OrderBook Pressure Market Microstructure Prediction

Dimitri Chemla

June 2023

Contents

1 Introduction

The goal of this project is to analyse the micro-structure of the cryptocurrency spot market, and extract valuable information from the former. In a nutshell, this is done by quantifying buying and selling pressure in the orderbook, and finding patterns to predict the direction of price movements in the near future. That is, can we find certain behaviors in the orderbook, allowing an accurate enough prediction of the timing and direction of the underlying asset's price movement? Note that this particular model serves as one of the pillars for the development of the automated market making strategy Enigma is building. A correct prediction of price movement allows us to capitalise by adapting our quotes accordingly. For example, by bidding aggressively when our model predicts a substantial price increase in the next moments, and asking a low price to neutralise the position when we anticipate a price drop. More on risk neutrality in the Bollinger Bands documentation.
In the following sections, we explain the data engineering, statistical signals, logic and KPI's used throughout the project. The final sections cover our obtained preliminary results and possible improvements.

2 Data Processing

Our model requires an input of historical orderbook data, which we clean and transform into useable dataframes for analysis. The ordebook is created from 200 columns of data relating to bid/ask size and quantity, which equates to the 50 most competitive bids and asks. Each orderbook is then stored in a dataframe, which is sent as the value of a dictionary and the key assigned to this value is the unix timestamp at which the orderbook occurs. The raw data is quite badly formulated and so a few functions are necessary to reformat, clean and process the data. These functions are taken out of the actual obp.py program so as to not slow down the actual calculations and have to do the data processing everytime.

Once the raw data had been made useable, the final orderbook dictionary will have around 80 000 keys (translating to 1 orderbook per second), will look like this, with a depth of 50 lines,:

Processed OrderBook

{1678319321.0:

	Bid qty	Bid price	Ask price	Ask qty
0	0.02206	21720.86	21721.02	0.00066
1	0.00527	21720.83	21721.07	0.00374
2	0.00460	21720.79	21721.08	0.02302
3	0.00900	21720.77	21721.30	0.03561
4	0.00065	21720.73	21721.31	0.03581
5	0.01200	21720.61	21721.32	0.00070

Figure 1: Processed orderbook with a depth of 50.

3 Implementation

The actual calculation of the OBP is quite simple and given by the following formula:

$$OBP = \frac{\sum_{i}^{depth} (bid\_quantity_i - ask\_quantity_i)}{\sum_{i}^{depth} (bid\_quantity_i + ask\_quantity_i)}$$

The OBP is calculated for every second of the period that we are analysing. In our test case, this was usually 1 day, so around 85 000 seconds. Once we have a value for the OBP at a depth of n, which for testing was set to 15, but after backtesting we have found that there may be depths which are more profitable, more on this later.

Once we have this OBP_n, the real logic of the program can come in. We have defined certain regimes for the OBP_n values (OBP_n values can only lie between -1 & 1), to be able to see where the extreme values are and if they lead to a movement in price action. These regimes range from A-E and are defined as follows:

- *Regime A* : -0.75 $\geq$ OBP$_n$ $\geq$ -1
- *Regime B* : -0.3 $\geq$ OBP$_n$ $\geq$ -0.75
- *Regime C* : 0.3 $\geq$ OBP$_n$ $\geq$ -0.3
- *Regime D* : 0.75 $\geq$ OBP$_n$ $\geq$ 0.3
- *Regime E* : 1 $\geq$ OBP$_n$ $\geq$ 0.75

The regimes of interest to our application are A & E as these are considered to be extreme OBP values. This 'extremity' threshold is determined by us, however is a source of optimisation which will be covered in the backtesting section of this document.
We have a few functions used for this logic, wherein in first determines which regime the OBP lies, then checks the Bid/Ask spread to make sure it is over a certain threshold we have determined, also another source for optimisation.
There then is a final part of the logic which we will decide at a later date if we want to include or not, which is to check if the extreme OBP$_n$ are unique, or if they over a few seconds, are sequential. This is intersting to check as there are certain occasions where there is very heavy imbalance on one side of the orderbook and very little change in price in the coming seconds after this heavy imbalance, which is an indication that there may be a few 'spoof' orders and potentially some manipulation to the orderbook.

Below is the code for the calculation of the OBP$_n$ itself, as well as the regime matching, and the threshold conditon for Bid/Ask spread.

```
for unix, orderbook in orderbook_dict.items():
     obp_n = (((orderbook['Bid qty'][:depth_n]).sum() - (orderbook[
    'Ask qty'][:depth_n]).sum())) / ((orderbook['Bid qty'][:depth_n
    ]).sum() + (orderbook['Ask qty'][:depth_n]).sum())
     obp_arr.append(obp_n)
     unix_arr.append(unix)

compa_df = pd.DataFrame(obp_arr, columns=['obp_n'])
compa_df['unix'] = unix_arr
compa_df.index = compa_df['unix']
compa_df = compa_df.drop(columns={'unix'})

conditions = [((compa_df['obp_n'] >= -1.0) & (compa_df['obp_n'] <=
    -0.75)),((compa_df['obp_n'] >= -0.75) & (compa_df['obp_n'] <=
    -0.3)),((compa_df['obp_n'] >= -0.3) & (compa_df['obp_n'] <=
    0.3)),((compa_df['obp_n'] >= 0.3) & (compa_df['obp_n'] <= 0.75)
    ),((compa_df['obp_n'] >= 0.75) & (compa_df['obp_n'] <= 1.0))]
regimes = ['A','B','C','D','E']

compa_df['regimes'] = np.select(conditions, regimes)

for unix, orderbook in orderbook_dict.items():
  max_bs = orderbook['Bid qty'].nlargest(depth_n).max()
  max_as = orderbook['Ask qty'].nlargest(depth_n).max()
  compa_df.loc[compa_df.index == unix, 'max_bs'] = max_bs
```

```
 compa_df.loc[compa_df.index == unix, 'max_as'] = max_as

selected_A = compa_df[(compa_df['regimes'] == 'A') & ((compa_df['
    max_bs'] + threshold) < compa_df['max_as'])]
selected_E = compa_df[(compa_df['regimes'] == 'E') & (compa_df['
    max_bs'] > (compa_df['max_as'] + threshold))]
selected_df=pd.concat([selected_A, selected_E],ignore_index=False)
```

Listing 1: Application OBP

4 Initial Results

5 Optimisation & Results

6 Improvements & Next Steps