1. Algorithmic Trading Challenge

1.1 Overview

Goal:

- The aim of this competition is to determine the relationship between recent past order book events and future stock price mean reversion following shocks to liquidity.

Submission Format(Things to be predicted) :

- For each observation a participant should provide 100 numbers describing the next 50 bid and ask prices(from t51 to t100) following a liquidity shock. There should be 50,001 rows in total (50,000 entries plus a header row) and 101 columns for each row.

Evaluation Method :

- Root Mean Square Error(RMSE) : $\sqrt{\frac{\Sigma_{t=1}^{n}(\hat{y}_t - y_t)^2}{n}}$, where $\hat{y}_t$ and y_t are the predicted/actual value respectively.

1.2 Data

The competition host provide 4 files for preparation， train.csv & test.csv, plus example_entry_example.csv&example_entry_naive.csv, the last two seems to be used as submission template, though I can't tell their difference. But it seems that these two files have similiar dimension, and differences seems to have patterns. Here's the comparison using pandas.DataFrame：

```
In [26]: ex_entry_linear == ex_entry_naive
Out[26]:
       row_id  bid51  ask51  bid52  ask52  bid53  ask53  bid54  ask54  bid55  \
0        True  False   True  False   True  False   True  False   True  False
1        True   True  False   True  False   True  False   True  False   True
2        True  False   True  False   True  False   True  False   True  False
3        True   True  False   True  False   True  False   True  False   True
4        True   True  False   True  False   True  False   True  False   True
5        True  False   True  False   True  False   True  False   True  False
6        True  False   True  False   True  False   True  False   True  False
7        True  False   True  False   True  False   True  False   True  False
8        True   True  False   True  False   True  False   True  False   True
9        True   True  False   True  False   True  False   True  False   True
10       True   True  False   True  False   True  False   True  False   True
11       True   True  False   True  False   True  False   True  False   True
12       True   True  False   True  False   True  False   True  False   True
13       True  False   True  False   True  False   True  False   True  False
14       True   True  False   True  False   True  False   True  False   True
```

Anyway, the meaning of provided labels in training/test set are shown as below:

Table 1 Data Schema

Variable Type	Predictors																				Responses				
Event Time								t=1				...	t=49				t=50				t=51		...	t=100	
Column Id	1	2	3	4	5	6	7	8	9	10	11	...	200	201	202	203	204	205	206	207	208	209	...	305	306
Variable Name	row_id	security_id	p_tcount	p_value	trade_vwap	trade_volume	initiator	transtype<t>	time<t>	bid<t>	ask<t>	⋮	transtype<t>	time<t>	bid<t>	ask<t>	transtype<t>	time<t>	bid<t>	ask<t>	bid<t>	ask<t>	⋮	bid<t>	ask<t>
													Trade				Quote								
													Liquidity Shock												

Table 2 Data Fields

Variable Name	Description	Type	Example
row_id	Unique row identifier	Integer	6
security_id	Unique stock identifier	Integer	24
p_tcount	Count of previous day's on-market trades in current security	Integer	670
p_value	Sum of previous day's on-market trade values in current security (£)	Integer	65,000
trade_vwap	Volume-weighted average price of the trade causing the liquidity shock (pence)	Double	4.250
trade_volume	Size of the trade causing the liquidity shock (number of shares)		1,500
initiator	Whether the trade is buyer or seller initiated ('B'=Buyer, 'S'=Seller)	String	B
transtype<t>	Whether the time-series event is a trade or a quote ('T'=Trade, 'Q'=Quote) at event time t	String	T
time<t>	Event time (HH:MM:SS.mmm) at event time t	String	10:05:36.488
bid<t>	Best buy price (pence) at event time t	Double	213.85
ask<t>	Best sell price (pence) at event time t	Double	214.10

The Size of each files are listed:

```
##train.csv
In [10]: training_set.index
Out[10]: RangeIndex(start=0, stop=754018, step=1)

##test.csv
In [22]: test_set.index
Out[22]: RangeIndex(start=0, stop=50000, step=1)

##example_entry_linear.csv
In [13]: ex_entry_linear.index
```

```
Out[13]: RangeIndex(start=0, stop=50000, step=1)

##example_entry_naive.csv
In [16]: ex_entry_naive.index
Out[16]: RangeIndex(start=0, stop=50000, step=1)
```

1.3 Selected Solutions

Key Methods: Time Partitioning Prediction, Random Forest, Ensemble Methods and Feature Extractions.

a. Top1 Solution

a. Results:

- Ranked 1st in the private leaderboard, 4th in the public leaderboard.

a. Work Flow:

- **Step1: Time Interval Partitioning Algorithm**, Due to identity between t=50 and t=51, the partition start from t=52 to t=100. Use greedy algorithm to make partition(when the error in algorithm step7 begins to rise, then stop and separate), and use different sub-models($M_{bit}(t)$ & $M_{ask}(t)$ for bit/ask prediction respectively). Furthermore, assume the length of later segmentation is larger than the earlier ones(length(C_{i+1}) > length(C_i)), this is derived from the main hypothesis of the model:

  ```
  'an always increasing prediction error may require averaging longer price time series to obtain a constant price prediction with an acceptable error'
                                                    -Cited from author
  ```

  Different $M_{bit,i}(t)$ & $M_{ask,i}(t)$ are used to describe each time periods(C_i):

$$M_{\text{bid}}(t) = \sum_{i=1}^{K} a_{i,t} M_{\text{bid},i}(t),\ M_{\text{ask}}(t) = \sum_{i=1}^{K} a_{i,t} M_{\text{ask},i}(t),$$
$$\text{where } a_{i,t} = \begin{cases} 1 \text{ if t} \in C_i, \\ 0 \text{ otherwise}, \end{cases} \qquad t \in [52, 100].$$

Algorithm 1 Time interval partitioning algorithm

```
b ← e ← 52; P ← NULL; i ← 1
while b < 100 do
    C_i ← NULL; e ← b + length(C_i); bestError← ∞
    repeat
        C_i ← createTimeInterval(b,e)
        C_all ← createTimeInterval(e+1,100)
        error← evaluateModel(P,C_i,C_all)
        if bestError > error then
            bestError← error
        end if
        e ← e + 1
    until bestError≠ error
    addTimeInterval(P, C_i)
    i ← i + 1; b ← e
end while
```

- **Step2: Feature Extraction**, The author have extracted over 150 features(divided into 4 classes: *Price, Liquidity book, Spread, Rate*) from the original data, using a specified R module (I guess)

> *Price:* Price features provide information about the bid/ask normalized price time series (price values divided by the post liquidity shock price). Technical analysis [4] and statistical estimators are the fundamental instruments to compute these predictors (e.g., the detrended price oscillator, the exponential moving average of the last n [bid/ask] prices before the liquidity shock, the number of price increments during the last n [bid/ask] prices before the liquidity shock).
>
> *Liquidity book:* This class contains all features able to provide information about the depth of the liquidity book (e.g., liquidity book improvements in the last n time periods understood as bid/ask price increases between two consecutive quotes).
>
> *Spread:* Spread related features are meant to distill information about the bid/ask spread. As price features, technical analysis and statistical estimators are the fundamental instruments to compute these predictors. Before computing the predictor, spread time series were divided by the minimum price increment allowed for the particular *security_id* (e.g., exponential moving average of the last n spreads before the liquidity shock).
>
> *Rate:* This class of features provides information about the arrival rate of orders and/or quotes (e.g., number of quotes and/or trades during the last n events).

- **Step3: Modeling Approach Selection,** First, using MIC(Maximal Information Coefficient) to analyze the mutual infromation between features and the dependent variable, revealing a very low mutual information and non-functional relationships; then tried both Random Forest and Gradient Boosting Machines to select the data.(4-cross validation, and the **refined training set** consisted of 1.same size 2. same combination of security_id as the test set, then choose RF for lower RMSE cost.)

#

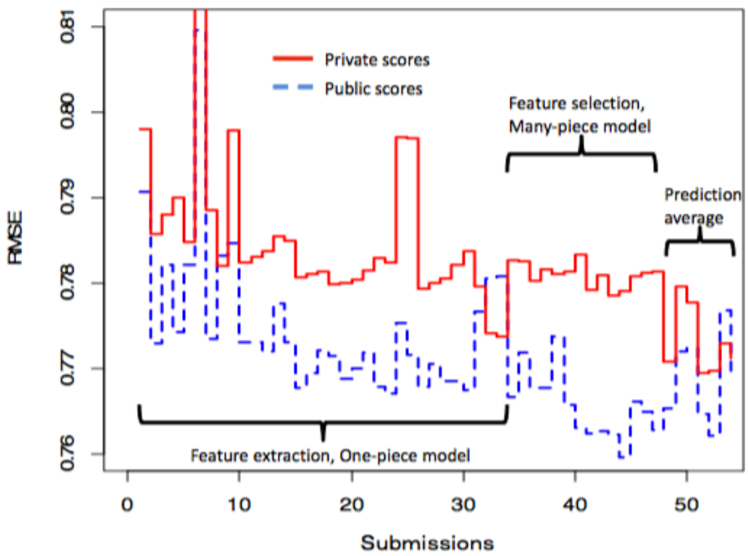
- **Step4: Feature Selection**, inspired by a similiar method applied to the *Kaggle*'s Heritage Health Prize dataset. The algorithm are divided into 2 parts(1. step1 - 8: to get the quasi-optimized S_f quickly, 2. Rest of the steps: to make local adjustments on these S_f feature set).

Algorithm 2 Feature selection algorithm

1: Train a single piece model using all S features
2: Compute model performance against the test set
3: Rank features importance (RF importance method)
4: **for** each subset size $S_i = S, S-1, \ldots, 1$: **do**
5: Retrain the model with only S_i most important features
6: Re-compute individual variable importance and re-rank
7: Fit the model to S_f features and rank individual features
8: **end for** S_i
9: Determine which S_i yielded the smallest RMSE. Call this S_f
10: **repeat**
11: Choose a set of semantically similar features from S_f
12: Select the feature with less rank not selected before
13: Evaluate the model performance
14: If smaller RMSE, then remove the feature
15: **until** no improvement
16: **repeat**
17: Choose a feature set among the already removed in Steps 1)-9) considering only those semantically orthogonal with the already selected in Steps 1)-15)
18: If smaller RMSE, then we add the feature to S_f
19: **until** no improvement

- **Step5: Validation**:
    - **Feature Set** Validation: Ironically, the author used the same S_f set for both F_b and F_a (bit and ask feature set for bit and ask price respectively) at the final stage due to the lack of time for calcu dlation. **BUT, for the different time period, the F_b is different.**
    - **Optimal Time Segmentation**: The final partition of the time period is {t=52, t=53, t=54–55, t=56–58, t=59–64, t=65–73, t=74–100}, which I thought quite make sense: the nearest period has the highest weight, so is sepearted.

- **Model Performance via methods**:



a. Comments

- Pros: Innovated on both *Time Segmentation* and *Feature Selction* process.
- Cons: In reality, the company only cares about the most recent prediction on the price, input new data and genreate new prediction, so there won't be error accumulation, the *Time Segmentation* process seems to be meaningless, plus, the feature set for the F_a is not optimal.

b. Forum Disscussion

- Christopher Hef (4th in this Competition)
    - Observations on data
    1. Bids/asks from T=1...T=47 seemed to provide little predictive value;
    2. The error contribution right at the market open (at 8AM) was extremely large;
    3. Prediction accuracy varied across time. Using a holdout set & one of our models, I found that the error rose as you got farther from the liquidity-event trade;
    4. The "liquidity event" trades did not seem to impact prices very much.

- Sergey Yurgens(6th in this Competition)

  Here is the secret recipe for Linear Regression meal:

    - go to your friendly neighbor datastore and choose couple fresh pieces of data (day1 and last 50k)
    - cut out bones and extra fat (leave only columns 5 170 206 207)
    - cook separately "seller initiated" transactions and "buyer initiated" transactions using your favorite linear regression function (do it separately for each askN and bidN to be predicted)
    - use 200 created LRs to calculate required predictions and nicely plate them into submission file
    - Serve hot, because you do not want to miss 0.77590 public score and 0.77956 private score :)

- Cole Harris • (9th in this Competition)
    - I had separate models for t<60 & t>60.