

思考一：分箱处理异常数据

在之前的讨论中我们发现，数据清洗带来的效果不一定好。那么现在就来探究其中的原因。

先说一下我们的思路：

- 我们将距离分箱，探究不同区间距离内 LightGBM 算法的性能
- 如果性能差距不大，说明这个距离区间数据分布接近
- 如果性能差距较大，说明这个区间数据有很多异常值

好，其实也很简单，代码如下：

```
def xgb_train(x,y,valid=testXsplit,valid_y=testYsplit):
    xgb_regressor = lgb.LGBMRegressor(
        n_estimators=500,
        learning_rate=0.1,
        max_depth=25,
        subsample=0.9,
        random_state=42,
        colsample_bytree=0.8,
        eta=0.05
    )
    xgb_regressor.fit(x[features], y)
    print("Score in Train
%.10f"%np.sqrt(MSE(xgb_regressor.predict(x[features]),y)))
    print("Score in Val
%.10f"%np.sqrt(MSE(xgb_regressor.predict(valid[features]),valid_y)))
    return xgb_regressor
```

我们固定了一个回归器，传入参数是训练数据和验证数据，这些数据都在之前的分割中做好了。

接着我们确定距离区间，并传入固定回归器查看性能：

```
# 随着 k 值增大，验证集和训练集的表现都变得更好。所以没问题，k就表示异常
k=0.2
error,error_val=trainXsplit[trainXsplit['distance']
<k],testXsplit[testXsplit['distance']<k]
# 异常值问题很大
E=xgb_train(error,error.trip_duration,error_val,error_val.trip_duration)

...
Score in Train 0.8628670500
```

```
Score in Val 1.5765071968
'''
```

```
k1,k2=0.2,1
error,error_val=trainXsplit[trainXsplit['distance'].between(k1,k2)],testXsplit[
testXsplit['distance'].between(k1,k2)]
# 异常值问题很大
E=xgb_train(error,error.trip_duration,error_val,error_val.trip_duration)

'''

Score in Train 0.4521522822
Score in Val 0.4926791280
'''
```

```
k1,k2=1,5
error,error_val=trainXsplit[trainXsplit['distance'].between(k1,k2)],testXsplit[
testXsplit['distance'].between(k1,k2)]
# 异常值问题很大
E=xgb_train(error,error.trip_duration,error_val,error_val.trip_duration)

'''

Score in Train 0.3216578993
Score in Val 0.3285489193
'''
```

```
k1,k2=5,30
error,error_val=trainXsplit[trainXsplit['distance'].between(k1,k2)],testXsplit[
testXsplit['distance'].between(k1,k2)]
# 异常值问题很大
E=xgb_train(error,error.trip_duration,error_val,error_val.trip_duration)

'''

Score in Train 0.2548541791
Score in Val 0.2644395541
'''
```

```
k1,k2=30,1000000
error,error_val=trainXsplit[trainXsplit['distance'].between(k1,k2)],testXsplit[
testXsplit['distance'].between(k1,k2)]
# 异常值问题很大
E=xgb_train(error,error.trip_duration,error_val,error_val.trip_duration)


'''

Score in Train 0.0492034340
Score in Val 0.3990647060
'''
```


结果已然不言而喻。区间[0, 0.2]问题十分夸张，区间[5, 30]表现最为良好。

那么，下一步该怎么做？其实答案也快浮出水面了：分箱训练。

这是全部用[5,30]这个回归器的得分。


image-20240823152901638


这是全部用平均值填充的得分。

image-20240823153056312

看起来还不如平均值填充？

我们在0-0.2区间采用原始数据0-0.2区间的平均值填充、以及原始数据总平均值填充的结果：

image-20240823161148991

image-20240823161856625

这个东西过于奇葩了，我们用1.5IQR处理一下，再训练一个模型看看？

```
# 请注意一件事，无论何时都会有duration大于8000s的，很奇怪，关注这些数据
trainXsplit[(trainXsplit.trip_duration>9) &(trainXsplit.speed<10)]
['trip_duration'].count()
```

```
# 剔除异常值后进行训练
# 获得一个差不多的模型
# 这个模型的值作为输出值
# 训练一个神经网络模型，预测哪个有可能出错（玄学）
# 方案一：原始输出
# 方案二：预测为异常值的结果用异常值的平均输出
# 方案三：异常值平均*权重+原始输出*（1-权重）
```

我们弄了三个区间比较稳定(相对)的结果：

```
def get_model(k1,k2):
    x=x_var[x_var['distance'].between(k1,k2)]
    lgb_params = {
        #'metric' : 'rmse',
        'learning_rate': 0.1,
        'max_depth': 25,
        'num_leaves': 1000,
        'objective': 'regression',
        'feature_fraction': 0.9,
```

```

'bagging_fraction': 0.5,
'max_bin': 1000
, 'verbosity': -1
}

lgb_df = lgb.Dataset(x[features], x.trip_duration)
lgb_model = lgb.train(lgb_params, lgb_df, num_boost_round=1500)
return lgb_model

```

可以看一下他们几个在各自区间上的表现：


第一个模型 `lgb_2`，处理区间是`[1, 1000000]`，结果如下：

```

# 整个数据集
'''
Score in Train 0.3459272904
Score in Val 0.3604114883
'''

# 对应区间
'''
Score in Train 0.1271533114
Score in Val 0.1535727598
'''

```

image-20240823181109394

还行吧，只是很多异常值没有学到所以导致结果很差。我觉得他在他的区间应该是没什么问题。

第二个模型 `lgb_05`，处理区间是`[0.5, 1]`

```

# 整个数据集
'''
Score in Train 0.8714976487
Score in Val 0.8905686372
'''

# 对应区间
'''
Score in Train 0.0151601645
Score in Val 0.0177238618
'''

```

第三个模型 `lgb_03`，处理区间是`[0.3, 0.5]`

```
# 整个数据集
'''
Score in Train 1.2949600411
Score in Val 1.3132815278
'''

# 对应区间
'''
Score in Train 0.0062081007
Score in Val 0.0073961800
'''
```

然后重点关注对象就是[0,0.3]这个区间。

```
# 剔除异常值后进行训练
# 获得一个差不多的模型
# 这个模型的值作为输出值
# 训练一个神经网络模型，预测哪个有可能出错（玄学）
# 方案一：原始输出
# 方案二：预测为异常值的结果用异常值的平均输出
# 方案三：异常值平均*权重+原始输出*（1-权重）
```

目前来说，对于整个数据集直接进行训练的得分最高，可以达到0.37014，然后我们对[0,0.2], [0.2,1],[1,10000000]做区分。

如果是混合预测，得分情况为：

0.37615

如果是直接用[1,10000000]的做预测，得分情况为：

0.39730

如果用lgb_T替换掉lgb_1的[0,1]部分，得分情况为：

0.37905

如果用lgb_T和lgb_1平均投票，得分情况为：

0.37849

如果用lgb_T和替换后的lgb_1平均投票，得分情况为：

0.37048

诶，先别急，还记得我们这里lgb_1在前俩区间预测都不好吗？如果稍作修改呢？

0.37210 可见，lgb_1在原始数据上过拟合了，但实际上跟lgb_T差别不大。

如果他俩混合？

0.36886 还不错诶，看看这个排多少？ 59/1206

如果三个混合？

0.36866

看起来，三个混合的效果还真不错，我们接下来替换掉lgb_4，也就是表现最不好的区间：

0.37433

差别也不太大。

那如果再混合呢？

0.36897

好，至此结束。可见，最好的结果仍然是三者混合的结果。

混合方式	得分
64	0.36781
73	0.36793
82	0.36840
46	0.36942
55	0.36866

可以看到，基本上差的不多，最优的应该是这个73。

至此，理论上可以告一阶段了，我们再训练一个XGBoost模型出来进行融合，这样就有三个模型了，下一阶段将对lgb_4调优，一定要找到比较好的优化结果。

四模融合

混合方式	得分
1111	0.36859
4321	0.36760
615151	0.36748
5221	0.36733
521515	0.36753
325252	0.36817
5311	0.36753
4411	0.36741

或者我们用深度学习方式融合，其实我们可以在训练集上评估我们的策略：

```
k=np.matmul(ToTrain.iloc[:, :4].values, np.array([0.5, 0.2, 0.2, 0.1]).T)
np.sqrt(MSE(np.log(k), tar))
# 5221: 2548
# 1111: 2722
# 4411 2435
# 5311 2453
# 325252 2672

# 可见与最后的得分确实有点关系
```

但也不全是，我们用CNN处理后，在训练集上达到了0.22，但是在验证集上分数最低。有可能是我们的数据量不够多。

至于复杂网络，我们重新测试了下UNet-Transformer1D，这个网络一般只能训练十多个Epoch，一般就到顶点了，再训练结果只会越来越差。

简单CNN的表现是0.40，这个结果其实很差了。