琪石量化项目实战研究班笔试

Introduction

In high frequency trading, level1 data is a crucial component of the system. Level1 data includes quote update (bid/ask’s price and size changes), and trade update. Normally exchanges publish order-by-order data, and the trading system can derive these two updates from such one data stream. In this example, the exchange publishes quote and trade updates separately. In live trading and offline simulation, we need to query the state of quote and trading volume at any given time, and also track their statistics. For this project, you can use 1-month of level1 data [here](https://scientech-pub-training.s3.amazonaws.com/qishi_quant_class_2020/2019-09.data.tar.gz).

*Thanks to Laura’s modifications to the original questions, and we added comments/solutions below the questions. Before we do any data analysis on the dataset, it is always a good exercise to inspect the data, identify any peculiar data and clean the data. In this example, there are some quotes that are missing bid or ask price, caused by 10% limits. It is better to filter out those quotes in the analysis.*

Part I.

Given above level1 data, write Python/C++ to merge the two time series data, and provide callback interface. One example is:

void quote\_update(Symbol sym, Quote new\_quote\_event)

void trade\_update(Symbol sym, Trade new\_trade\_event)  
You can come up with your own interface. The timestamp column in the file is the epoch time in microsecond. The timestamp of quotes and trades were recorded by the same server. We expect your program to act like a data replayer for any given day, strictly following the time ordering between the events. If the quote and trade events have the same timestamp, process trade first.

*For HFT system, merging data/events is an important component of the system. The trading signal/strategy codes are usually just a block of codes handling certain events (quote/trade). In this example, the upstream event handler can be written like this in C++:*

*std::vector<std::shared\_ptr<Quote>> quotes;*

*readFiles<Quote>(&quotes, kQuoteFile, parseQuoteFromString);*

*std::vector<std::shared\_ptr<Trade>> trades;*

*readFiles<Trade>(&trades, kTradeFile, parseTradeFromString);*

*Universe universe;*

*size\_t i = 0, j = 0;*

*while (i < quotes.size() || j < trades.size()) {*

*if (i >= quotes.size()) {*

*Symbol &symbol = universe.getSymbol(trades[j]->ticker());*

*symbol.addTrade(\*trades[j]);*

*++j;*

*} else if (j >= trades.size()) {*

*Symbol &symbol = universe.getSymbol(quotes[i]->ticker());*

*symbol.addQuote(\*quotes[i]);*

*++i;*

*} else if (quotes[i]->ts() < trades[j]->ts()) {*

*Symbol &symbol = universe.getSymbol(quotes[i]->ticker());*

*symbol.addQuote(\*quotes[i]);*

*++i;*

*} else {*

*Symbol &symbol = universe.getSymbol(trades[j]->ticker());*

*symbol.addTrade(\*trades[j]);*

*++j;*

*}*

*}*

*Later quants/traders just need to implement his/her own logic in the addTrade/addQuote function. For example, a simple strategy pseudo code:*

*addQuote(Quote quote)*

*current\_bid\_ = quote.bid*

*current\_ask\_ = quote.ask*

*addTrade(Trade trade)*

*if(trade.qty > 4\*prev\_20\_day\_trade\_median\_qty)*

*Buy at current\_ask\_ or Sell at current\_bid\_ with qty = median\_qty. Direction is the same as market trade.*

*We can then backtest this strategy using the historical data.*

Part II.

For all the analysis below, ignore the lunch break period, which is 11:30AM-1PM local time, and you can treat trading time as 9:30AM-11:30AM and 1-3PM local time. Use all available data to compute following:

1. Plot the dollar-volume weighted spread profile (<spread> vs time at minute level). Basically compute time-weighted average spread in minute bucket for every stock, then do the dollar-volume weighted average across the universe of stocks in each minute. Could you come up with a function to fit the curve?

Dollar-volume: dollar notional trade volume, i.e.

You can use the below definition for time-weighted average spread for a given stock:

**Suppose in each minute interval (T, T’) there are N quote updates, occurring at times ti , i = 1, 2, …., N, with spreads BASi, and that t0 = T, tN+1 = T’, and BAS0 is based upon the quote that is outstanding at time T, i.e. the quote outstanding at the beginning of the minute. Quote from previous days are not used, so BAS0 does not exist prior to the first quote of the day. That means:**

**For the minute interval during which the first quote of the day occurs, the time-weighted spread is:**

**For minute intervals beginning subsequent to the first quote of the day, the time-weighted spread is:**

1. For each stock each day, compute average of the following stats with each stat calculated on minute level in a given day, i.e. compute daily average of minute stats: volume, spread, volatility (based on GARMAN-KLASS YANG-ZHANG model). When submit the result, just provide 2019-09-02 data. GKYZ model is described [here](http://www.todaysgroep.nl/media/236846/measuring_historic_volatility.pdf).

***One can use pandas resample to get the minute level data, and do the rest formula calculation:***

***ohlc = self.trades.trade\_price.resample(freq).agg(['first', 'max', 'min', 'last'])***

1. Compute daily median spread for each stock conditional on quote updates, and daily median spread for each stock conditional on trades (i.e. spread when trade happens; If there are multiple trade events between two quote updates, you count them as multiple observations in your sample). Spread is defined as ask\_price - bid\_price. Quote update is either price or size change on either bid or ask side. If it is only size change, you still treat the instant “spread” as one observation. For example:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Quote** | **recv\_time** | **symbol** | **bid** | **bid\_size** | **ask** | **ask\_size** |
| Q1 | 09:45:30.130 | XYZ | 10.00 | 200 | 10.02 | 300 |
| Q2 | 09:45:40.300 | XYZ | 10.00 | 100 | 10.01 | 200 |
| Q3 | 09:45:50.400 | XYZ | 10.00 | 200 | 10.02 | 300 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Trade** | **recv\_time** | **symbol** | **price** | **qty** |
| T1 | 09:45:35.220 | XYZ | 10.00 | 100 |
| T2 | 09:45:45.110 | XYZ | 10.01 | 100 |
| T3 | 09:45:47.310 | XYZ | 10.01 | 100 |
| T4 | 09:45:53.000 | XYZ | 10.00 | 100 |

In this case, the samples for median spread computation on **quote updates**, would be 2cent, 1cent, 2cent, total three observations (equivalent to three quote updates). The samples for median spread computation on **trade updates**, would be 2cent, 1cent, 1cent, 2cent, total four observations. For the middle two trades (T2/T3), when the trade happened, the quote was (10.00, 10.01) updated by 09:45:40.300,XYZ,10.00,100,10.01,200. You need to capture the instant quote/spread when the trade happens.

When submit the result, just provide 2019-09-02 data.

***One can use the pandas merge\_asof to get the instantaneous quote when the trade happens, to compute the spread on trades. Then the rest is simple:***

***d\_quote\_on\_trade = pd.merge\_asof(d\_trade, d\_quote, on='recv\_time', by='symbol')***

1. Compute daily t-stat of 1-minute/5-minute returns for each stock. 1-minute return is defined as log( / ). When submit the result, just provide 2019-09-02 data.

***One can use the resample method as in Q2, and compute the 1-min/5-min log return****.*

1. Open question: investigate whether there is a correlation between abnormal spread/volume with forward momentum. And how would you build a model to predict momentum symbols (predicting direction is not the target here)? You will need to define “abnormal” volume/spread and “momentum” symbol. One simple example for "abnormal" volume: current minute volume is x-times higher/lower than the previous 5-day average at the same minute, same as spread. Example for momentum symbol definition: t-stat of forward x-hours' 1-minute/5-minute return is high. You are free to come up with your own definition. You can also add other features to your model, such as volatility, lag return .... Use the 1-month of level1 data provided for your research.

***As many of you have found that using simple abnormal spread or volume variable doesn’t have strong predictive power on forward momentum in this dataset. We would normally try to add other variables or transformations, or do the bucket analysis (price/adv/volatility), or look for alternative data confirmation (such as news events, twitter volume spur …). The reason why we wanted to identify a momentum symbol early is to avoid trading mean-reversion signals on them, that would lose money in the trending market easily.***

1. Add a function to estimate intraday median spread from market open to a given time of the day for each stock, one conditional on quote updates and another one conditional on trade updates: We want to query the estimated median spread based on the quote/trade updates since the open. For example, at 10:13:45AM, we want to know the median spread of symbol XYZ given the quote updates from 9:30AM up to that point. Please note there is limited memory to store all the data. How would you optimize your code in terms of memory use? Remember it is to “estimate”. You want to be quick and

deterministic. We can tolerate some error, but need to provide the estimation uncertainty.

*This is the actual problem I had in the real production trading, and I used something similar to this histogram method, though implementation is slightly different in C++.*

*class Spread\_Hist:*

*hist={} #save number of spreads in each bin in a list with bin of range(i,i+1)\*0.01.*

*count=0 #total number of spreads*

*def \_\_init\_\_(self):*

*self.hist={}*

*self.count=0*

*def read\_next(self, spread): #When read a new spread data, update count, possible new bin and bin size*

*self.count+=1*

*Bin=int((spread/0.01) + 0.5)*

*if Bin in self.hist:*

*self.hist[Bin]+=1*

*else:*

*self.hist[Bin]=1*

*def median\_query(self,): # When client query the median, find the bin median must be in*

*Sum=0 # by count the sum of each bin size until half of total count.*

*bins=sorted(self.hist.keys())*

*for b in bins:*

*if Sum>=self.count/2:*

*break*

*else:*

*Sum+=self.hist[b]*

*return 0.01\*(b-1)*

1. We may add other data streams as well, such as book data, news data ... Would be nice to make the code interface easy to add other data sources.

You can use any programming language for this test (Python/C++ is preferred). For the data analysis questions (1,2,3,4,5), Python Pandas library may suit your needs well. The code’s efficiency and readability are also important for this test. When you submit the project, please include the source code, figures, and write-ups.