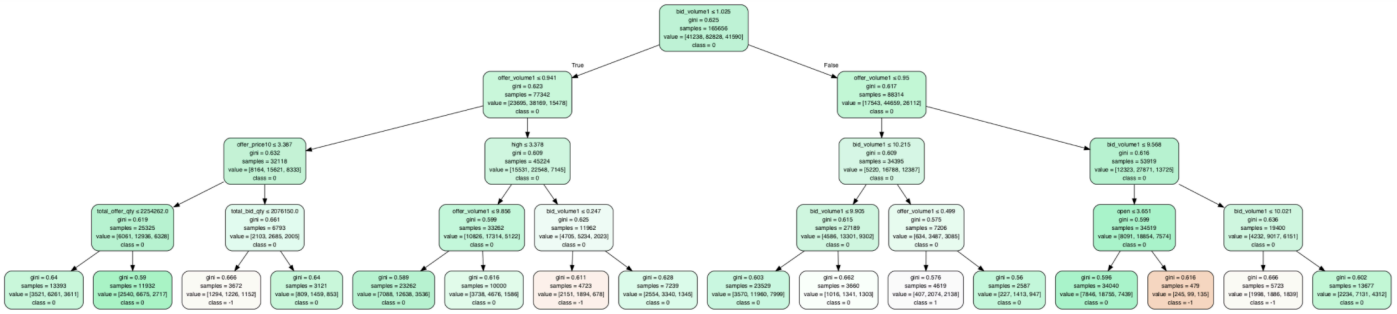
1简单决策树预测结果

1.1 决策树初步

单个决策树可以清晰看出树形结构，评估使用的特征以及分割的方式，可以为之后的集成模型做参考。下面首先使用单个决策树进行分析，首先考察分类决策树，使用的数据是2016.01-2016.06。

首先考虑到数据中标签具有很大的不平衡性。这样会导致分类器倾向于特定类别，导致看起来的准确性很高，其实没有效果。所以我采取重新采样的方法。平均来看，为+1和-1的样本量接近，主要是为0的样本很多，在这一时间区间内为(-1:41238, 0:196730, 1:41590)。因此我用重新抽样的方法在196730个为0的样本中抽取了大约80000个，进行训练得到下面的模型：



从上面图中看出，当树的深度设置为4时，只有当bid_valume1>10.215, offer_volume1<0.0499时样本才被分类成为了上涨+1，其他情况下主要是0和-1（下面的表格展示了bid_volume1和offer_volume1的分布）。 随着深度的增加，分类为-1和1的次数会随着增多。

	bid_volume1	offer_volume1
count	279558	279558
mean	4.935679	5.366560
std	13.957018	20.485556
min	0.000100	0.000100
25%	0.370000	0.470000
50%	1.230000	1.442600
75%	5.090000	5.440300
max	909.946200	1277.651600

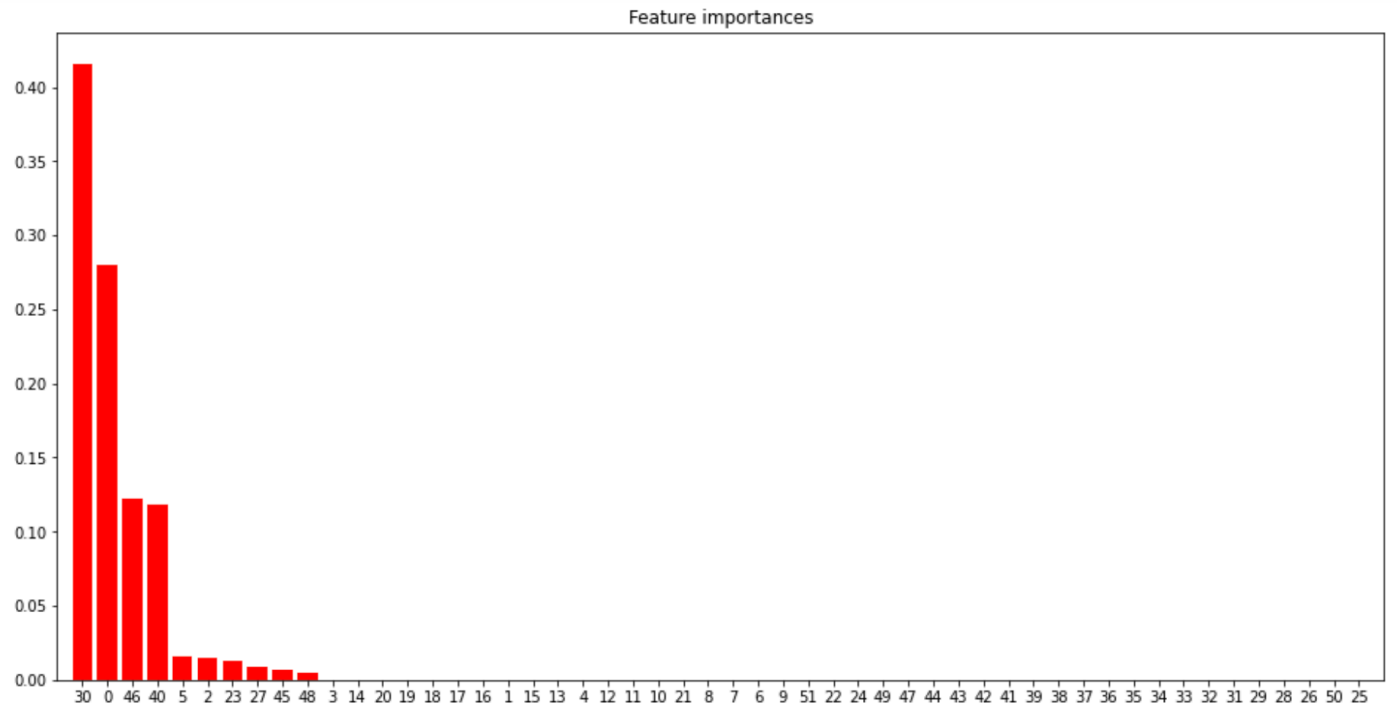
上面模型的样本内 Accuracy Score 为0.504377，AUC Score 为 0.602459。这是做了重采样的结果。为了说明标签不平衡的影响，我比较了不进行重采样的结果：未进行重采样得到样本内 Accuracy Score 为0.704094，AUC Score 为0.597004。可以看出模型通过大量预测0，使得准确性大大增加。这提醒我们后面评价模型需要查看每一个类别上的预测准确性。同时，我们可以通过参数'class_weight'对不平衡进行调整，这是下一部分网格搜索的内容。

```
accuracy_score Score (Train): 0.504377
AUC Score (Train): 0.602459
```

```
accuracy_score Score (Train): 0.704094
AUC Score (Train): 0.597004
```

考虑特征的重要性，发现只有少数特征被使用，这也是决策树的问题之一。因此后续会考虑使用集成算法，这样不同特征都有可能被用上。同时可以考虑使用特征预处理，进行特征的标准化处理，这样有利于决策树的相关指标计算。

```
Feature ranking:
1. feature 1 bid_volume1 (0.416011)
2. feature 2 offer_price1 (0.279640)
3. feature 3 total_volume_trade (0.121885)
4. feature 4 preclose (0.118189)
5. feature 5 offer_price6 (0.015502)
6. feature 6 offer_price3 (0.015148)
7. feature 7 bid_price4 (0.013358)
8. feature 8 bid_price8 (0.009193)
9. feature 9 num_trades (0.006621)
10. feature 10 total_bid_qty (0.004453)
```



1.2 特征预处理

上面提到特征处理可以优化决策树的优化过程，同时为了后面使用回归模型，需要对模型feature进行预处理。下面进行简单的数据处理，并且比较模型是否得到优化，同时进行分类模型和回归模型的分析。

数据的处理过程包括极端值去除，正态标准化，0-1标准化等，此外使用SparsePCA进行了初步的特征选择，放弃了相关性较高且没有信息增益的特征。

```
robustscaler = RobustScaler().fit(x_train)
x_train = robustscaler.transform(x_train)
standardscaler = StandardScaler().fit(x_train)
x_train = standardscaler.transform(x_train)
minmaxscaler = MinMaxScaler().fit(x_train)
x_train = minmaxscaler.transform(x_train)
pca = SparsePCA(n_components=7, random_state=15).fit(x_train)
x_train = pca.transform(x_train)
```

1)未采取特征预处理的分类模型

```
accuracy_score Score (Train): 0.555073
AUC Score (Train): 0.546976
[[  5791  28030   7417]
 [ 18955 136999  40776]
 [  5157  24048  12385]]
```

2) 未采取特征预处理的回归模型

```
accuracy_score Score (Train): 0.601936
[[   571  33436   7231]
 [   591 155449  40690]
 [   243  29091  12256]]
```

3) 采取了特征预处理的分类模型

```
accuracy_score Score (Train): 0.478462
AUC Score (Train): 0.537372
[[ 11940  21269   8029]
 [ 46290 113026  37414]
 [ 11348  21450   8792]]
```

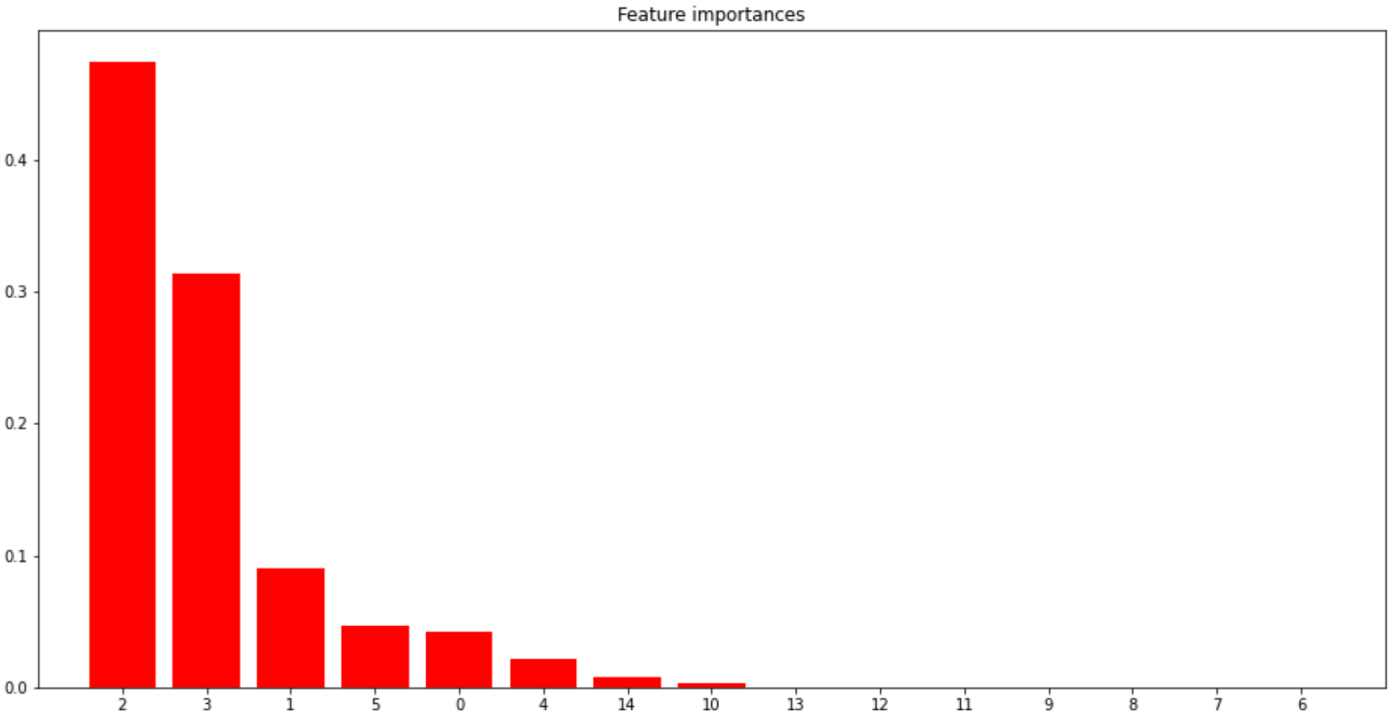
4) 采取了特征预处理的回归模型

```
accuracy_score Score (Train): 0.694461
[[  1334  39380    524]
 [  2898 192223   1609]
 [  1009  39996    585]]
```

上面的结果显示进行特征处理可以优化分类决策树的表现，从混淆矩阵的第一行和最后一行可以看到，对于真实情况为-1和+1的样本，预测结果有了一定提升。-1的True Positive 从0.163提高到0.407，但同时-1的True negative从0.424降低到了0.268。但是对于回归模型并没有显著提升。因此下面考虑了一种使用**决策树降维**的方法：首先使用第一层决策树，并且选择排名前5的feature，去除这些选中的feature，再次使用第二层决策树，选择排名前5的feature，以此类推进行组合，关键在于能够使用更多的feature信息。

经过三次筛选，选择了如下的15个feature，并且将这15个feature再次进行组合得到如下的排序：

```
Feature ranking:
1. feature 2 total_volume_trade (0.474469)
2. feature 3 preclose (0.313627)
3. feature 1 offer_price1 (0.090738)
4. feature 5 bid_price4 (0.047210)
5. feature 0 bid_volume1 (0.042158)
6. feature 4 offer_price6 (0.021156)
7. feature 14 offer_price9 (0.007260)
8. feature 10 offer_volume3 (0.003382)
9. feature 13 bid_volume4 (0.000000)
10. feature 12 offer_price5 (0.000000)
11. feature 11 bid_volume10 (0.000000)
12. feature 9 offer_volume9 (0.000000)
13. feature 8 offer_price6 (0.000000)
14. feature 7 bid_volume3 (0.000000)
15. feature 6 offer_price1 (0.000000)
```




```
accuracy_score Score (Train): 0.704276
AUC Score (Train): 0.540176
[[   586  40645      7]
 [   435 196288      7]
 [   258  41320     12]]
```

从上面可以看出，模型基本上输出了很多0，只有少数情况下输出了-1和1，因此模型表现也不是很好。上面的结论表明，使用单个决策树很难完成任务。因此后面我们考虑集成模型如random forest和lightgbm。

1.3 参数优化

以上的分析没有采用参数优化，使用的多是默认参数，只控制了max_depth=4。接下来使用网格搜索进行参数优化，采用的优化目标为'roc_auc_ovr_weighted'，比用'accuracy'更好的评价非平衡样本。

```
{'max_depth': 4} 0.3923693803981838
{'min_samples_split': 50} 0.3923693803981838
{'min_samples_leaf': 110} 0.3934609352759666
{'class_weight': 'balanced'} 0.4385350164593582
{'random_state': 0} 0.4385350164593582
```

初始化参数为上面的较优组合，进行第二次搜索，可以发现基本上被参数max_depth主导了，其他参数的改变基本没有影响模型评价的值。

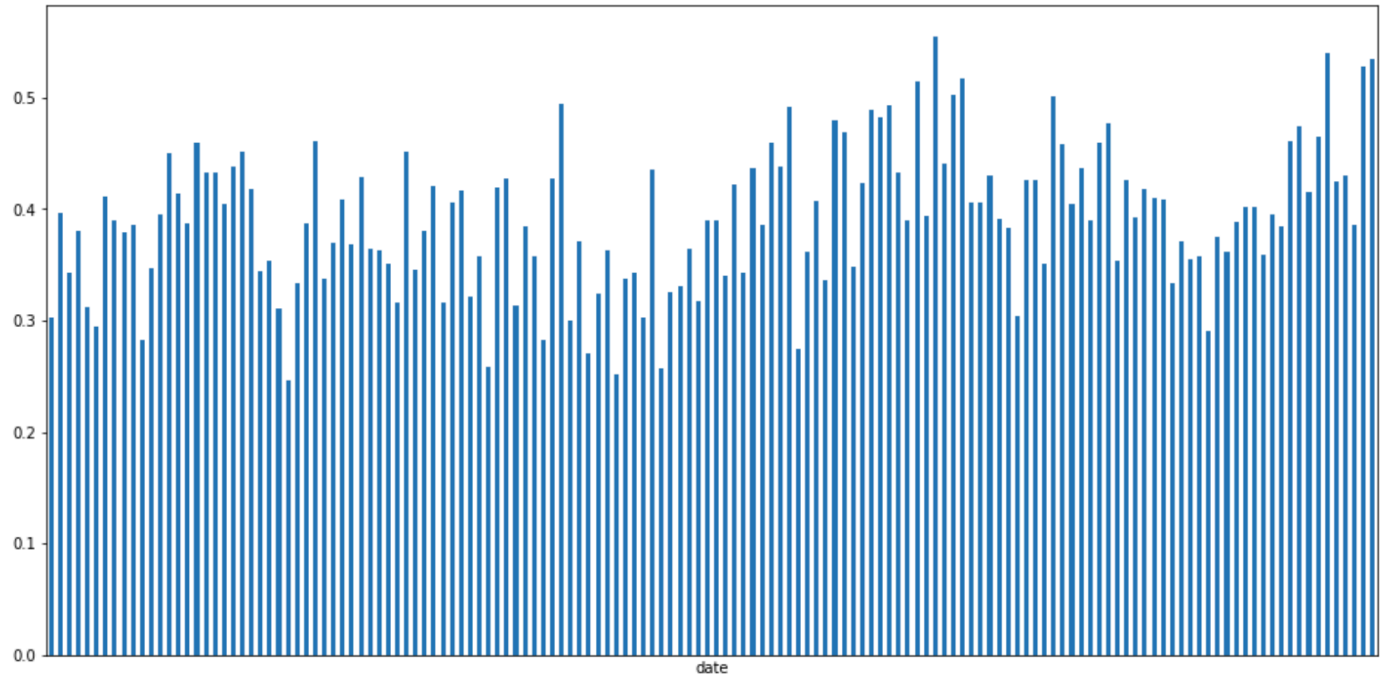
```
{'max_depth': 5} 0.4470726250827394
{'min_samples_split': 2} 0.4471562127675544
{'min_samples_leaf': 19} 0.44732252568931735
{'class_weight': 'balanced'} 0.44732252568931735
{'random_state': 0} 0.44732252568931735
```

因此我再次使用了几个不同的max_depth进行搜索，得到如下的结果：

```
'params': [{'max_depth': 5}, {'max_depth': 10}, {'max_depth': 20}, {'max_depth': 30}, {'max_depth': 100}, {'max_depth': 200}],
'split0_test_score': ([0.47057082, 0.46827349, 0.45203304, 0.45090343, 0.45091593, 0.45091593]),
'split1_test_score': ([0.40963037, 0.38840689, 0.36947176, 0.37140439, 0.37143221,0.37143221]), 'split2_test_score': ([0.46176639, 0.42938902, 0.43317736, 0.43932789, 0.43972534,0.43972534]), 'mean_test_score': ([0.44732253, 0.4286898 , 0.41822739, 0.42054524, 0.42069116, 0.42069116]), 'std_test_score': ([0.02689366, 0.03260915, 0.03532438, 0.0350677 , 0.03512967,0.03512967]), 'rank_test_score': ([1, 2, 6, 5, 3, 3], dtype=int32)}
{'max_depth': 5} 0.44732252568931735
```

上面的结果显示，当max_depth较大（200）的时候，反而比20表现更好。但是差异并不明显。只有当差不多是10以内表现较好。因此后续考虑将值设置为3-10。

这里先尝试一下简单决策树在测试集上的效果。训练区间为20170101-20170601， 预测区间为20170601-20171230。对于输入feature进行了去极值，标准化的处理，得到如下结果：图中展示了预测区间内从20170601-20171230每日的预测准确度，可以看出单个决策树准确度较低，基本上无法达到50%的准确性，说明单决策树模型难以处理众多feature。



2 随机森林预测结果

以上决策树的分析为使用随机森林等集成模型提供了参考。随机森林通过使用随机抽样建立相互独立的决策树，最后通过bagging进行投票选择最优的预测结果，很好的最大化利用了不同特征的信息。为了利用随机森林进行预测，首先采用网格搜索的方法确定随机森林相关参数。这里运用20160104-20170101的数据进行训练，预测目标为30s之后的价格方向。

```
第一步: param_opt = {'n_estimators': range(60, 100, 10)}
{'n_estimators': 80}
0.43743378402975813
第二步: param_opt = {'max_depth': range(3, 12, 1)}
 {'max_depth': 3}
0.4569473679297679
第三步: param_opt = {'min_samples_split': range(90,200,20)}
 {'min_samples_split': 90}
0.4569473679297679
```

最终的模型参数如下:

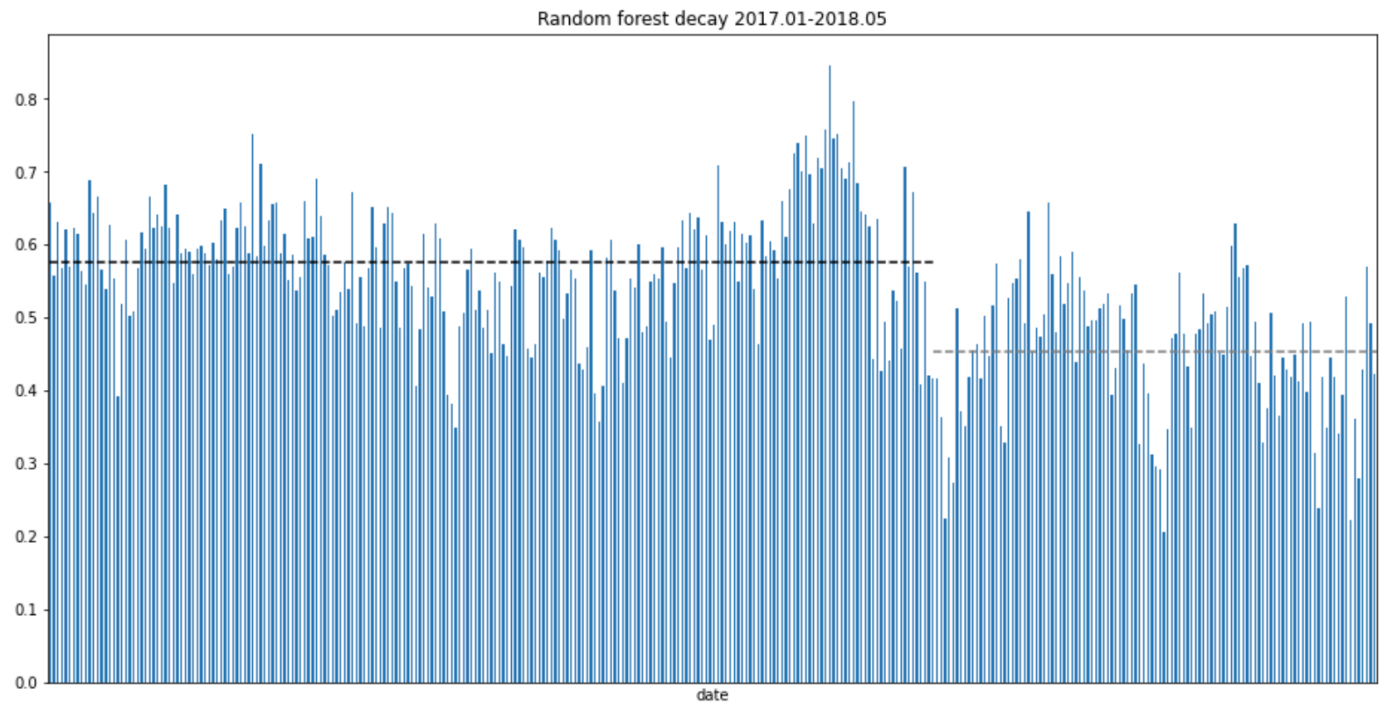
```
model final params
{'n_estimators': 80, 'oob_score': True, 'max_features': 'auto', 'max_depth': 3,
'min_samples_split': 90, 'min_samples_leaf': 10, 'min_weight_fraction_leaf': 0,
'max_leaf_nodes': None, 'min_impurity_decrease': 0, 'n_jobs': None, 'random_state':
0}
0.5244180384359886
Accuracy Score (Train): 0.524365
AUC Score (Train): 0.616040
```

下面我使用每200天的数据进行训练，将模型用于下一天的全天收益方向预测。模型一直滚动训练，初步来看模型的准确度超过了50%，在55%-65%附近，表现优于决策树模型。但是2017年下半年的某些天数准确度有所降低，仅有40%附近。因此考虑模型具有一定的decay。

```
date
2017-01-13  2065  0.553511  0.446489
             obs     ratio       mse
date
2017-01-16  2368  0.392736  0.607264
             obs    ratio      mse
date
2017-01-17  2015  0.51861  0.48139
             obs     ratio       mse
date
2017-01-18  1989  0.606838  0.393162
             obs     ratio       mse
date
2017-01-19  1929  0.502333  0.497667
             obs     ratio       mse
date
2017-01-20  2425  0.507629  0.492371
             obs     ratio       mse
date
2017-01-23  1933  0.567512  0.432488
             obs     ratio       mse
date
2017-01-24  1795  0.616713  0.383287
```

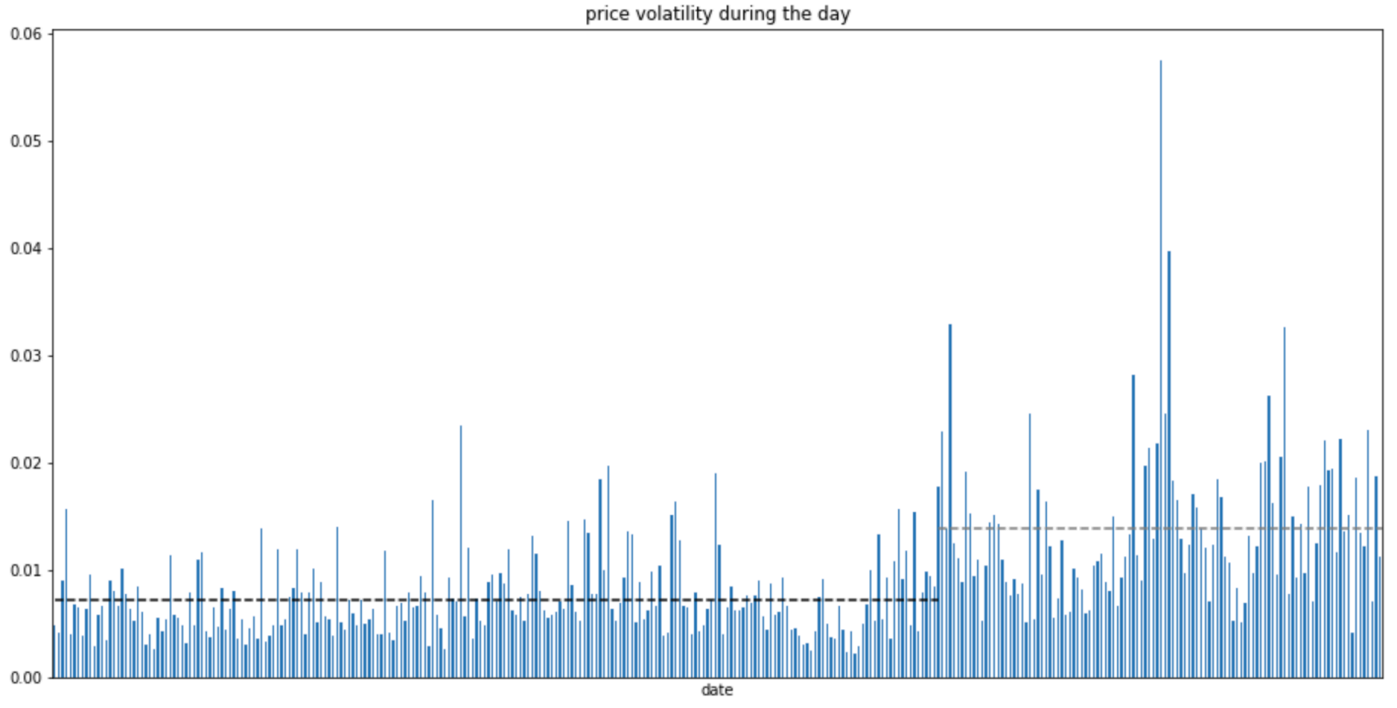
```
2017-11-28  3576  0.512304  0.487696
             obs     ratio       mse
date
2017-11-29  3811  0.371818  0.628182
             obs     ratio       mse
date
2017-11-30  3952  0.350202  0.649798
             obs     ratio       mse
date
2017-12-01  3712  0.418642  0.581358
             obs     ratio       mse
date
2017-12-04  3659  0.454496  0.545504
             obs     ratio       mse
date
2017-12-05  3606  0.463949  0.580976
             obs     ratio       mse
date
2017-12-06  3686  0.415898  0.586544
             obs     ratio       mse
date
2017-12-07  3595  0.502921  0.506259
             obs     ratio       mse
```

(1) 模型随时间decay的情况：下图展示了在预测区间从20170101到20180530每日预测精度情况，两条横线为对应时间区间的均值。由于模型的参数是利用20160101-20170101的数据进行调整的，因此距离参数调整较近的2017年初，模型表现较好，但是到了2017年末，模型表现变差，说明原来的参数已经不适合现在的市场环境。



影响预测准确度的因素：**价格区间波动性**

为了说明市场环境很可能发生变化，导致模型参数不再适用，我们绘制了价格变化图和日内价格波动率变化图。从图中可以看出，日内价格波动率与模型表现呈现负相关性。当日内价格波动较为剧烈的时候，也是模型预测效果较差的时候。当日内价格波动较低，模型表现较好。



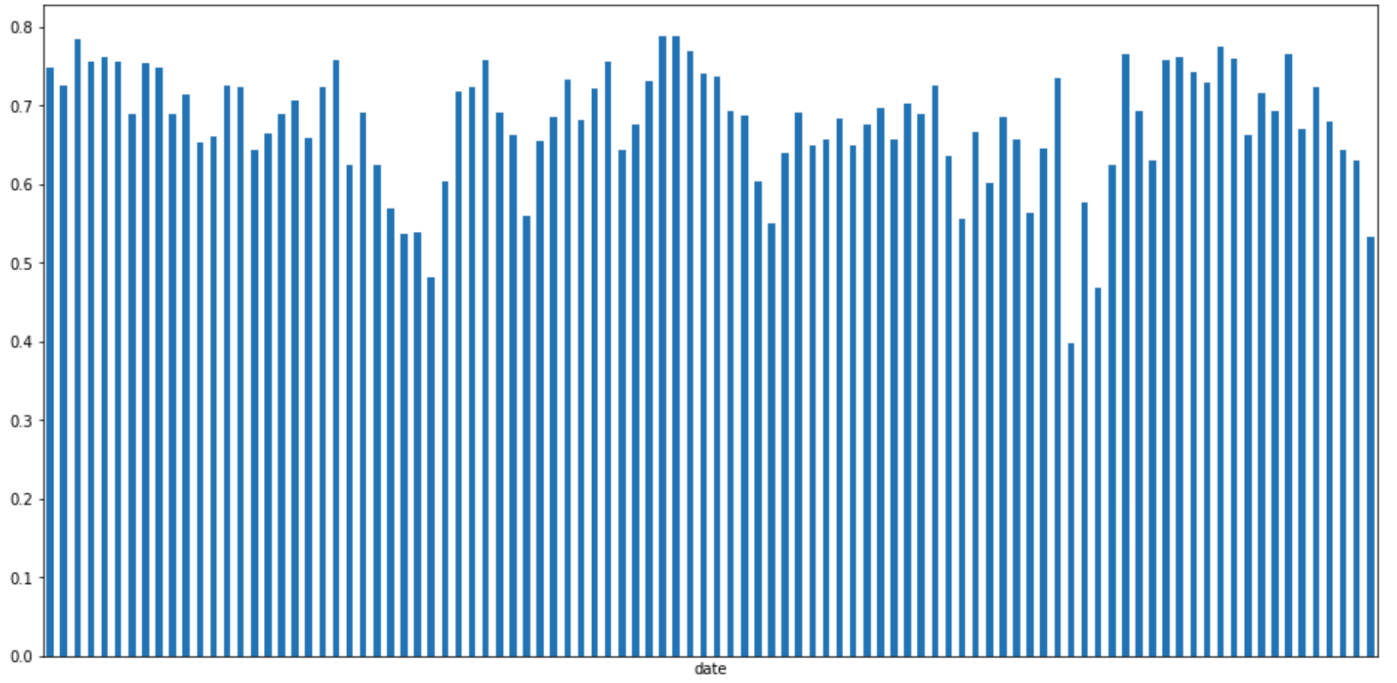
下面的表格比较了这一区间内mid price，price volatility, accuracy ratio的相关系数。可以看出ratio与std相关系数为负且通过pearsonr计算得到显著性水平为2.535e-46，同时计算的秩相关系数Spearmanr为-0.71, pvalue=1.428e-53。

	mid	std	ratio
mid	1.000000	0.356652	-0.304423
std	0.356652	1.000000	-0.678131
ratio	-0.304423	-0.678131	1.000000

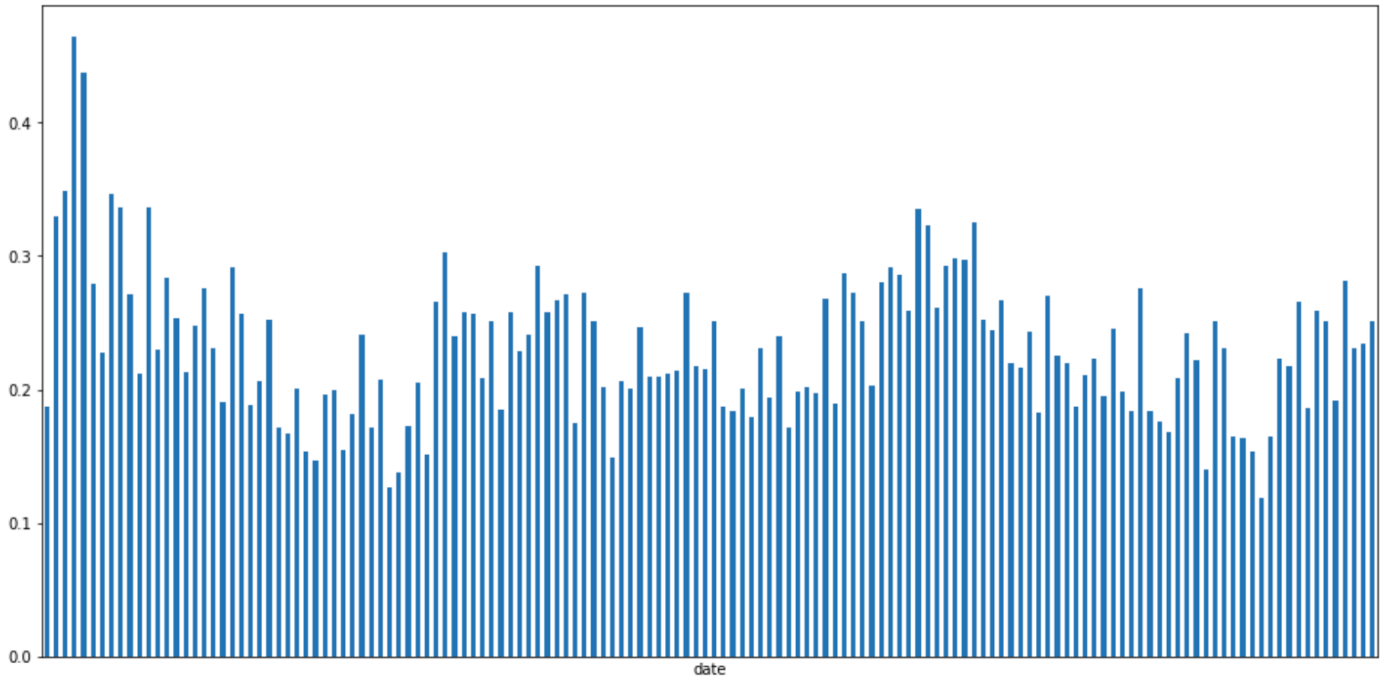
解决措施：每隔一段时间（一年），进行一次模型超参数重新调整，来适应不断变化的市场环境。

（2）预测不同时间区间的比较

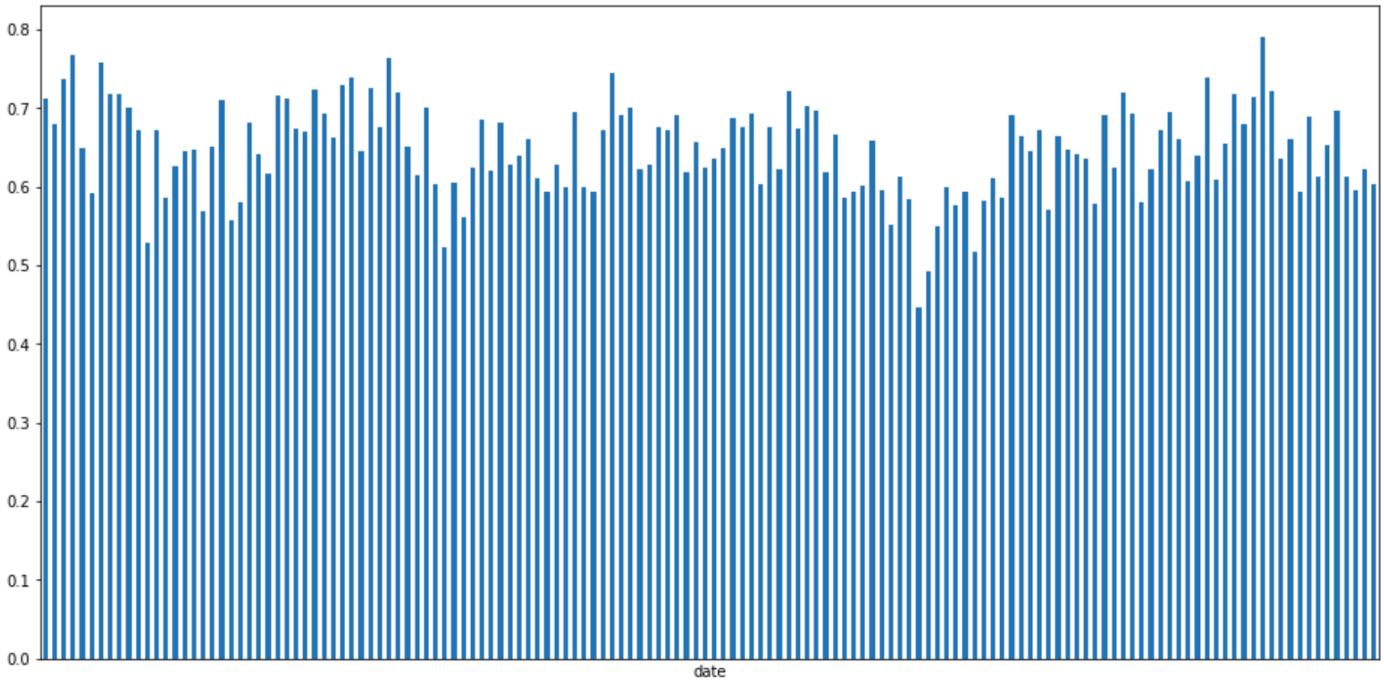
上面的预测模型使用的是30s价格走势，我运用同样的方法进行10s价格走势预测。为了节省时间，这次统一使用20180101-20180530的区间进行预测。可以看出此时预测精确度有较大提高。一开始，问题在于模型的预测值此时很大的偏向于0，这源于当时间区间设置为10s时，训练数据中具有更多标签为0的值。但是当我设置相关参数class_weight为'balanced'时，模型又极大的倾向于-1和+1，这同样可能是其中的标签为0的值太多，因此导致了当使用balanced时，预测值中0的权重变得很小。所以我采取了对于过多的0标签进行重新采样的方法，这样就可以保证各个标签之间样本较为均衡。



（1）未进行训练数据采样

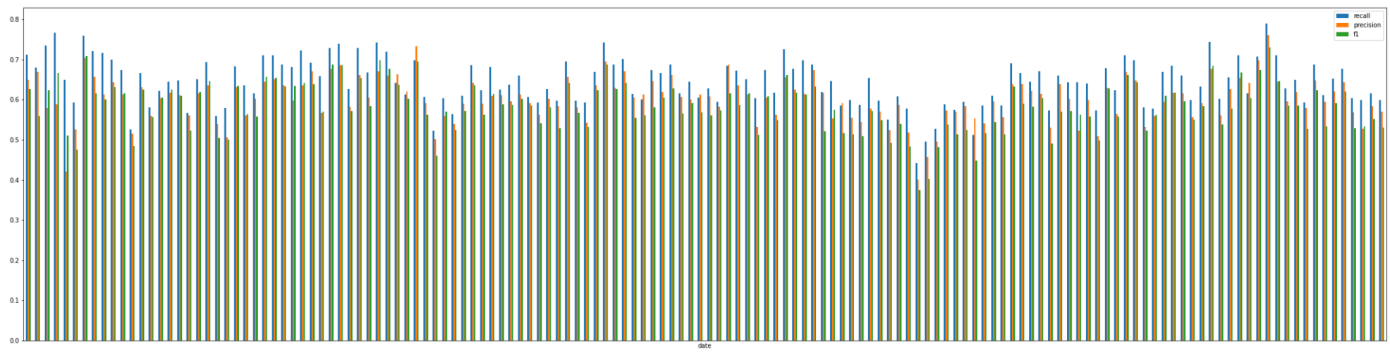


（2）训练模型class_weight设置为balanced



（3）通过采样丢弃大量的0标签，使得训练数据各标签样本量近似相同

从上面的结果来看，进行重新采样效果最好，并且在2018年预测10s的表现超过了预测30s的情形。为了进一步分析模型的表现情况，防止模型过度预测单一标签，我查看了模型的精准度，召回率等指标如下：



模型这一区间内整体的混淆矩阵为：

```
array([[ 26049,  28807,  11702],
       [ 72697, 322589,  76903],
       [  3872,   7383,  10558]])
```

从以上的分析来看，通过进行重采样，模型的预测能力有了很大提高，精准率和召回率都比较高，基本大于0.5的水平。说明随机森林模型是具有预测能力的。

3 lightgbm模型预测结果

随机森林是重要的bagging集成算法，另一类集成算法是gradient boosting，lightgbm是一个轻量的gradient boosting框架，相比于xgboost更轻便灵活。这里我首先进行了lightgbm的参数网格搜索，这里使用20160101-20170101的数据进行训练：

```
{'n_estimators': 50} 0.38018118089619674
{'max_depth': 3} 0.4134474620517345
{'learning_rate': 0.01} 0.41743792228481463
model final params
{'objective': 'multiclass', 'is_unbalance': True, 'n_estimators': 50, 'max_depth': 3, 'num_leaves': 40, 'learning_rate': 0.01, 'min_child_samples': 21, 'min_child_weight': 0.001, 'feature_fraction': 0.7, 'bagging_fraction': 0.6, 'bagging_freq': 15, 'reg_alpha': 0.001, 'reg_lambda': 8, 'random_state': 0}
Accuracy Score (Train): 0.545025
AUC Score (Train): 0.672773
```

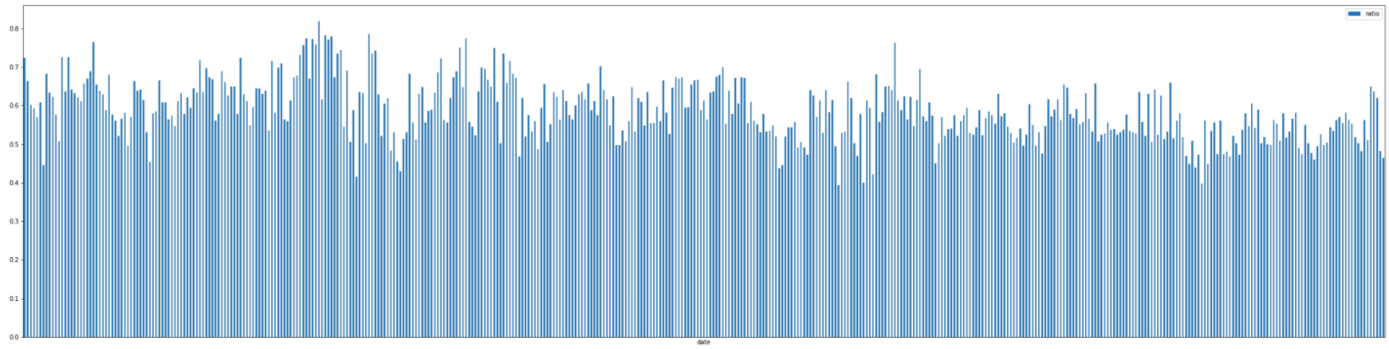
与此同时，我尝试了更大的搜索区间，进行随机搜索，随机搜索相比于网格更加快速，我设置的搜索参数区间如下：

```
param_dist = {'objective': ['multiclass'],
          'is_unbalance': [True],
          'n_estimators': randint(100, 1000),
          'max_depth': randint(2, 7),
          'num_leaves': randint(40, 200),
          'learning_rate': uniform(0.001,0.2),
          'min_child_samples': randint(10, 100),
          'min_child_weight': [0.001],
          'feature_fraction': [0.7],
          'bagging_fraction': [0.6],
          'bagging_freq': [15],
          'reg_alpha': [0],
          'reg_lambda': [8],
          'random_state': [0, 15]
          }
```

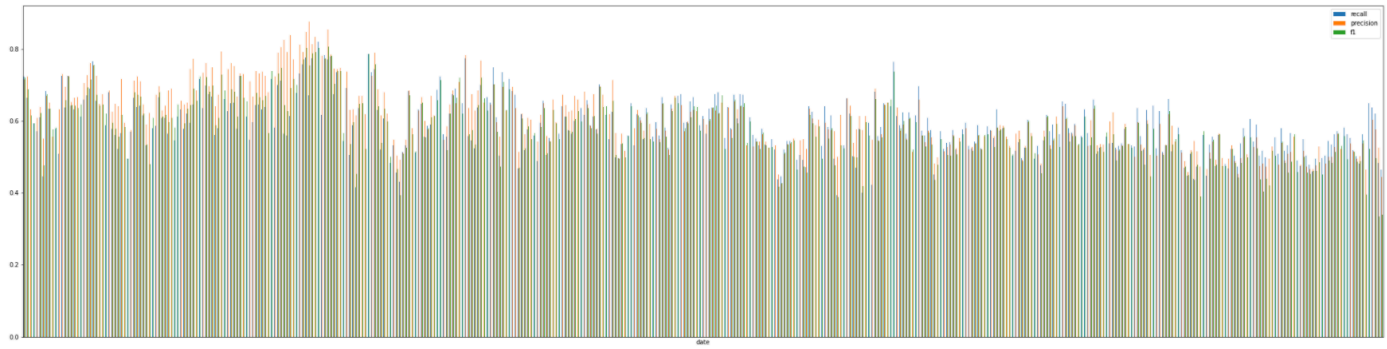
随机搜索得到的参数如下，可以看到相比于网格，参数更加灵活，而且最终的Accuracy Score 和AUC Score 均有一定提高。因此后续模型使用随机搜索得到的参数。

```
model final params
{'objective': 'multiclass', 'is_unbalance': True, 'n_estimators': 262, 'max_depth': 3, 'num_leaves': 128, 'learning_rate': 0.042648602718351296, 'min_child_samples': 78, 'min_child_weight': 0.001, 'featu
re_fraction': 0.7, 'bagging_fraction': 0.6, 'bagging_freq': 15, 'reg_alpha': 0, 'reg_lambda': 8, 'random_state': 15}
Accuracy Score (Train): 0.753675
AUC Score (Train): 0.680751
```

1）模型随时间decay情况：为了节约计算成本，我使用2016-2017的数据训练得到超参数，接下来分三组，分别使用201701-201706的数据训练，201707-201712的数据预测，201801-201806的数据训练，201807-201812的数据预测，201901-201906的数据训练，201907-201912的数据预测，将预测值拼接绘成如下的图像：



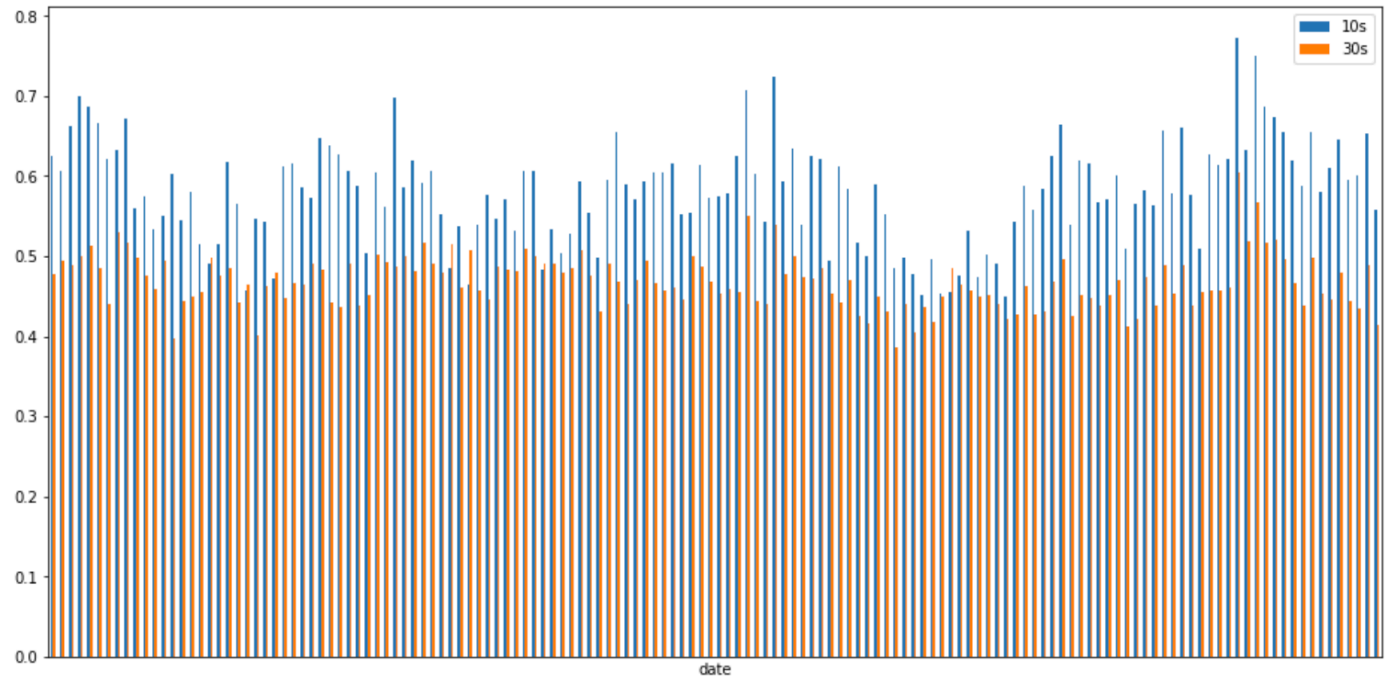
同样绘出精准率，召回率和f1如下图：



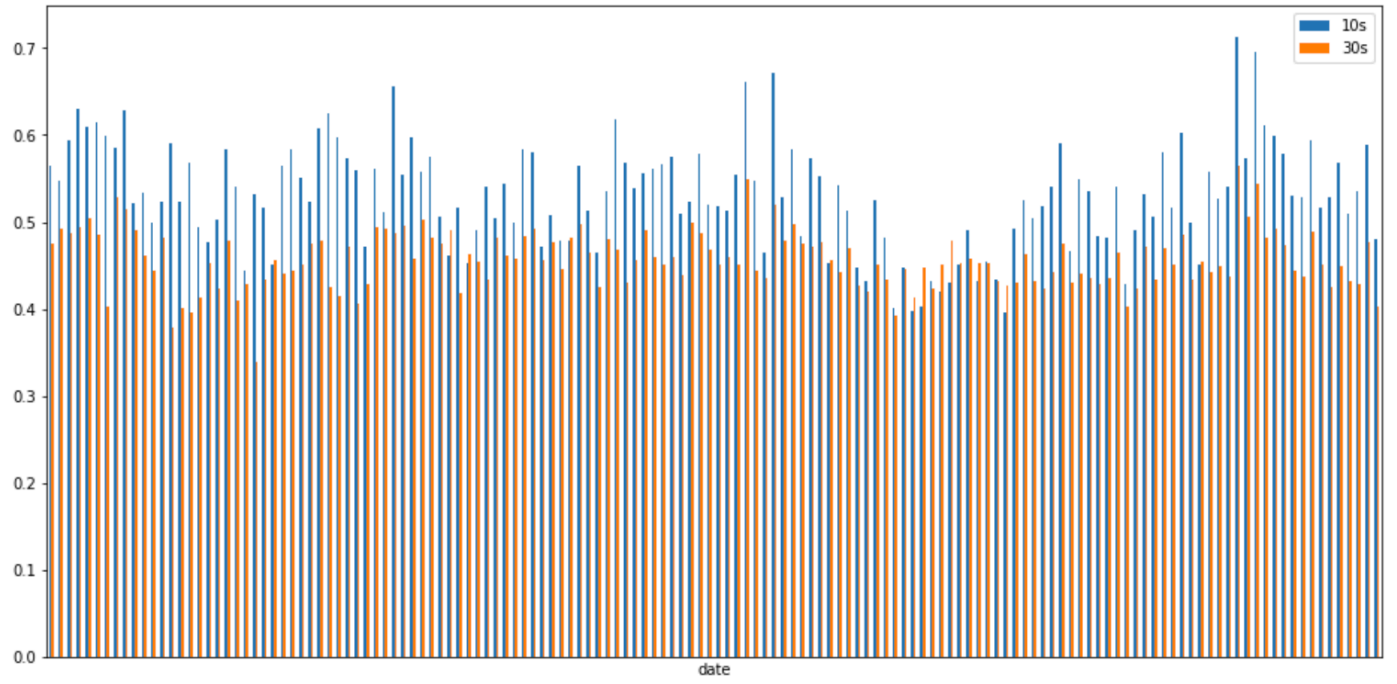
从这两幅图可以看出，随着时间的推移，模型训练的超参数具有一定的decay。但是这里decay的情况没有随机森林明显。

2）分别使用10s和30s数据训练

下面的图像分析了分别预测30s后的return和10s后的return，可以看出预测10sreturn得到了更高的准确度和f1-score，这说明lightgbm模型预测10s相对于预测30s具有优势。

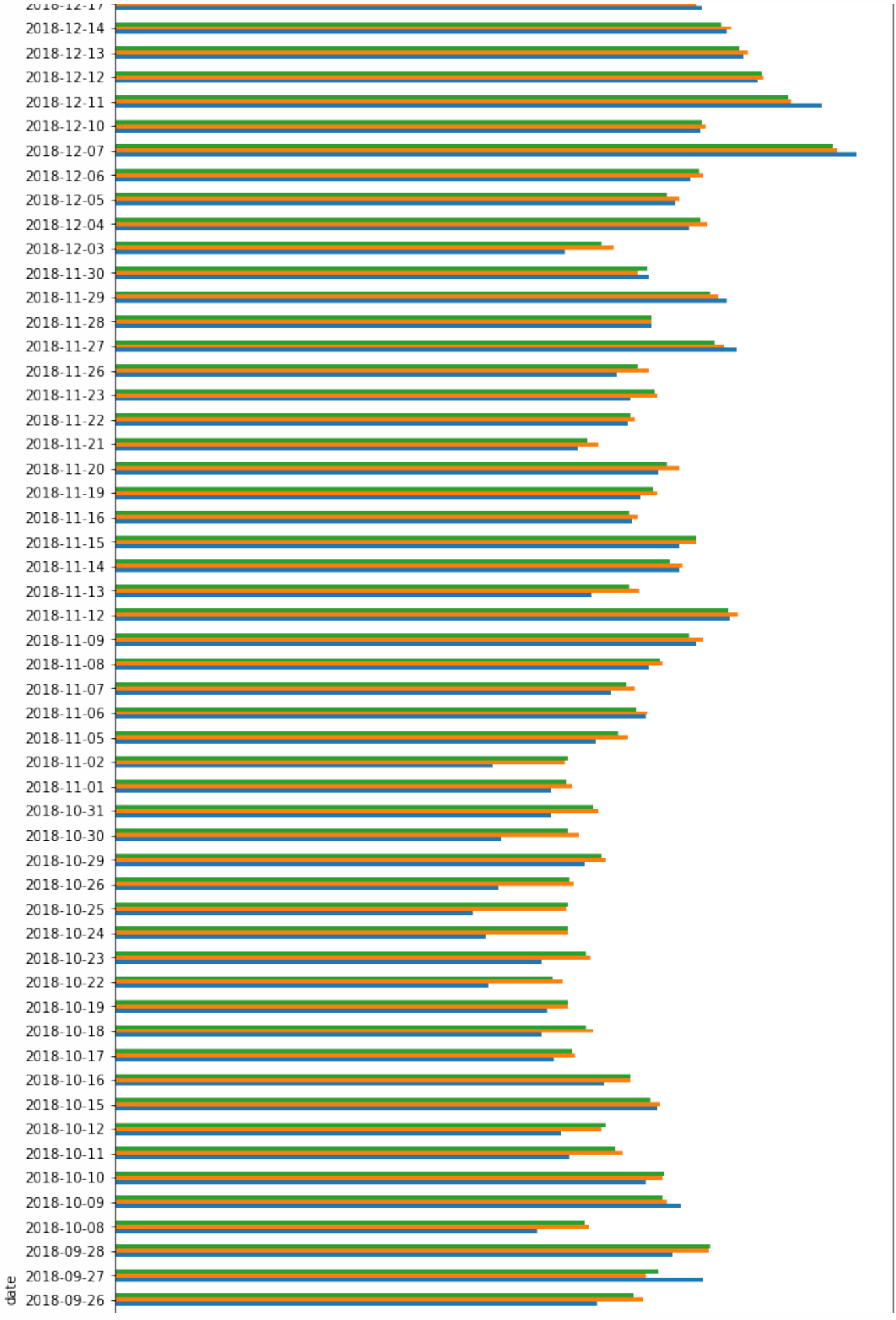


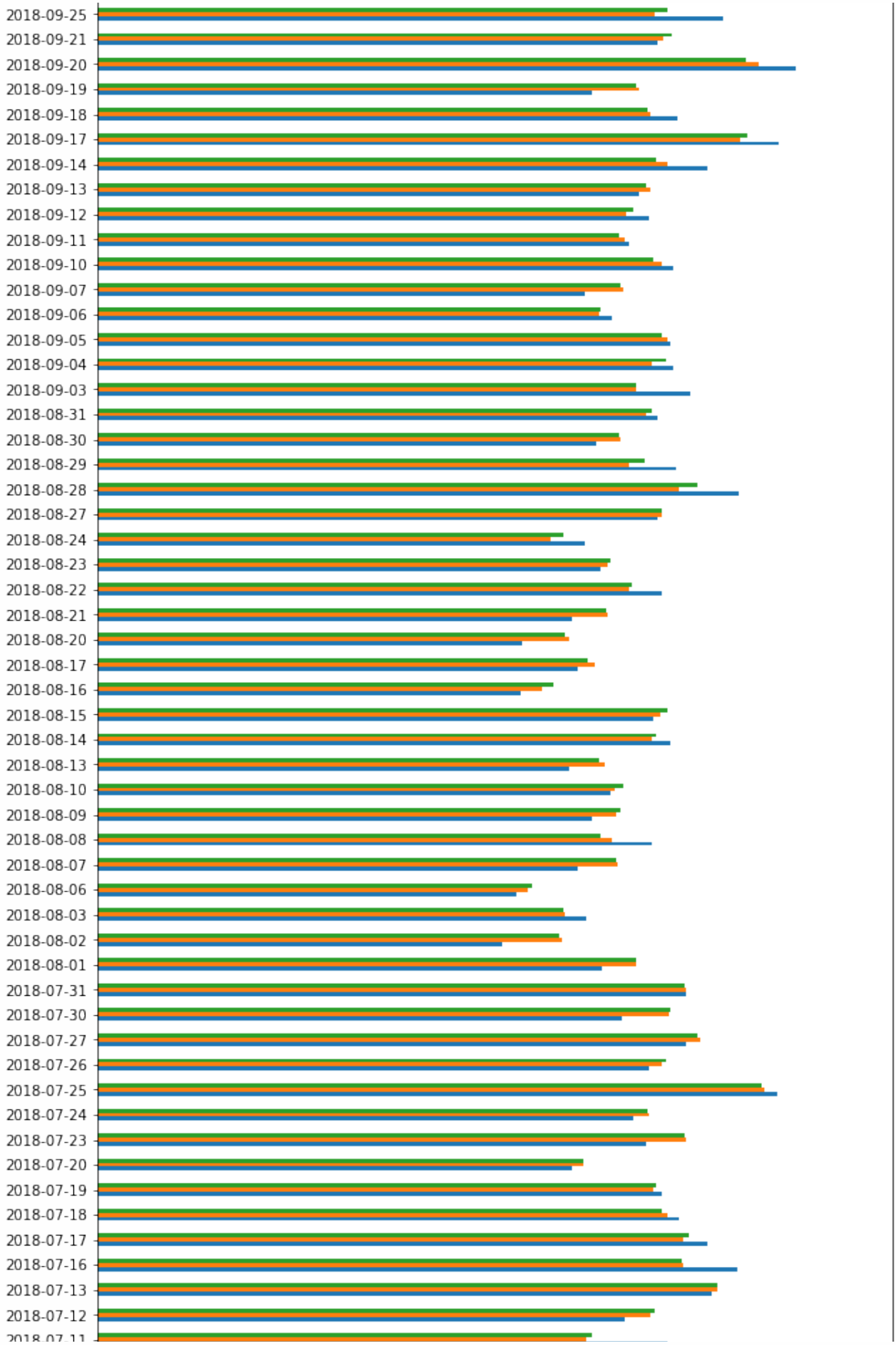
（1）比较30s数据和10s数据的准确度

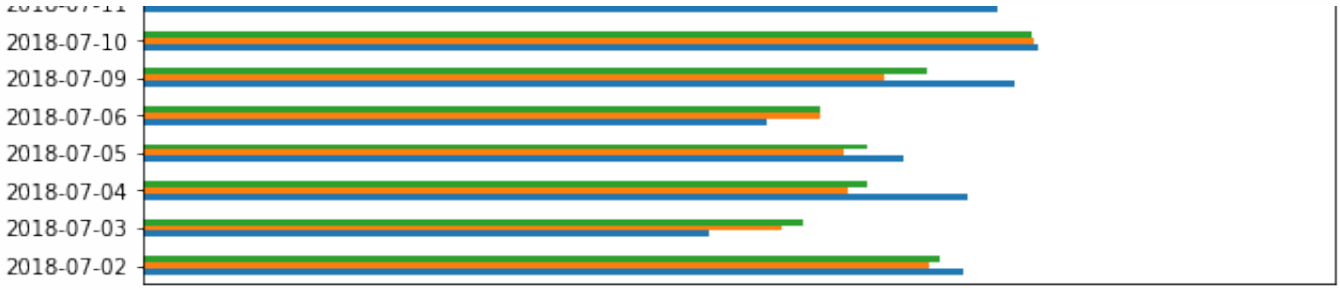


（2）比较30s和10s的f1-score

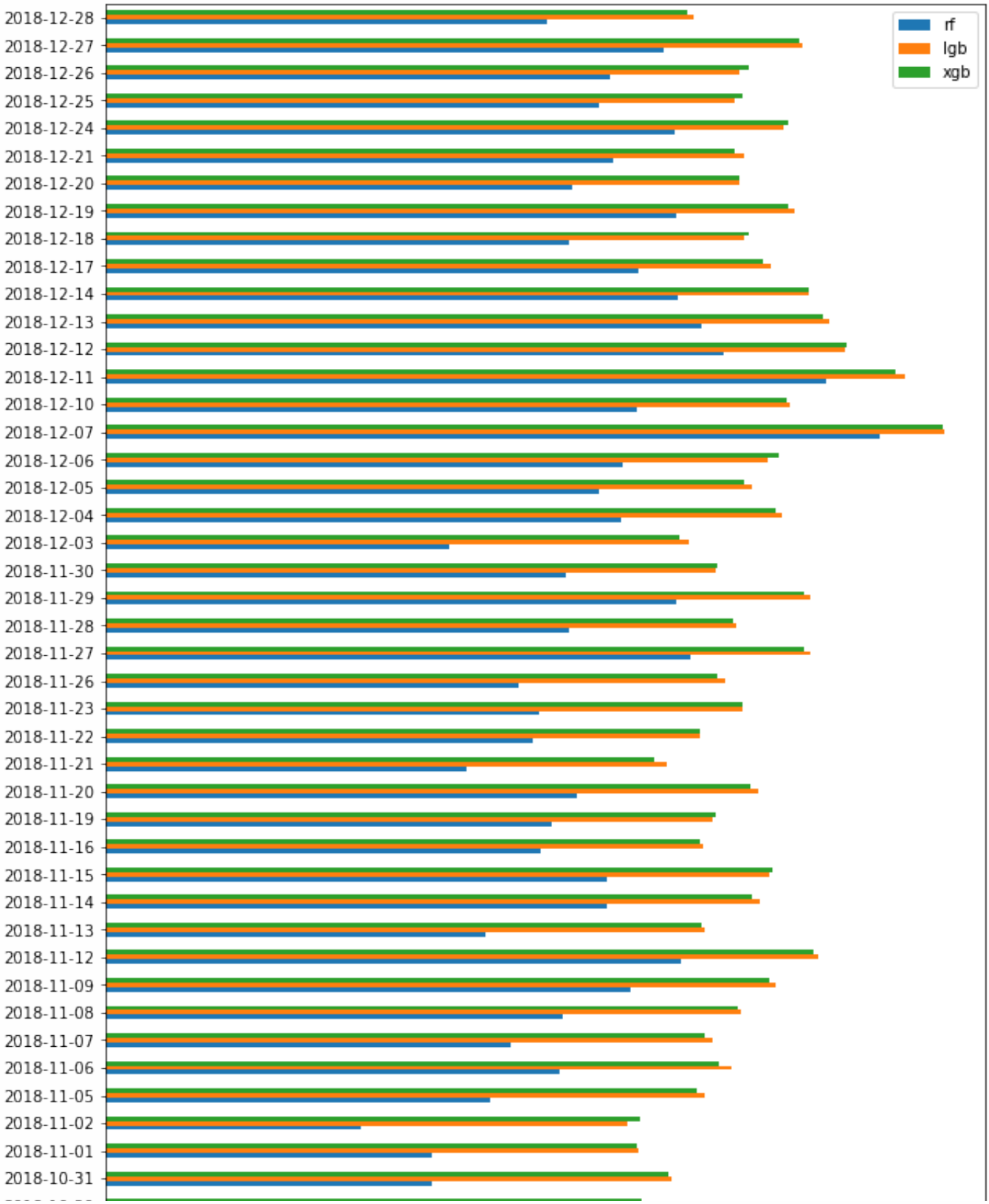
作为一个比较，使用20180101-20180630的数据进行训练，使用20180701-20181231的数据进行预测，绘制出random forest, lightgbm, xgboost 的相关预测效果比较图：

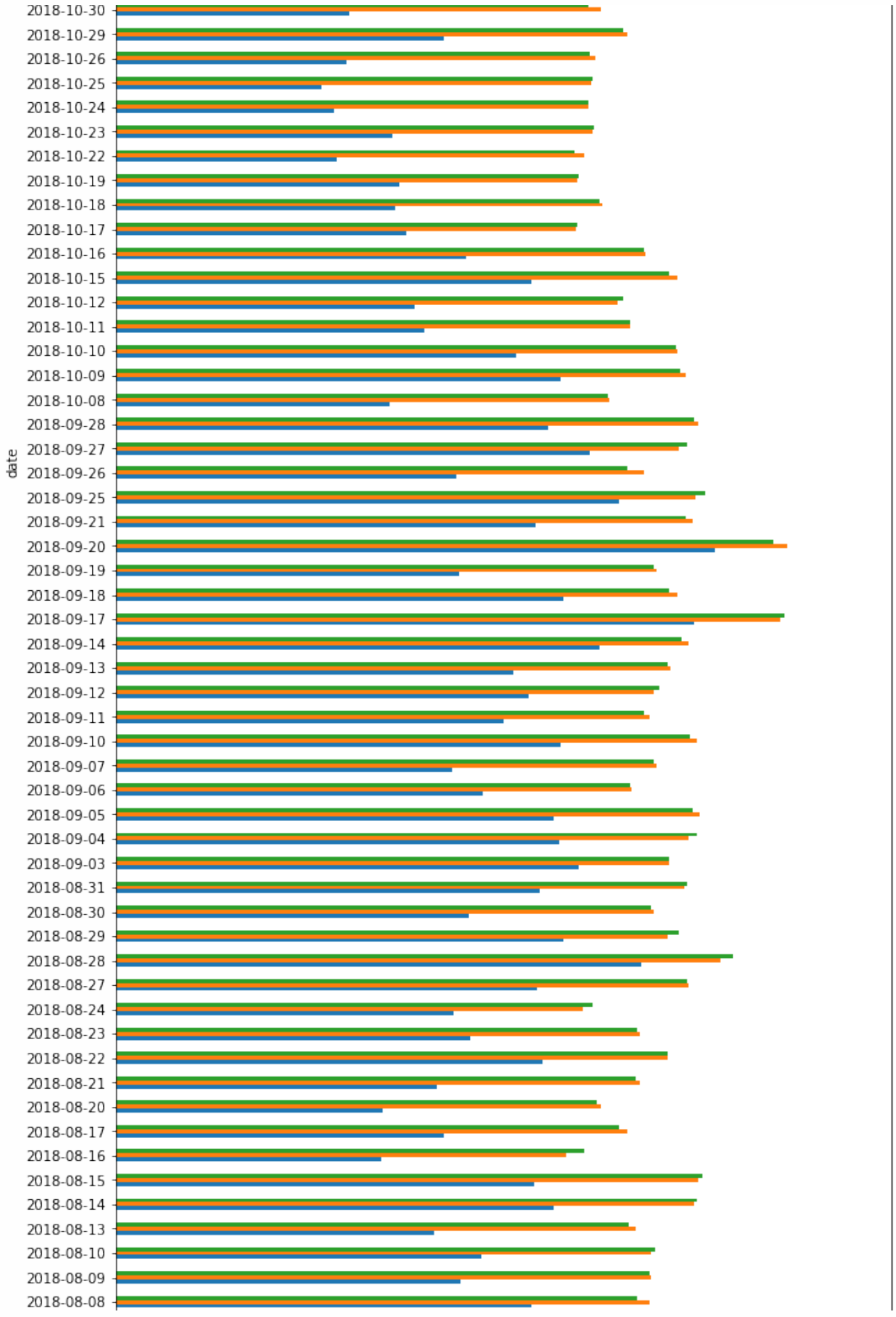

2018-12-14
2018-12-13
2018-12-12
2018-12-11
2018-12-10
2018-12-07
2018-12-06
2018-12-05
2018-12-04
2018-12-03
2018-11-30
2018-11-29
2018-11-28
2018-11-27
2018-11-26
2018-11-23
2018-11-22
2018-11-21
2018-11-20
2018-11-19
2018-11-16
2018-11-15
2018-11-14
2018-11-13
2018-11-12
2018-11-09
2018-11-08
2018-11-07
2018-11-06
2018-11-05
2018-11-02
2018-11-01
2018-10-31
2018-10-30
2018-10-29
2018-10-26
2018-10-25
2018-10-24
2018-10-23
2018-10-22
2018-10-19
2018-10-18
2018-10-17
2018-10-16
2018-10-15
2018-10-12
2018-10-11
2018-10-10
2018-10-09
2018-10-08
2018-09-28
2018-09-27
2018-09-26
date

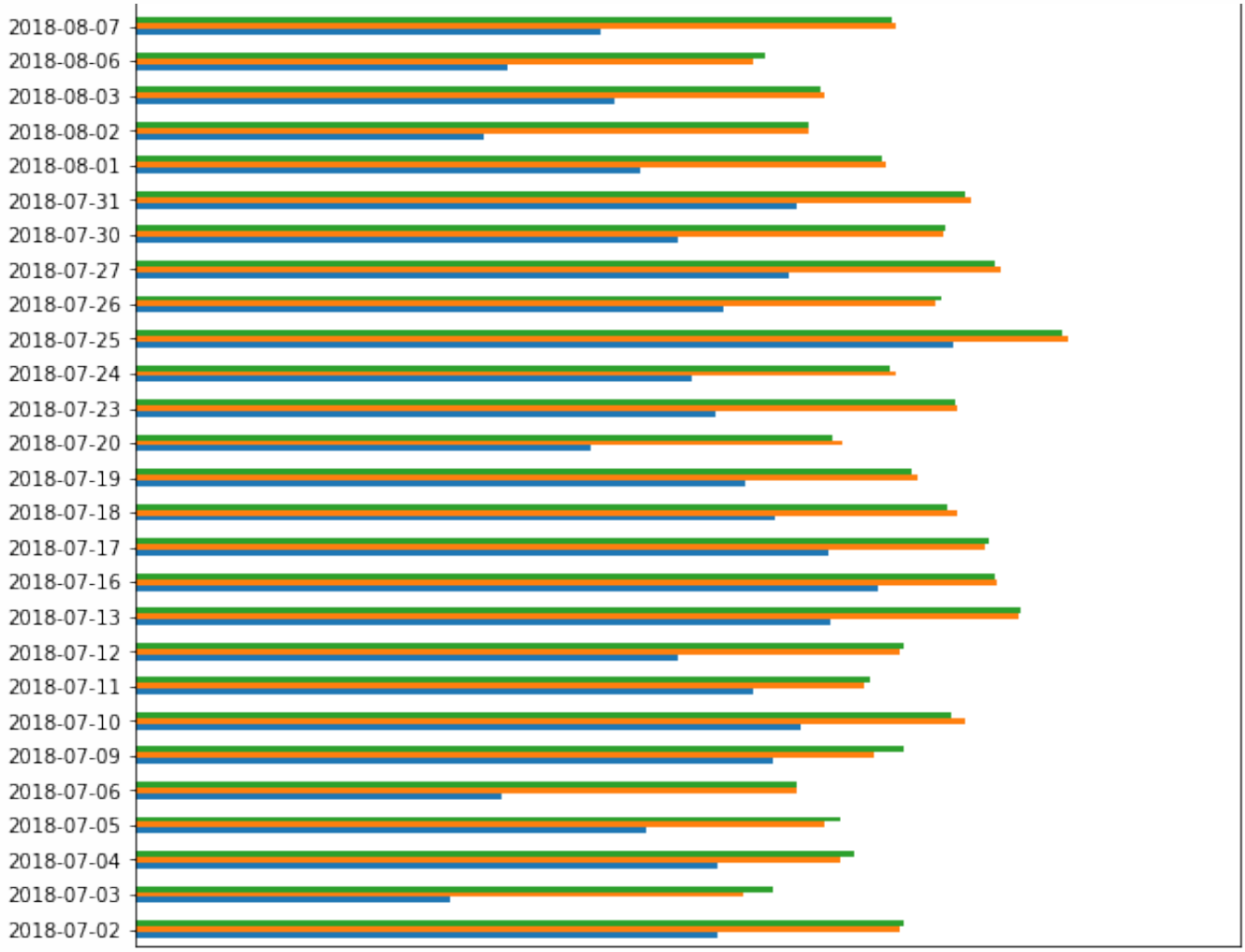

2018-09-25
2018-09-21
2018-09-20
2018-09-19
2018-09-18
2018-09-17
2018-09-14
2018-09-13
2018-09-12
2018-09-11
2018-09-10
2018-09-07
2018-09-06
2018-09-05
2018-09-04
2018-09-03
2018-08-31
2018-08-30
2018-08-29
2018-08-28
2018-08-27
2018-08-24
2018-08-23
2018-08-22
2018-08-21
2018-08-20
2018-08-17
2018-08-16
2018-08-15
2018-08-14
2018-08-13
2018-08-10
2018-08-09
2018-08-08
2018-08-07
2018-08-06
2018-08-03
2018-08-02
2018-08-01
2018-07-31
2018-07-30
2018-07-27
2018-07-26
2018-07-25
2018-07-24
2018-07-23
2018-07-20
2018-07-19
2018-07-18
2018-07-17
2018-07-16
2018-07-13
2018-07-12
2018-07-11


2018-07-10
2018-07-09
2018-07-06
2018-07-05
2018-07-04
2018-07-03
2018-07-02

(1)三种模型准确度比较


rf
lgb
xgb
2018-12-28
2018-12-27
2018-12-26
2018-12-25
2018-12-24
2018-12-21
2018-12-20
2018-12-19
2018-12-18
2018-12-17
2018-12-14
2018-12-13
2018-12-12
2018-12-11
2018-12-10
2018-12-07
2018-12-06
2018-12-05
2018-12-04
2018-12-03
2018-11-30
2018-11-29
2018-11-28
2018-11-27
2018-11-26
2018-11-23
2018-11-22
2018-11-21
2018-11-20
2018-11-19
2018-11-16
2018-11-15
2018-11-14
2018-11-13
2018-11-12
2018-11-09
2018-11-08
2018-11-07
2018-11-06
2018-11-05
2018-11-02
2018-11-01
2018-10-31


date
2018-10-30
2018-10-29
2018-10-26
2018-10-25
2018-10-24
2018-10-23
2018-10-22
2018-10-19
2018-10-18
2018-10-17
2018-10-16
2018-10-15
2018-10-12
2018-10-11
2018-10-10
2018-10-09
2018-10-08
2018-09-28
2018-09-27
2018-09-26
2018-09-25
2018-09-21
2018-09-20
2018-09-19
2018-09-18
2018-09-17
2018-09-14
2018-09-13
2018-09-12
2018-09-11
2018-09-10
2018-09-07
2018-09-06
2018-09-05
2018-09-04
2018-09-03
2018-08-31
2018-08-30
2018-08-29
2018-08-28
2018-08-27
2018-08-24
2018-08-23
2018-08-22
2018-08-21
2018-08-20
2018-08-17
2018-08-16
2018-08-15
2018-08-14
2018-08-13
2018-08-10
2018-08-09
2018-08-08



（2）三种模型f1-score比较

从上图来看，boosting model相比于random forest具有更高的f1分数，尽管三者的精确度相似且均大于50%。同时总体来看，lightgbm运行速度更快，是更好的选择。

4 模型总结

经过以上分析，初步得到如下结论：

1）数据预处理对于决策树模型来说提升不显著，可能主要由于决策树模型的if-else结构。

2）使用回归树并将结果转换成层次变量意义不大，在于进行结果转换时需要定义新的超参数，增加模型复杂度。

3）简单决策树模型无法使用足够的feature，因此表现较差。

4）集成模型random forest和lightgbm，xgboost效果更好，相对来说，random forest和lightgbm效果接近，xgboost比较耗费时间，运行速度很慢。

5）模型超参数的搜索需要重复进行，因为不同的市场环境会导致模型效果下降。

6）预测10s以后的return相对来说比预测30s以后的return更容易。

之后可能的任务：如何从预测模型中产生可能的交易策略？