Summary Report 3

**SOI Team for price prediction**

*Bad News:*

We tested the concurrent result of using quotes mid points to calculate the bucket return. Compared to last week’s best results, new model produces much weaker results. We also add a time delay to all quotes, and the results are still not satisfactory.

Given those results, and the computational capacity limit of out laptops, we decide to postpone the testing of using EMA of quotes to classify trade’s BUY/SELL. The difficulty for using EMA of quotes is the need to pre-process all the quotes and trades before parsing them. If we only need the mid-points of the recent quotes, the process gets much simpler and computational time only doubles; yet for when first calculating EMA of quotes, then feeding into the parser, the pre-processing alone can consume up so many time. For details, the implementation is wrapped up as a function filter\_trades\_quotes\_EMA in parser.csv. More importantly, based on the results we have thus far obtained, we find that using quotes only made the results worse (substantially worse that is), the results are presented below.

Use quotes mid point as bucket return proxy, and delay all the quotes by alpha millisecond, filter out large trades (>10000).

The parameters ranges are as the following:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Start | End | Step |
| Bucket Size | 1000 | 10000 | 1000 |
| Time Bin | 30 | 180 | 30 |
| Delay Time(s) | 0 | 0.1 | 0.005 |

The results are depressing:

|  |  |
| --- | --- |
| **(bucketVol\_time\_bin\_delay)** | **R^2** |
| 10000 \_ 150 \_ 0.04 | 14.8% |
| 10000 \_ 150 \_ 0.09 | 14.4% |
| 7000 \_ 150 \_ 0.09 | 14.3% |
| 10000 \_ 150 \_ 0.055 | 14.1% |
| 8000 \_ 150 \_ 0.06 | 13.9% |
| 9000 \_ 150 \_ 0.07 | 13.7% |
| 9000 \_ 150 \_ 0.055 | 13.3% |
| 10000 \_ 180 \_ 0.055 | 13.1% |
| 10000 \_ 150 \_ 0.045 | 13.0% |
| 8000 \_ 180 \_ 0.06 | 13.0% |
| 8000 \_ 150 \_ 0.09 | 12.6% |
| 10000 \_ 150 \_ 0.035 | 12.4% |
| 10000 \_ 180 \_ 0.04 | 11.9% |
| 10000 \_ 150 \_ 0.1 | 11.5% |
| 7000 \_ 150 \_ 0.025 | 11.4% |
| 8000 \_ 150 \_ 0.045 | 11.3% |
| 9000 \_ 150 \_ 0.075 | 11.3% |
| 7000 \_ 180 \_ 0.09 | 11.2% |

The case when delay time equals 0 can be used to compare with our previous implementation (when we ignore quotes completely and only implemented a Lee/Ready type of classification based on transaction prices). And if you remember from last week’s report, some concurrent regression yielded R^2 as large as 41.3%; but if we substitute quotes mid-points and use them to calculate bucket returns, the best concurrent regression only yielded a 14% R^2.

*Good News:*

So far, we conclude that the Lee-Ready concurrent model involving three parameters: bucket size, time bin, and threshold is the best model. And after taking a closer look at the data, we got inspired with a new direction to improve the Lee-Ready concurrent model. We found that the latency variable is the difference between time (the variable in the data, i.e. the time when the corresponding row of data received) and exchange time.

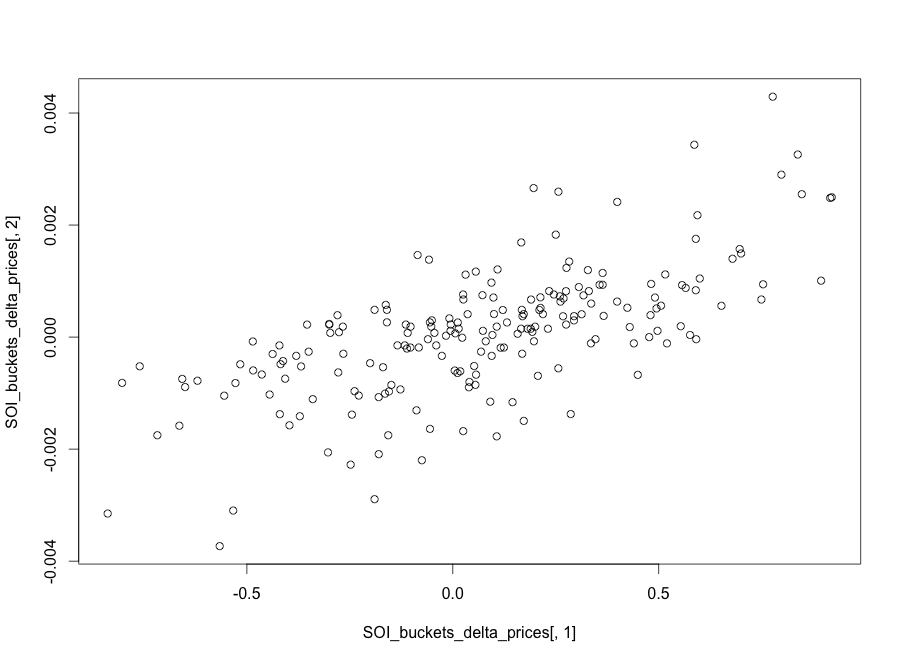
lantency\*0.001 = time – exchange time

* Thus, exchange time = time – latency\*0.001 (1)

Use formula (1) to adjust the time variable, we can get the actual time that each quote and trade got posted in the Exchanges. We rerun the optimization of the Lee Ready model (Bucket Size, Time bin, threshold) and use the trades return as bucket return proxy. (so the only difference between this set of runs and last week’s run was adjusting the trade time for latency). And we have obtained better R^2.

For best Lee-Ready results, slightly improved the R^2 from 41.3% to 52.18%.

|  |  |
| --- | --- |
| **(Bucket,Time bin, Trade EX)** | **Adj-R2** |
| 2000 \_ 90 \_ 2000 | 52.18% |
| 15000 \_ 135 \_ 1000 | 50.03% |
| 2000 \_ 105 \_ 2000 | 49.68% |
| 14000 \_ 135 \_ 1000 | 47.79% |
| 15000 \_ 105 \_ 1000 | 47.29% |
| 13000 \_ 135 \_ 1000 | 47.25% |
| 11000 \_ 135 \_ 1000 | 47.18% |
| 10000 \_ 135 \_ 1000 | 47.11% |
| 13000 \_ 135 \_ 2000 | 47.10% |
| 15000 \_ 60 \_ 1000 | 47.04% |
| 14000 \_ 150 \_ 1000 | 46.40% |
| 10000 \_ 150 \_ 1000 | 46.40% |
| 15000 \_ 120 \_ 1000 | 46.14% |
| 15000 \_ 45 \_ 1000 | 46.06% |
| 2000 \_ 30 \_ 2000 | 46.04% |
| 8000 \_ 135 \_ 1000 | 44.87% |
| 11000 \_ 150 \_ 1000 | 44.72% |
| 13000 \_ 150 \_ 1000 | 44.09% |
| 13000 \_ 45 \_ 1000 | 42.93% |



but the lagged prediction result is still not very satisfactory