树模型与模型融合

一、lgb 模型的一些问题

问题一：GBDT 的回归模型中，单颗树是 cart 分类树还是 cart 回归树?

问题二：GBDT 的分类模型中，单颗树是 cart 分类树还是 cart 回归树?

问题三：GBDT 的分类模型中，单颗树的叶子节点值是如何确定的?

问题四：GBDT 的 3 分类模型中，每一次迭代会生成有几棵树?

问题五：GBDT 和 Xgb 的关系是什么?

问题六：Lgb 和 Xgb 的关系是什么?

问题七：Xgb , Lgb, GBDT 三者最终的模型形式是否一致?

二、树模型的局限性

单树模型与 K 近邻

单树模型可以从两个不同的视角来观察。第一种是递归生成子树的视角，核心思想是不断的划分训练数据，使得划分后的集合变得尽量的「纯」。第二种是空间划分的视角，核心思想是对训练数据所在的特征空间进行划分，每一个划分区域对应一个预测值，即一个分片常数函数：

$$f(\mathbf{x}^i) = \sum_{m=1}^{M} c_m I(\mathbf{x}^i \in R_m)$$

当定义好「损失」之后，利用「二叉树」的特性，递归的对特征空间进行划分，从

而得到最终的空间划分。在这种视角下，「树」更多的是体现在对上述待优化式子的一种学习方式，就如同梯度下降法一般。

这两种视角在大部分情况下是等价的，比如当使用 **平方误差** 作为损失函数时，对应的度量集合「纯度」的方法就是 **方差**。

现在我们从第二个视角出发，思考单树模型和 `K近邻算法` 的区别与联系。

`K近邻算法` 和单树模型从最终形式上看都可以统一成分片常数函数的形式，且随着训练数据的增加，两者的分片区域是单调递增的。即计算量和训练数据样本数正相关，所以单树模型和 `K近邻算法` 一样都属于非参数模型。

然而 `K近邻算法` 和单树模型在空间划分上有一定的区别。一方面是关于子区域的边界，前者是不规则的曲线，后者则是 **平行** 于「特征」方向的一系列直线组成。另一方面是关于空间划分的过程， `K近邻算法` 在进行空间划分时并未使用到训练数据的标签信息，只利用了训练数据的特征，即通过定义的样本间的「距离」来对特征空间进行划分。单树模型则不然，单树模型的空间划分充分利用了训练数据的标签信

息。最后则是关于空间划分和特征量纲的关系，`K近邻算法`的空间划分依赖于样本间的「距离」，很明显，距离是受到样本特征量纲的影响的，换句话说，当我们对样本的特征做不同的放缩变化时，会影响到空间划分的结果。然后单树模型则不受样本特征的量纲影响，因为在进行节点划分时，样本的划分只和选择的特征的相对大小有关，而和绝对大小无关，所以在使用单树模型时，没必要对特征进行标准化或是归一化的操作。

多树模型

$$f(\mathbf{x}^i) = \sum_{t=1}^{T}\sum_{m=1}^{M} c_{tm} I(\mathbf{x}^i \in R_{tm})$$

无论是单树模型还是多树模型，他们只适用于「区间内」预测，对于区间外的样本标签，无法进行良好的预测。

举个例子，如果利用树模型直接对一段连续的时间序列直接预测，那么是难以直接学习到后面的变化趋势的，因为后面的时序变化可能超过了历史训练集的最大值或

最小值，属于「区间外」预测。

如何解决这个问题?

将直接预测改造为相对预测，即 lgb 拟合的不是直接的标签，而是一个相对标签。

比如对于前面的例子，可以如下进行相对标签的构造：

- 利用过去一段时间的平均值和未来的标签的差生成相对标签。
- 利用另一个趋势模型的预测值和未来的标签的差生成相对标签。

回到我们这次的题目，这次题目属于 lgb 直接预测，如果需要间接预测，请问如何进行改造?

直接预测：lgb 预测拟合未来 10 分钟的波动率。

三、参数搜索

参数搜索又叫做超调整搜索。为什么要进行超参数的搜索？因为我们无法通过模型来学习这些参数。换句话说，这些参数无法通过梯度下降等技巧进行学习。

在算力无穷的情况下，可以通过验证集的反馈结果搜索得到最优的参数。超参数的搜索大致可以分为下面三类。

- 网格搜索
- 随机搜索
- 贝叶斯搜索
- 强化学习搜索

前两者是假设在不同时刻搜索的点之间是没有任何联系的，即历史的搜索路径不会指导我们以后的搜索路径。后两者则是假设不同时间搜索的点之间有联系，即历史的搜索路径会指导我们以后的搜索路径。

比如有的贝叶斯搜索会假设搜索的点和反馈结果满足高斯过程的假设。从而利用历

史的搜索路径指导我们下一次的搜索。再比如有的强化学习，则是将搜索路径看作为序列决策过程，利用设计的奖励函数指导我们后续的搜索。

```
pip install bayesian-optimization
```

```
from bayes_opt import BayesianOptimization
import warnings
import numpy as np
import pandas as pd
import lightgbm as lgb
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error
warnings.simplefilter(action='ignore', category=FutureWarning)
```

```
def LGB_CV(
          max_depth,
          num_leaves,
```

```
            min_data_in_leaf,
            feature_fraction,
            bagging_fraction,
            lambda_l1
           ):

    folds = KFold(n_splits=5, shuffle=True, random_state=15)
    oof = np.zeros(train.shape[0])

    for fold_, (trn_idx, val_idx) in enumerate(folds.split(trai
n.values, target.values)):
        print("fold n°{}".format(fold_))
        trn_data = lgb.Dataset(train.iloc[trn_idx][features],
                               label=target.iloc[trn_idx],
                               categorical_feature=categorical_
feats)
        val_data = lgb.Dataset(train.iloc[val_idx][features],
                               label=target.iloc[val_idx],
                               categorical_feature=categorical_
```

```
feats)

        param = {
            'num_leaves': int(num_leaves),
            'min_data_in_leaf': int(min_data_in_leaf),
            'objective':'regression',
            'max_depth': int(max_depth),
            'learning_rate': 0.01,
            "boosting": "gbdt",
            "feature_fraction": feature_fraction,
            "bagging_freq": 1,
            "bagging_fraction": bagging_fraction ,
            "bagging_seed": 11,
            "metric": 'rmse',
            "lambda_l1": lambda_l1,
            "verbosity": -1
        }

        clf = lgb.train(param,
```

```
                          trn_data,
                          10000,
                          valid_sets = [trn_data, val_data],
                          verbose_eval=500,
                          early_stopping_rounds = 200)

        oof[val_idx] = clf.predict(train.iloc[val_idx][features
],
                                   num_iteration=clf.best_itera
tion)

        del clf, trn_idx, val_idx
        gc.collect()

    return -mean_squared_error(oof, target)**0.5
```

```
LGB_BO = BayesianOptimization(LGB_CV, {
    'max_depth': (4, 10),
    'num_leaves': (5, 130),
```

```
    'min_data_in_leaf': (10, 150),
    'feature_fraction': (0.7, 1.0),
    'bagging_fraction': (0.7, 1.0),
    'lambda_l1': (0, 6)
    })

```

```
with warnings.catch_warnings():
    warnings.filterwarnings('ignore')
    LGB_BO.maximize(init_points=2, n_iter=20, acq='ei', xi=0.0)
```

n_iter: How many steps of bayesian optimization you want to perform. The more steps the more likely to find a good maximum you are.

init_points: How many steps of random exploration you want to perform. Random exploration can help by diversifying the exploration space.

exploration strategy: UCB (Upper Confidence Bound), EI (Expected

Improvement)) are used to determine the next point that should be explored (see the gif below).

```
print('Final Results')
print('Maximum  value: %f' % LGB_BO.max['target'])
print('Best  parameters: ', LGB_BO.max['params'])
```

BayesianOptimization 的 github 链接

四、模型的融合

打比赛的模型融合目的很简单很单纯。就是得到更高的分数，不用关心执行的效率，甚至过拟合。

本次比赛是针对回归的预测，后面讨论的主要是针对回归预测的结果融合。融合可

以大致分为两类，一类是不引入外部信息的结果融合，一类是引入外部信息的结果融合。

不引入外部信息的结果融合

对样本 i 的不同预测结果 y_{ij} ，不引入外部信息，这里的外部信息可能是验证集的评估反馈结果，也可能是线上测试集的评估得分。

1. 算术平均融合

$$\hat{y}_i = \frac{\sum_{j=1}^{m} y_{ij}}{m}$$

2. 几何平均融合

$$\hat{y}_i = \sqrt[m]{\prod_{j=1}^{m} y_{ij}}$$

3. 平方平均融合

$$\hat{y}_i = \sqrt{\frac{\sum_{j=1}^{m} y_{ij}^2}{m}}$$

4. 幂平均融合

$$\hat{y}_i = \sqrt[t]{\frac{\sum_{j=1}^{m} y_{ij}^t}{m}}$$

$$最小值 \leq 几何平均 \leq 算数平均 \leq 平方平均 \leq 幂平均 \leq 最大值$$

注意这里要求幂平均中的 $t \geq 2$ 。

引入外部信息的结果融合

对样本 i 的不同预测结果 y_{ij} ，引入外部信息，这里的外部信息既可以是验证集的评估反馈结果，也可以是线上测试集的评估得分。前者是为了模型鲁棒和泛化，后者

是为了对抗线上样本相对于线下样本的迁移。

$$\hat{y_i} = \sum_{j=1}^{m} \alpha_j y_{ij}$$

$$\sum_{j=1}^{m} \alpha_j = 1$$

如何获得 α_j 呢？一种方式是通过人工定义的规则获取，线下/线上分数高的权重就高，线下/线上成绩相对低的权重就低，最终将加权的结果作为最终的结果(需保证所有的结果的权重之和为1)。另一种方式是利用线下/线上的得分反馈不断调整 α_j，容易看到，此时又成为了参数搜索的问题。