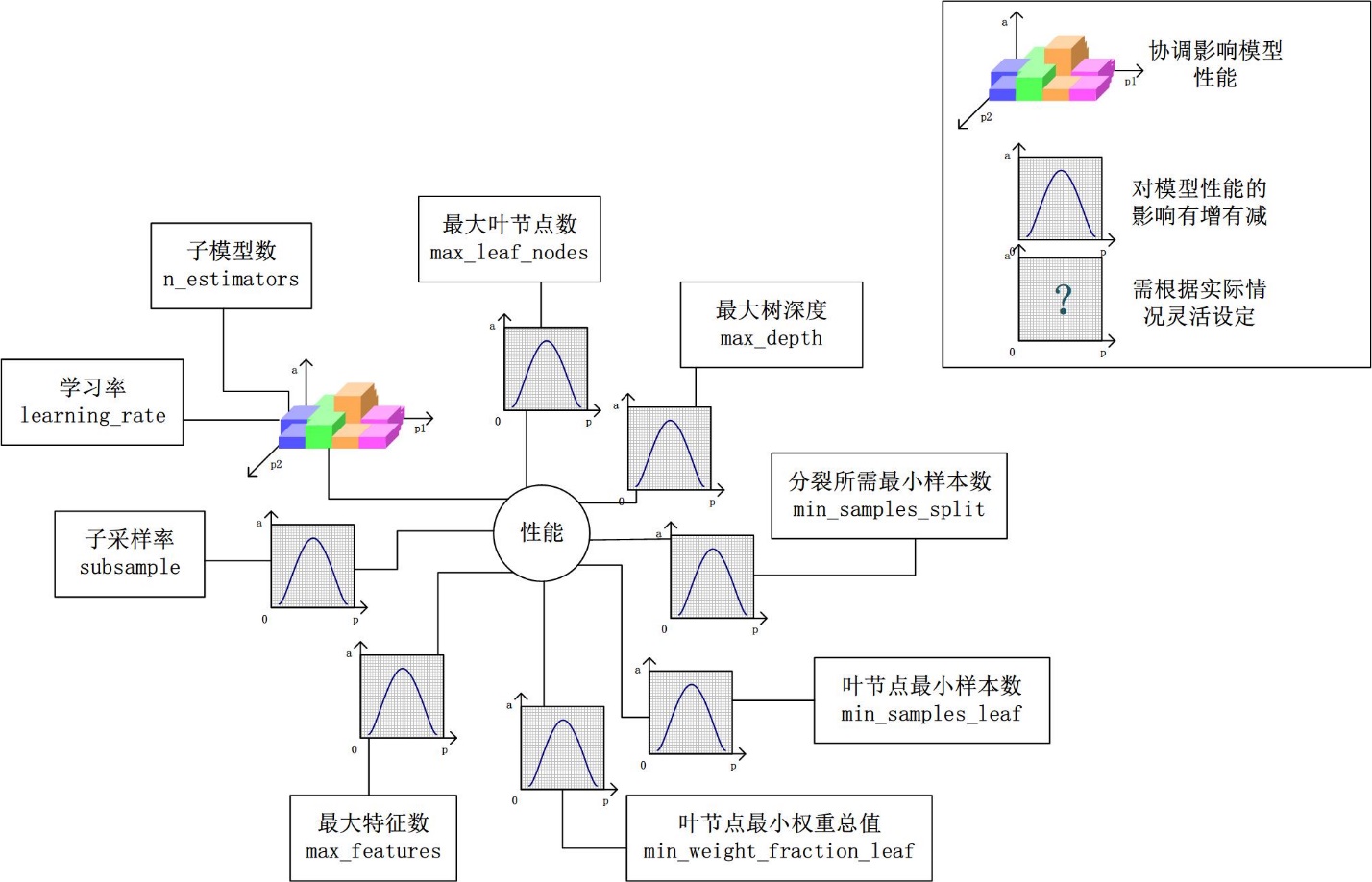
gsearch3.fit(X,y)

gsearch3.grid\_scores\_, gsearch3.best\_params\_, gsearch3.best\_score\_

## Gradient boosting low variance high bias

Need more n\_estimators, weak learners (max\_depth =3)

1. Tune n\_estimators and learning rate together
2. Max depth, min\_samples,split (set as small)
3. Max\_features, subsample ( decorrelate doesn’t help much)



1. How to tune (an optimization method)
   1. Objective: bias and variance trade off
2. Determine the object hyperparameters first
3. Parameters that affect the Training process: n\_estimators and learning rate (important for boosting)
4. Parameters that affect base learners: max\_depth, criterion
   1. What parameters affect the objective, positive or negative and how big

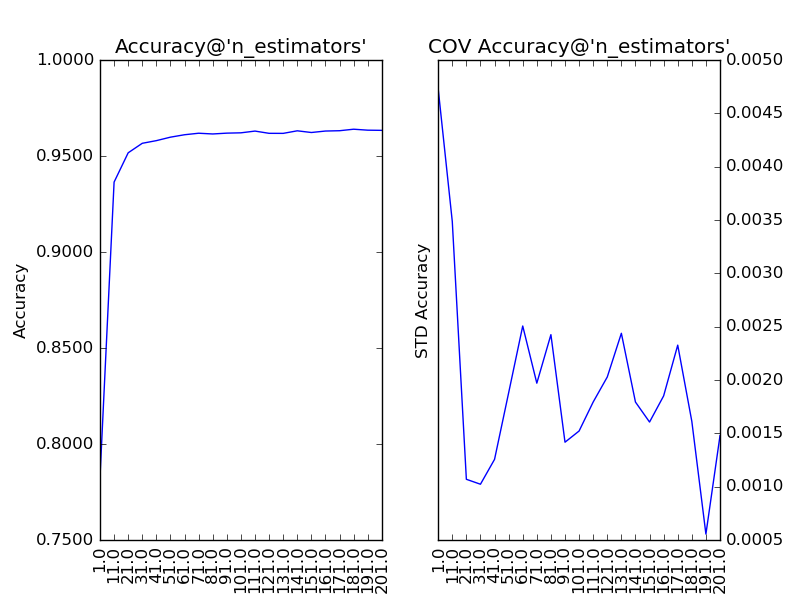
Improve the efficiency in the training

* 1. coordinate descent and greedy approach

don’t need to calculate the gradient

贪心地选取了对整体模型性能影响最大的参数。参数对整体模型性能的影响力是动态变化的，故每一轮坐标选取的过程中，这种方法在对每个坐标的下降方向进行一次直线搜索（line search）。首先，找到那些能够提升整体模型性能的参数，其次确保提升是单调或近似单调的。这意味着，我们筛选出来的参数是对整体模型性能有正影响的，且这种影响不是偶然性的，要知道，训练过程的随机性也会导致整体模型性能的细微区别，而这种区别是不具有单调性的。最后，在这些筛选出来的参数中，选取影响最大的参数进行调整即可。

　　无法对整体模型性能进行量化，也就谈不上去比较参数影响整体模型性能的程度。是的，我们还没有一个准确的方法来量化整体模型性能，只能通过交叉验证来近似计算整体模型性能。然而交叉验证也存在随机性，假设我们以验证集上的平均准确度作为整体模型的准确度，我们还得关心在各个验证集上准确度的变异系数，如果变异系数过大，则平均值作为整体模型的准确度也是不合适的。在接下来的案例分析中，我们所谈及的整体模型性能均是指平均准确度，请各位留心。

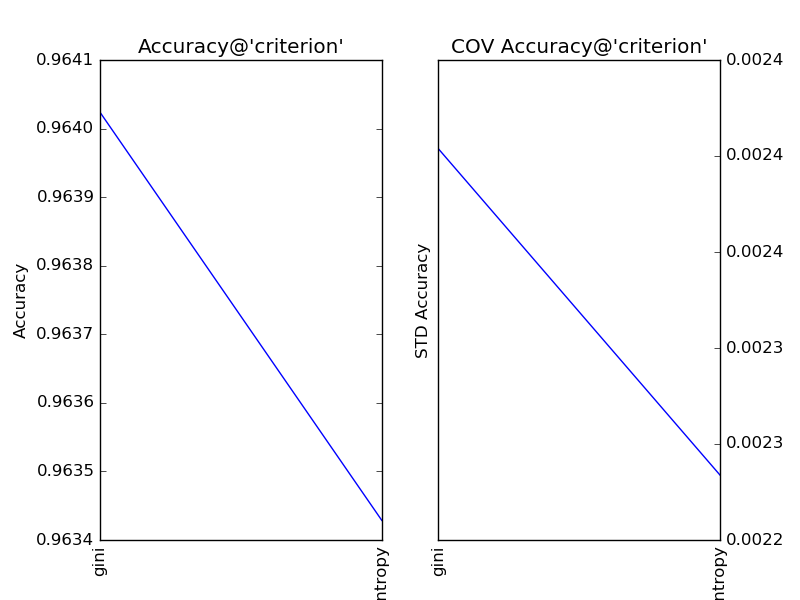


当“子模型数”增加到40以上时，准确度的提升逐渐不明显。考虑到训练的效率，最终我们选择“子模型数”为200。

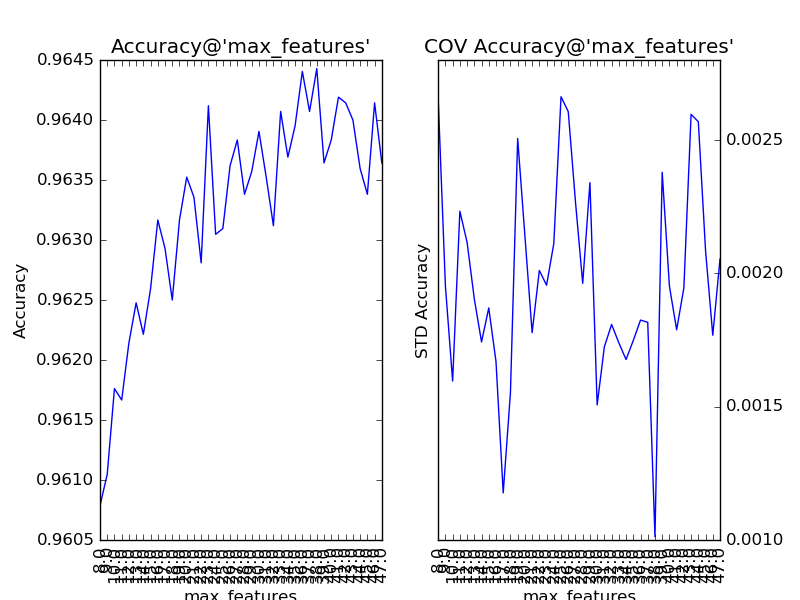
所以一般选择一个适中的数值。默认是100

保持n estimator 为200 ，调参gini和entrophy

max\_depth: 默认可以不输入，如果不输入的话，决策树在建立子树的时候不会限制子树的深度。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。

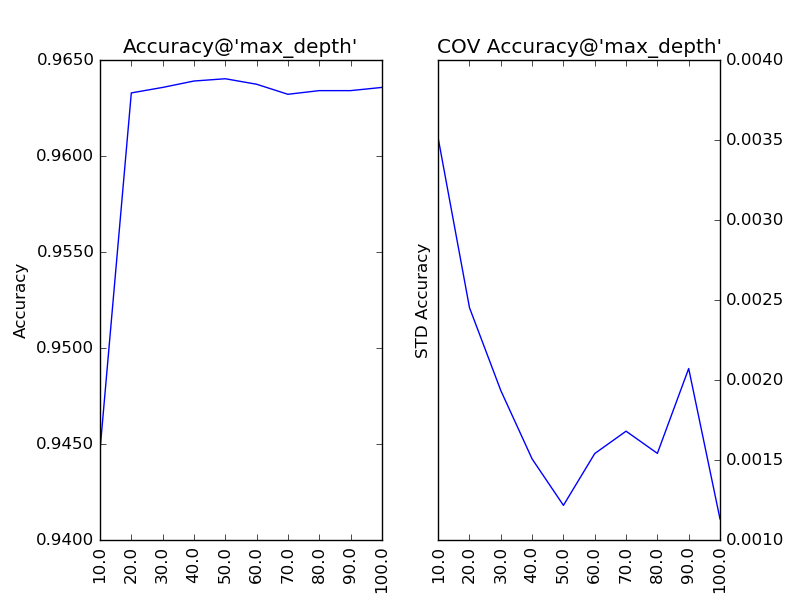


对“分裂时参与判断的最大特征数”（max\_feature）以1为单位，设定取值范围为28至47，得到调参结果如下

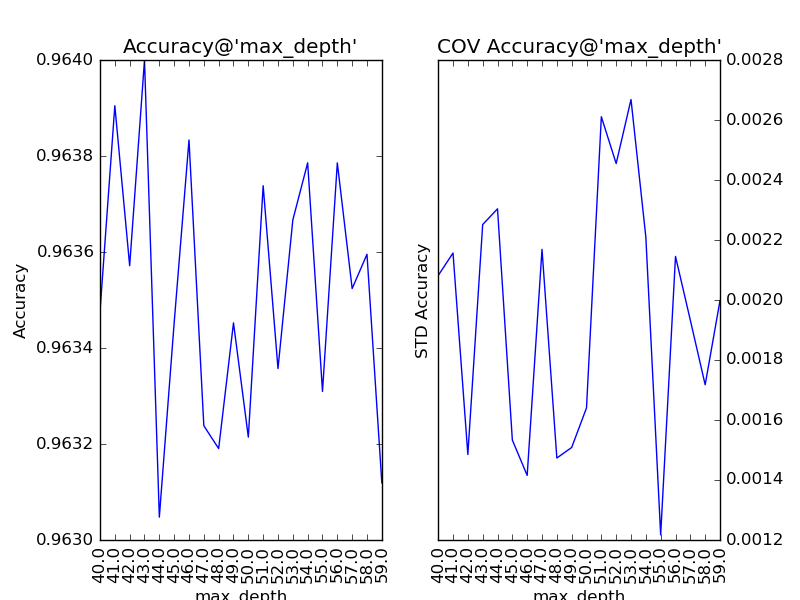


“分裂时参与判断的最大特征数”的默认值auto，即总特征数（sqrt(784)=28）的开方。通过提升该参数，整体模型的准确度得到了提升。可见，该参数的默认值过小，导致了子模型的偏差过大，从而整体模型的偏差过大。同时，我们还注意到，该参数对整体模型性能的影响是近似单调的：从28到38，模型的准确度逐步抖动提升。所以，我们可考虑将该参数纳入下一步的调参工作。

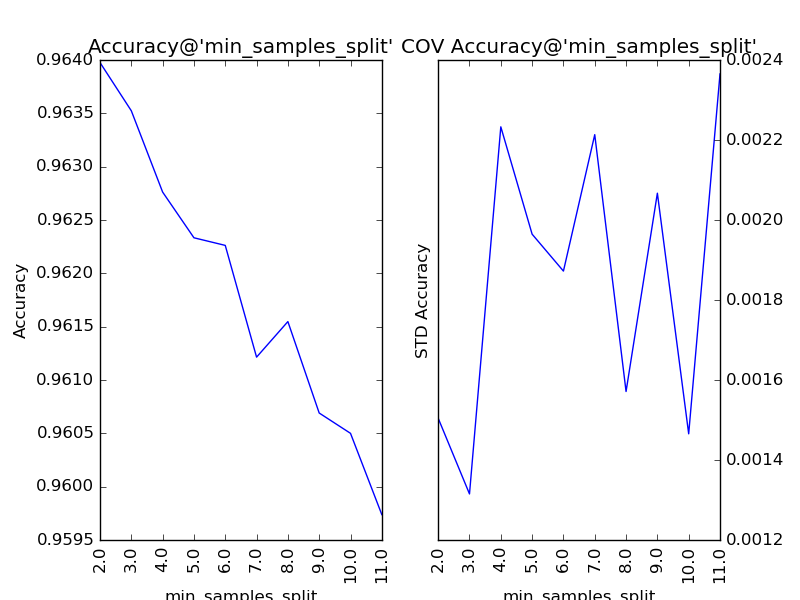
　　对“最大深度”（max\_depth）以10为单位，设定取值范围为10到100，得到调参结果如下：



随着树的深度加深，子模型的偏差减少，整体模型的准确度得到提升。从理论上来说，子模型训练的后期，随着方差增大，子模型的准确度稍微降低，从而影响整体模型的准确度降低。看图中，似乎取值范围从40到60的情况可以印证这一观点。不妨以1为单位，设定取值范围为40到59，更加细致地分析：

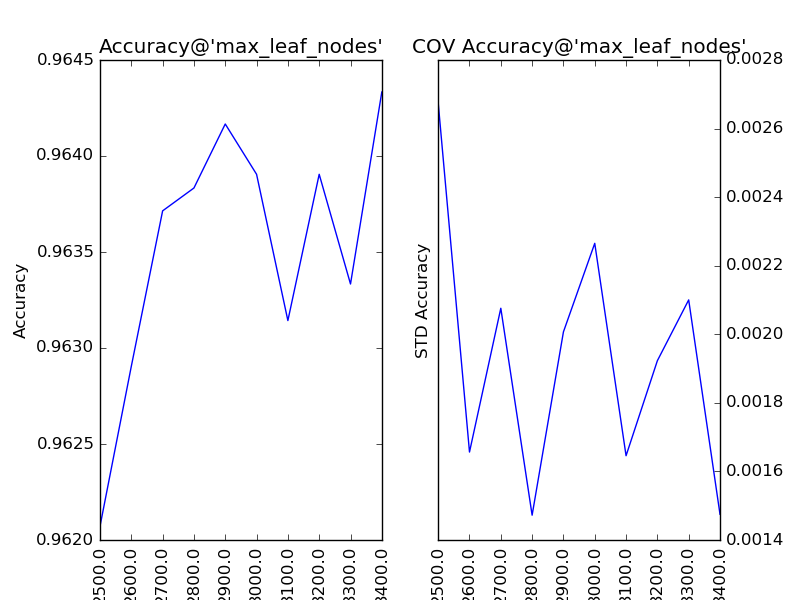


　有点傻眼了，怎么跟预想的不太一样？为什么模型准确度的变化在40到59之间没有鲜明的“规律”了？要分析这个问题，我们得先思考一下，少一层子节点对子模型意味着什么？若少的那一层给原子模型带来的是方差增大，则新子模型会准确度提高；若少的那一层给原子模型带来的是偏差减小，则新子模型会准确度降低。所以，细粒度的层次变化既可能使整体模型的准确度提升，也可能使整体模型的准确度降低。从而也说明了，该参数更适合进行粗粒度的调整。在训练的现阶段，“抖动”现象的发生说明，此时对该参数的调整已不太合适了。



　我们看到，随着分裂所需的最小样本数的增加，子模型的结构变得越来越简单，理论上来说，首先应当因方差减小导致整体模型的准确度提升。但是，在训练的现阶段，子模型的偏差增大的幅度比方差减小的幅度更大，所以整体模型的准确度持续下降。该参数的默认值为2，调参后，最优解保持2不变。

对“最大叶节点数”（max\_leaf\_nodes）以100为单位，设定取值范围为2500到3400，得到调参结果如下：



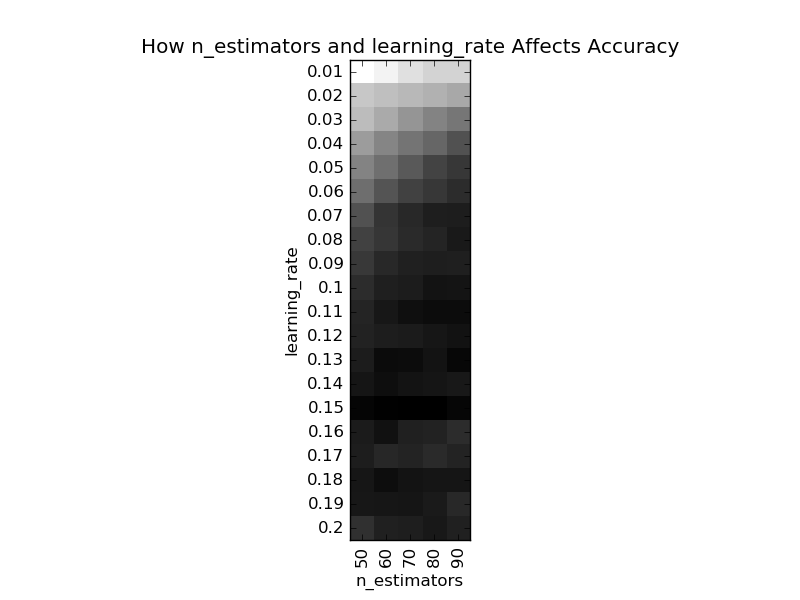
抖动就不调了

如果您以前编写过一个决策树，你能体会到最小样本叶片大小的重要性。 叶是决策树的末端节点。 较小的叶子使模型更容易捕捉训练数据中的噪声。 一般来说，我更偏向于将最小叶子节点数目设置为大于50。

Gradient boosting :

GridSearchCV找到关于这（n\_estimators）和“学习率”（learning\_rate两个参数的最优解

在前期会比较高，在后期会比较低，如果一开始我们将这两个参数调成最优，这样很容易陷入一个“局部最优解”。在目标函数都不确定的情况下（如是否凸？），谈局部最优解就是耍流氓，本文中“局部最优解”指的是调整各参数都无明显性能提升的一种状态，所以打了引号。下图中展示了这个两个参数的调参结果：



Local optimal

现在我们可以回过头来，调整这两个参数，调整的方法为成倍地放大“子模型数”，对应成倍地缩小“学习率”（learning\_rate）

[Aarshay Jain](https://www.analyticsvidhya.com/blog/author/aarshay/)对Gradient Tree Boosting总结了一套[调参方法](https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-python/)，其核心思想在于：对过程影响类参数进行调整，毕竟它们对整体模型性能的影响最大，然后依据经验，在其他参数中选择对整体模型性能影响最大的参数，进行下一步调参。

这种方法的关键是依照对整体模型性能的影响力给参数排序，然后按照该顺序对的参数进行调整。如何衡量参数对整体模型性能的影响力呢？基于经验，Aarshay提出他的见解：“最大叶节点数”（max\_leaf\_nodes）和“最大树深度”（max\_depth）对整体模型性能的影响大于“分裂所需最小样本数”（min\_samples\_split）、“叶节点最小样本数”（min\_samples\_leaf）及“叶节点最小权重总值”（min\_weight\_fraction\_leaf），而“分裂时考虑的最大特征数”（max\_features）的影响力最小

但是，对Gradient Tree Boosting调参时，遇到“局部最优解”的可能性就大得多。

Aarshay按照其定义的参数对整体模型性能的影响力，按序依次对参数进行调整。

在依次对参数进行调整时，还是需要像贪心的坐标下降法中一样对参数的“动态”影响力进行分析一下，如果这种影响力是“抖动”的，可有可无的，那么我们就不需要对该参数进行调整。

* 1. Class imbalance

直接先调试“最大深度”（max\_depth），会发现其会保持默认值3作为最优解这是因为，正例样本远远小于反例，所以在低深度时，子模型就可能已经对正例过拟合了

所以，在类别不均衡时，只有先确定“叶节点最小样本数”（min\_samples\_leaf），再确定“分裂所需最小样本数”（min\_samples\_split），才能确定“最大深度”。而Aarshay设定的初始值，则以经验和直觉避开了这个险恶的陷阱。

首先，我们需要初步地调一次“子采样率”（subsample）和“分裂时考虑的最大特征数”（max\_features），在此基础上依次调好“叶节点最小样本数”（min\_samples\_leaf）、“分裂所需最小样本数”（min\_samples\_split）以及“最大深度”（max\_depth）。然后，按照Aarshay的方法，按影响力从大到小再调一次

<https://github.com/ljpzzz/machinelearning/blob/master/ensemble-learning/gbdt_classifier.ipynb>

首先我们从步长(learning rate)和迭代次数(n\_estimators)入手。一般来说,开始选择一个较小的步长来网格搜索最好的迭代次数。这里，我们将步长初始值设置为0.1。对于迭代次数进行网格搜索如下：

[复制代码](javascript:void(0);)

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=10),

param\_grid = param\_test1, scoring='roc\_auc',iid=False,cv=5)

gsearch1.fit(X,y)

gsearch1.grid\_scores\_, gsearch1.best\_params\_, gsearch1.best\_score\_

找到了一个合适的迭代次数，现在我们开始对决策树进行调参。首先我们对决策树最大深度max\_depth和内部节点再划分所需最小样本数min\_samples\_split进行网格搜索。

[复制代码](javascript:void(0);)

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)

gsearch2.grid\_scores\_, gsearch2.best\_params\_, gsearch2.best\_score\_

　由于决策树深度7是一个比较合理的值，我们把它定下来，对于内部节点再划分所需最小样本数min\_samples\_split，我们暂时不能一起定下来，因为这个还和决策树其他的参数存在关联。下面我们再对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一起调参。

[复制代码](javascript:void(0);)

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)

gsearch3.grid\_scores\_, gsearch3.best\_params\_, gsearch3.best\_score\_

　　现在我们再对最大特征数max\_features进行网格搜索。

[复制代码](javascript:void(0);)

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)

gsearch4.grid\_scores\_, gsearch4.best\_params\_, gsearch4.best\_score\_

现在我们再对子采样的比例进行网格搜索：

[复制代码](javascript:void(0);)

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)

gsearch5.grid\_scores\_, gsearch5.best\_params\_, gsearch5.best\_score\_

现在我们基本已经得到我们所有调优的参数结果了。这时我们可以减半步长，最大迭代次数加倍来增加我们模型的泛化能力。再次拟合我们的模型：

[复制代码](javascript:void(0);)

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)

下面我们继续将步长缩小5倍，最大迭代次数增加5倍，继续拟合我们的模型：

[复制代码](javascript:void(0);)

gbm3 = GradientBoostingClassifier(learning\_rate=0.01, n\_estimators=600,max\_depth=7, min\_samples\_leaf =60,

min\_samples\_split =1200, max\_features=9, subsample=0.7, random\_state=10)

gbm3.fit(X,y)

y\_pred = gbm3.predict(X)

y\_predprob = gbm3.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)

最后我们继续步长缩小一半，最大迭代次数增加2倍，拟合我们的模型：

[复制代码](javascript:void(0);)

gbm4 = GradientBoostingClassifier(learning\_rate=0.005, n\_estimators=1200,max\_depth=7, min\_samples\_leaf =60,

min\_samples\_split =1200, max\_features=9, subsample=0.7, random\_state=10)

gbm4.fit(X,y)

y\_pred = gbm4.predict(X)

y\_predprob = gbm4.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)

**H2O 的用法**

在 R 和 Python 里 H2O 的下载安装都很容易，只需安装一个新的 h2o 包即可。安装后，首先要初始化一个 H2O instance，连接到 **H2O.ai** 的分布式系统 H2O cluster。随后数据存储和计算都是在 H2O cluster 上完成的。

1. import h2o
2. h2o.init()
3. from h2o.estimators.random\_forest import H2ORandomForestEstimator
4. df=h2o.import\_file('train.csv')

model=H2ORandomForestEstimator(ntrees=6,max\_depth =16)

model.train(x=trainData.names,y='Catrgory',training\_frame=trainData)

df2=h2o.import\_file('test.csv')

test\_data=df2[2:]

pre\_tag=H2ORandomForestEstimator.predict(model ,test\_data)

predict=df2.concat(pre\_tag)

dfnew=predict[predict['Catrgory']==predict['predict']]

Precision=dfnew.nrow/predict.nrow