**参数选择原因：**

* num\_leaves: 可以设置每个弱学习者拥有的叶子的最大数量。较大的numleaves增加了训练集的精确度，也增加了因过度拟合而受伤的几率
* max\_depth：该参数控制每棵经过训练的树的最大深度
* learning\_rate：学习率
* feature\_fraction：特征分数或子特征处理列采样，LightGBM将在每次迭代(树)上随机选择特征子集。例如，如果将其设置为0.6,LightGBM将在训练每棵树之前选择60%的特性。
* bagging\_fraction：可以指定每个树构建迭代使用的行数百分比。这意味着将随机选择一些行来匹配每个学习者(树)。这不仅提高了泛化能力，也提高了训练速度。
* bagging\_freq：Bagging的频率，多少个迭代里做一次
* lambda\_l1：范式1
* lambda\_l2: 范式2
* is\_unbalance：算法将尝试自动平衡占主导地位的标签的权重(使用列集中的pos/neg分数)

**jupyter notebook**:

https://github.com/LoosonWu/geek\_homework/blob/main/lightgbm.ipynb

# Round 1: 只选取前面15个feature

**params** = params = {'boosting\_type': 'gbdt','objective': 'binary','metric': {'binary\_logloss', 'auc'},'num\_leaves': 16,'max\_depth': 4,'learning\_rate': 0.01,'feature\_fraction': 0.9,'bagging\_fraction': 0.95,'bagging\_freq': 5,'lambda\_l1': 1,'lambda\_l2': 0.001,'is\_unbalance': True}

**Accuracy**: 0.89608

# Round 2: 在同一个learning rate里调整叶子树和深度，叶子树保证是深度的2^max\_depth，结果是正确率提高了

**params** = {'boosting\_type': 'gbdt','objective': 'binary','metric': {'binary\_logloss', 'auc'},'num\_leaves': 64,'max\_depth': 6,'learning\_rate': 0.01,'feature\_fraction': 0.9,'bagging\_fraction': 0.95,'bagging\_freq': 5,'lambda\_l1': 1,'lambda\_l2': 0.001,'is\_unbalance': True}

**Accuracy**: 0.89868

# Round 3: 在同一个叶子树和深度，调整learning rate到0.05, 结果是正确率降低了

**params** = {'boosting\_type': 'gbdt','objective': 'binary','metric': {'binary\_logloss', 'auc'},'num\_leaves': 64,'max\_depth': 6,'learning\_rate': 0.05,'feature\_fraction': 0.9,'bagging\_fraction': 0.95,'bagging\_freq': 5,'lambda\_l1': 1,'lambda\_l2': 0.001,'is\_unbalance': True}

**Accuracy**: 0.89788

# Round 4: 基于前两轮的经验，降低learning rate到0.005，结果是正确率相对上一轮提高了，但不是目前最高

**params** = {'boosting\_type': 'gbdt','objective': 'binary','metric': {'binary\_logloss', 'auc'},'num\_leaves': 64,'max\_depth': 6,'learning\_rate': 0.005,'feature\_fraction': 0.9,'bagging\_fraction': 0.95,'bagging\_freq': 5,'lambda\_l1': 1,'lambda\_l2': 0.001,'is\_unbalance': True}

**Accuracy**: 0.89822

# Round 5: 如果保证learning rate是0.005，把叶子树和深度提高，总体感觉是叶子树越多，learning rate同时越低，带来的拟合度越好

**params** = {'boosting\_type': 'gbdt','objective': 'binary','metric': {'binary\_logloss', 'auc'},'num\_leaves': 256,'max\_depth': 8,'learning\_rate': 0.005,'feature\_fraction': 0.9,'bagging\_fraction': 0.95,'bagging\_freq': 5,'lambda\_l1': 1,'lambda\_l2': 0.001,'is\_unbalance': True}

**Accuracy**: 0.90102

# Round 6: 基于Round 5的结果，我将bagging\_freq调成1，结果拟合度提高了一点点，我的理解是，这样改成了每轮迭代都做bagging，对于每轮来讲都是公平的，虽然损失了效率

**params** = {'boosting\_type': 'gbdt','objective': 'binary','metric': {'binary\_logloss', 'auc'},'num\_leaves': 256,'max\_depth': 8,'learning\_rate': 0.005,'feature\_fraction': 0.9,'bagging\_fraction': 0.95,'bagging\_freq': 1,'lambda\_l1': 1,'lambda\_l2': 0.001,'is\_unbalance': True}

**Accuracy**: 0.90122

# Round 7: 基于Round 6的结果，去掉feature\_fraction，让它用上所有feature，损失了效率，但拟合度提高了一点点

**params** = {'boosting\_type': 'gbdt','objective': 'binary','metric': {'binary\_logloss', 'auc'},'num\_leaves': 256,'max\_depth': 8,'learning\_rate': 0.005,','bagging\_fraction': 0.95,'bagging\_freq': 1,'lambda\_l1': 1,'lambda\_l2': 0.001,'is\_unbalance': True}

**Accuracy**: 0.9021

# Round 8: 基于Round 7的结果，去掉bagging，泛化降低

**params** = {'boosting\_type': 'gbdt','objective': 'binary','metric': {'binary\_logloss', 'auc'},'num\_leaves': 256,'max\_depth': 8,'learning\_rate': 0.005,','bagging\_fraction': 0.95,'bagging\_freq': 1,'lambda\_l1': 1,'lambda\_l2': 0.001,'is\_unbalance': True}

**Accuracy**: 0.9017

# Round 9: 基于Round 8的结果，降低bagging比例到0.9，提高泛化

**params** = {'boosting\_type': 'gbdt','objective': 'binary','metric': {'binary\_logloss', 'auc'},'num\_leaves': 256,'max\_depth': 8,'learning\_rate': 0.005,','bagging\_fraction': 0.9,'bagging\_freq': 1,'lambda\_l1': 1,'lambda\_l2': 0.001,'is\_unbalance': True}

**Accuracy**: 0.902

# Round 10: 基于Round 7的结果，bagging比例调到0.98，降低泛化，但拟合度降低，有可能0.95是最优选

**params** = {'boosting\_type': 'gbdt','objective': 'binary','metric': {'binary\_logloss', 'auc'},'num\_leaves': 256,'max\_depth': 8,'learning\_rate': 0.005,','bagging\_fraction': 0.98,'bagging\_freq': 1,'lambda\_l1': 1,'lambda\_l2': 0.001,'is\_unbalance': True}

**Accuracy**: 0.90174

# Round 11: 基于前面调bagging的结果，试下bagging比例调到0.955，效果也没0.95的时候好

**params** = {'boosting\_type': 'gbdt','objective': 'binary','metric': {'binary\_logloss', 'auc'},'num\_leaves': 256,'max\_depth': 8,'learning\_rate': 0.005,','bagging\_fraction': 0.955,'bagging\_freq': 1,'lambda\_l1': 1,'lambda\_l2': 0.001,'is\_unbalance': True}

**Accuracy**: 0.90204

# Round 12: 基于前面调bagging的结果，试下bagging比例调到0.95，同时去掉lambda\_l2

**params** = {'boosting\_type': 'gbdt','objective': 'binary','metric': {'binary\_logloss', 'auc'},'num\_leaves': 256,'max\_depth': 8,'learning\_rate': 0.005,','bagging\_fraction': 0.955,'bagging\_freq': 1,'lambda\_l1': 1,'is\_unbalance': True}

**Accuracy**: 0.9022

# ****Round 13:** 基于Round 12，尝试调lambda\_l1=0.5**

**params** = {'boosting\_type': 'gbdt','objective': 'binary','metric': {'binary\_logloss', 'auc'},'num\_leaves': 256,'max\_depth': 8,'learning\_rate': 0.005,','bagging\_fraction': 0.955,'bagging\_freq': 1,'lambda\_l1': 0.5,'is\_unbalance': True}

**Accuracy**: 0.90178

# ****Round 14:** 基于Round 13，尝试调lambda\_l1=2**

**params** = {'boosting\_type': 'gbdt','objective': 'binary','metric': {'binary\_logloss', 'auc'},'num\_leaves': 256,'max\_depth': 8,'learning\_rate': 0.005,','bagging\_fraction': 0.955,'bagging\_freq': 1,'lambda\_l1': 2,'is\_unbalance': True}

**Accuracy**: 0.90178

# ****Round 15:** 尝试用goss**

**params** = {'boosting\_type': 'goss','objective': 'binary','metric': {'binary\_logloss', 'auc'},'num\_leaves': 256,'max\_depth': 8,'learning\_rate': 0.005,','lambda\_l1': 2,'is\_unbalance': True}

**Accuracy**: 0.9002

# ****Round 16: 尝试用goss，learning\_rate改0.0001****

**params** = {'boosting\_type': 'goss','objective': 'binary','metric': {'binary\_logloss', 'auc'},'num\_leaves': 256,'max\_depth': 8,'learning\_rate': 0.0001,','lambda\_l1': 2,'is\_unbalance': True}

**Accuracy**: 0.80452

# ****Round 17:** 尝试用goss，learning\_rate改0.01**

**params** = {'boosting\_type': 'goss','objective': 'binary','metric': {'binary\_logloss', 'auc'},'num\_leaves': 256,'max\_depth': 8,'learning\_rate': 0.0001,','lambda\_l1': 2,'is\_unbalance': True}

**Accuracy**: 0.9003

# ****Round 18: 尝试用gbdt，learning\_rate改0.01，选中所有feature，事实证明并不是有更多feature拟合就更好****

**params** = {'boosting\_type': 'goss','objective': 'binary','metric': {'binary\_logloss', 'auc'},'num\_leaves': 256,'max\_depth': 8,'learning\_rate': 0.01,','lambda\_l1': 2,'is\_unbalance': True}

**Accuracy**: 0.90232

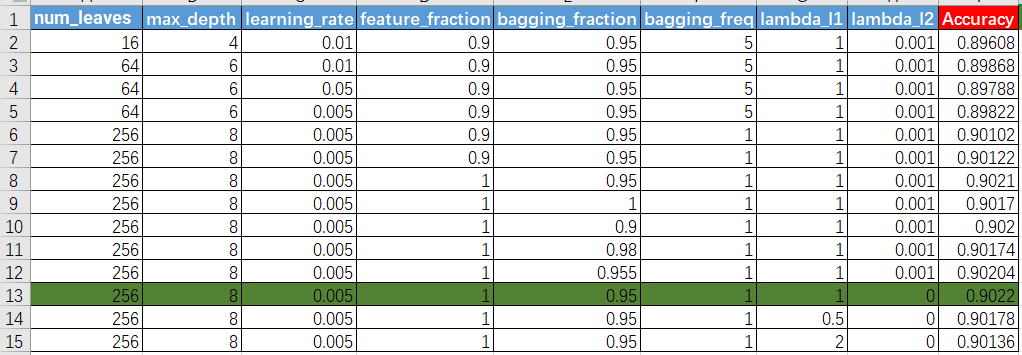
# ****Round 19:** 尝试用gbdt，learning\_rate改0.005，选中所有feature，事实证明并不是有更多feature拟合就更好**

**params** = {'boosting\_type': 'goss','objective': 'binary','metric': {'binary\_logloss', 'auc'},'num\_leaves': 256,'max\_depth': 8,'learning\_rate': 0.005,','lambda\_l1': 2,'is\_unbalance': True}

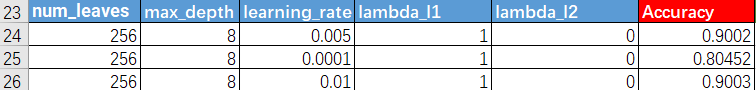
**Accuracy**: 0.90242

**小结：**

**用前面15个feature和gdbt**



**用前面15个feature和goss**



**用前面所有feature和gdbt**

