École Polytechnique Fédérale de Lausanne

FIN-525 - Financial Big Data

Performance of Market States Based Trading Strategies

Students:

- Zetong Liu (MSc. Applied Mathematics, KTH)
- Charles Gendreau (MSc. Applied Mathematics, EPFL)

Professor: Prof. Damien Challet
Teaching Assistant: Federico Baldi Lanfranchi

Academic Year 2023-2024

Contents

1 Abstract **2**

2 Data Processing & Parallelization **2**
2.1 Cleaning data and computer optimization 2
2.2 VWAP Trade and BBO prices 3
2.3 Code optimization 5

3 Trading Strategies **5**
3.1 Louvain clustering 5
3.2 Single market state strategy 5
3.3 Portfolio optimization 6
3.4 False discovery rate 6

4 Strategies Performances **7**
4.1 2007 and 2008 data 7
4.2 2010 data 9

5 Limitations and Possible Improvements **10**

A Appendix - Code snippets and Notebook **12**
A.1 Cleaning the data 12
A.2 Preprocessing of the data 13
A.3 Strategies 13

1 Abstract

This project explores the application of modern but computationally simple and efficient methods to build a trading strategy based on market states. In contrast to the fancy and popular machine learning algorithms that could perform very well but are computationally costly, we asses here the possibility to use simpler, but not less smart, algorithms with a very large amount of data.

In this report, we present the way we processed the large amount of financial intraday data available and how we can use it to find market states via clustering and build a trading strategy on it. We then discuss and implement various simple strategies and compare their performance.

A key aspect of the project was the handling of large files and data sets which pushed us to discover optimized computational methods as well as relevant Python libraries such as Dask, Vaex and the always-improving functionalities of Pandas.

The computations were done using Python 3.10.11. The code used is available in the attached .zip file, please refer to the associated README for further details.

2 Data Processing & Parallelization

For this project, we are provided with rough financial data from the S&P 100 index[1]. We have intraday trade and Best Bid and Offer (BBO) data files spanning from 2004 to 2008 for every day and every stock. We have data for 87 stocks, with quite surprisingly, no data for some important stocks like Microsoft (ticker: MSFT.O). This might be due to a loss of data in the copying process due to the old hard disk used to copy and store the data.

To test our strategy on more recent financial data (i.e. using financial data after the publication of the Louvain Clustering algorithm, see section 4.2), we also worked with 60-second intraday data of the S&P 500 from 2010. This dataset has data for 401 stocks from the 2010 S&P 500[2].

Figure 1 shows samples of the raw data available.

	xltime	trade-price	trade-volume	trade-stringflag	trade-rawflag
0	39455.604210	33.30	52900.0	uncategorized	[GEN]Open ...
1	39455.604219	33.25	100.0	uncategorized	[GEN]Low
2	39455.604219	33.30	100.0	uncategorized	
3	39455.604234	33.30	200.0	uncategorized	
4	39455.604238	33.29	300.0	uncategorized	

(a) Sample 2008 raw trade data

	xltime	bid-price	bid-volume	ask-price	ask-volume
0	39455.604208	33.26	10	33.31	1
1	39455.604208	33.25	5	33.31	1
2	39455.604209	33.25	5	33.30	3
3	39455.604211	33.25	4	33.30	3
4	39455.604211	33.25	4	33.30	2

(b) Sample 2008 raw bbo data

	index	X.Open	X.High	X.Low	X.Close
0	2010-01-12 09:31:00-05:00	16.72	1467.0	16.69	461.0
1	2010-01-12 09:32:00-05:00	16.69	1191.0	16.67	7.0
2	2010-01-12 09:33:00-05:00	16.72	417.0	16.71	131.0
3	2010-01-12 09:34:00-05:00	16.72	481.0	16.72	167.0
4	2010-01-12 09:35:00-05:00	16.71	661.0	16.66	266.0

(c) Sample 2010 raw data for BAC

Figure 1: Samples of the raw data

2.1 Cleaning data and computer optimization

We first had to clean the data which we did for the 2007-2008 data by transforming the given time into an appropriate format taking into consideration the timezone of the source of the data: the New York Stock Exchange (UTC+5:00). We then merged all the trade and bbo files for all the tickers and each month. The merging was performed parallelizing the computations with the Python library Dask. We first loaded and merged the trade and bbo data for all the days of a given ticker and parallelised the computation for the 87 tickers. This way of parallelizing was chosen after various tests as parallelization did not always improve the computations. This is probably due to the handling of memory in the different CPUs. Indeed, when parallelizing the computer had to perform the concatenation of the dataframes for every ticker which were computed in parallel on different CPUs. Perhaps, this should require further investigation and computer knowledge, the fact that the data was (partly) stored in different CPUs

[1] Data available in the provided folder sp100_2004-8

[2] Data available in the provided Switch Drive

increased the total required time. Overall, the time gained to clean all the 12 GB considered for the 2007 and 2008 data required. The table below shows the times required for the cleaning and saving of the clean file for February 2007[3]. At the end of the cleaning process, we obtained parquet files for every considered months with the trade and bbo data for every ticker, each cleaned file was around 500MB. The fragmentation over each month ensured the handling of possible memory issues. Table 2 shows a comparison of the time with and without parallelization.

Method	Time Required
Parallelized	251 seconds
Non-Parallelized	333 seconds

Table 1: Comparison of 2007-02 trade and bbo loading, cleaning and saving the data time: Parallelized vs. Non-Parallelized

For the 2010 data, we noted that only the high and close prices did not always have a clear meaning as seen in Figure 1c. We therefore only kept the open price.

Data Preprocessing

Unless stated otherwise, we refer to the 2007 and 2008 data.

2.2 VWAP Trade and BBO prices

To apply the clustering algorithm, we had to reduce the data to a reasonable trading frequency with enough data over every trading period. We chose to consider the minute-by-minute Volume-Weighted Average trade Prices (VWAP) to find the market states with the clustering algorithm (see section 3.1). This process gave us data for almost every minute of the NYSE trading periods of 2007 and 2008 (from 9:30 a.m. to 4:00 p.m. every working day). Knowing that every working day the US market is open for 390 minutes we can choose whether a day contains enough observations or not.

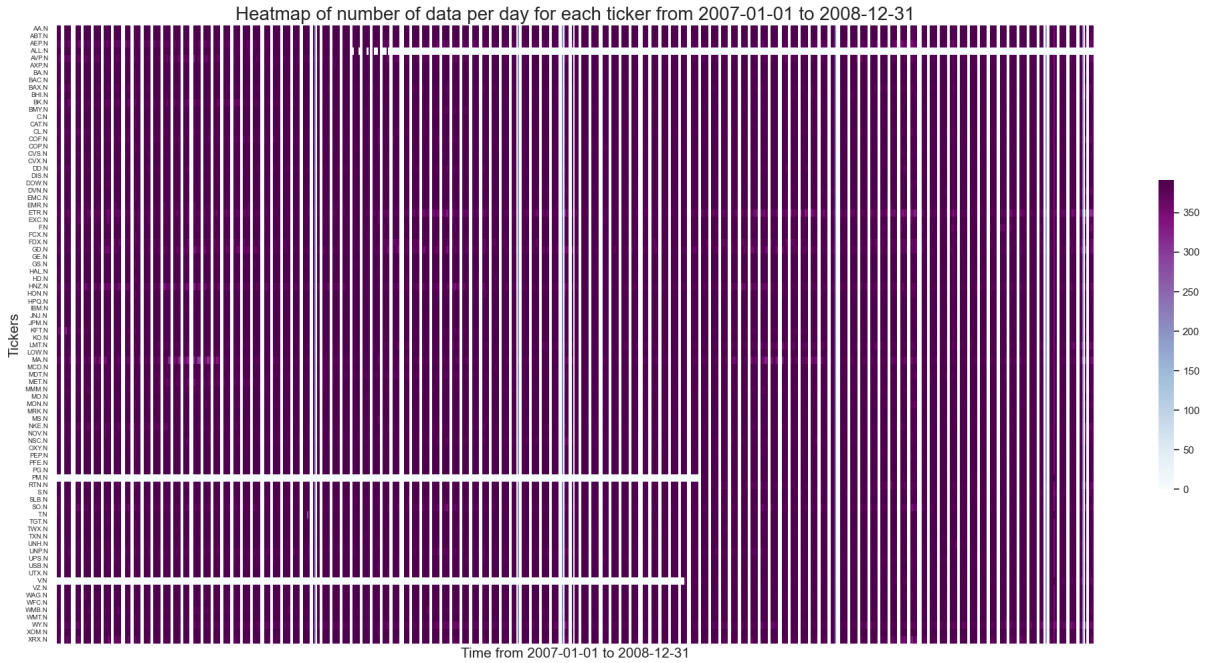


Figure 2: Heatmap of data availability for each ticker (2007-2008)

[3]The machine used has an Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz under Windows 10 and with 5.9/8 GB RAM available for both tests.

For the evaluation of the strategy (in 2007-2008) given the market states, we used the BBO data with the bid and ask prices to take into account the spread-cost, giving a more realistic evaluation of the strategy (3). For every minute and every stock, we considered the last available bid and ask prices.

Figure 2 shows a heatmap of the number of trades per day from January 2007 to December 2008 for every ticker. First, we observed that 3 tickers (namely ALL.N, PM.N, V.N) do not have data for the whole time range (probably these stocks were not part of the S&P100 for the whole time), so we decided not to consider them. We observe a certain homogeneity in the number of data available for each ticker. This is confirmed by the fact that the average of the number of minute-trading prices (the VWAP over the considered minute) per day ranges from 373 to 387 (over 390 possible minute-trading prices per day). The minimal number of minute trade prices per day over all the considered tickers is as high as 150, we therefore chose to consider all the days.

We applied the same procedure to the 2010 data whose data heatmap is shown in Figure 3. We observe that many stocks have a lack of data either over a given period or for the whole time. We therefore decided to keep only tickers with more than 90,000 data prices over the whole year which roughly corresponds to 93% of the possible observations. This resulted in considering only 366 stocks with a minimum over all days and all stocks of 198 observations per day.

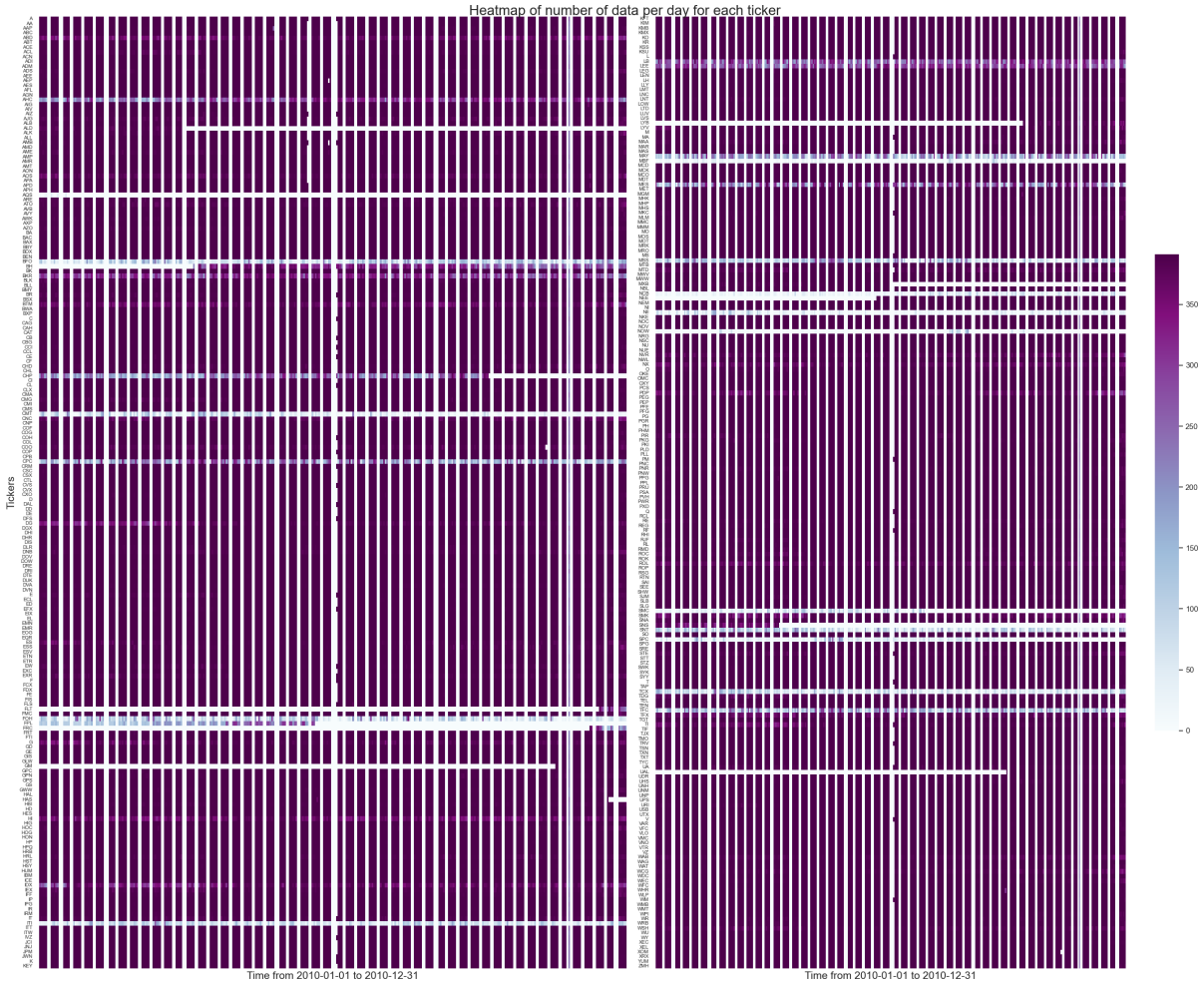


Figure 3: Heatmap of data availability for each ticker (2010)

2.3 Code optimization

The preprocessing of the data implied handling large dataframes which could cause large computation times and memory issues. For each month, we had cleaned files of approximately 500MB. To handle memory issues, we fragmented the data monthly however we still faced issues using Pandas. We therefore used Vaex, a Python library for lazy visualization and exploration of data frames, instead of Pandas as often as possible in the preprocessing of the data. This allowed us to perform the computations much faster. Table 2 presents the difference in the performance of the libraries for the preprocessing of the 2010 data[4].

Library	Time Required
Pandas	250 min
Vaex	26 min

Table 2: Comparison of time required for the preprocessing of the 2010 data

3 Trading Strategies

We now discuss the implementation of trading strategies based on market states.

3.1 Louvain clustering

To implement the strategy, we first need to find market states using a clustering algorithm.

Clustering is an unsupervised machine-learning algorithm that can separate data into distinct clusters without prior knowledge of data labels. The Louvain method is a clustering algorithm developed by Prof. Blondel and published in 2008 [1]. It has the particularity of not requiring a prior assumption on the number of clusters and of running in time $O(n \log n)$.

As described in [1, p. 2], one needs to optimize the modularity which is a measurement of the similarities within two clusters. For a weighted graph, the modularity is given by:

$$Q = \frac{1}{2m} \sum_{ij} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta\left(c_i, c_j\right),$$

where $A_{i,j}$ represents the edge between node i and j, k_i and k_j are the sum of the weights of the edges attached to node i and j respectively, m is the sum of all the edge weights in the graph, c_i and c_j are the communities of nodes and

$$\delta_{(x,y)} = \begin{cases} 1, & \text{if } x = y \\ 0, & \text{if } x \neq y \end{cases}$$

Clustering the Louvain way extracts distinct clusters from large networks of nodes. When the nodes are tickers, clusters represent asset sectors. When the nodes are timestamps, clusters represent market states.
In our simulations, we obtained the market states for each minute of years 2007 and 2008 applying the Louvain algorithm provided by Prof. Challet to the VWAP data. For 2010, we applied the same algorithm to the available minute-by-minute cleaned data.

3.2 Single market state strategy

The first strategy considered is the single market state. This strategy aims to predict the return of the next timestamp of the calibration window. By iterating over the calibration window, the strategy constantly adapts to a fast-changing market and gives one prediction of the return for the next timestamp outside of the calibration window. At time t_1, applying Louvain clustering on timestamps between t_0 and

[4]For Pandas, the time refers to an estimate: 20 minutes were required to complete 8% of the preprocessing of the 2010 data.

t_1 permits us to get the market state at t_1, i.e.,μ_{t_1}. Then one can look back and find all $\tilde{t} \in [t_0, t_1 - 1]$ such that $\mu_{\tilde{t}} = \mu_{t_1}$. For each asset i, we compute the performance measure of $(r_{i,\tilde{t}+1})$. This is an indicator of whether an asset will perform well. For this project, we computed the average, median and Hodge-Lehman (HL) pseudo-median returns[2]. The sign of the performance measure then tells us how to upgrade the portfolio: for asset i, x_{i,t_1}, the direction of the rebalancing from t_1 to $t_1 + 1$, is set to be long for a positive sign $(+)$, short for a negative sign $(-)$, or hold (0).

Let $\Delta_{t_1} = \{\tilde{t} \in [t_0, t_1 - 1] : \mu_{\tilde{t}} = \mu_{t_1}\}$, we have:

$$x_{i,t_1} = \text{sign}\left(\frac{\sum\limits_{t \in \Delta_{t_1}} r_{i,t+1}}{\#\Delta_{t_1}}\right) \tag{Mean}$$

$$x_{i,t_1} = \text{sign}\left(\text{Med}\left(\{r_{i,t+1} : t \in \Delta_{t_1}\}\right)\right) \tag{Median}$$

$$x_{i,t_1} = \text{sign}\left(\text{Med}\left(\left\{\frac{r_{i,t+1} + r_{i,s+1}}{2}\right\}_{\substack{s>t \\ s,t \in \Delta_{t_1}}}\right)\right). \tag{HL Pseudo Median}$$

3.3 Portfolio optimization

Until now, for simplicity, we gave equal weights to each asset. We now discuss how portfolio optimization affects the performance of our strategy.

Denoting the covariance matrix of returns by Ω and w the allocation weight on each asset, the minimum variance portfolio is the w satisfying [3]

$$\min_w w^T \Omega w \quad \text{s.t.} \quad \sum_i w_i = 1$$

which is given by

$$w^* = \frac{\Omega^{-1} e}{e^T \Omega^{-1} e}, \quad e = (1, 1, ..., 1)^T.$$

For our strategy, after obtaining the position x_t, we find the minimum variance portfolio by solving

$$\min_w w^T \Omega w \quad \text{s.t.} \quad w^T x_{t_1} = 1.$$

The solution is

$$w^* = \frac{\Sigma^{-1} x_{t_1}}{x_{t_1}^T \Sigma^{-1} x_{t_1}}$$

where $x_{t_1} = (x_{1,t_1}, x_{2,t_1}, ..., x_{N,t_1})^T$ is the direction of trades for all assets from t_1 to $t_1 + 1$.
Since we do clustering over dates, it is common that $T < N$. In this case, the covariance matrix is not invertible. So in a high-dimensional regime, cleaning methods of the covariance matrix need to be incorporated. In this project, we choose Bootstrap Average Linkage Hierarchical Clustering (BAHC)[4], which can optimize a portfolio well with $T < N$ [3].

3.4 False discovery rate

To reduce the fraction of false-positive returns prediction, we tried to control the false discovery rate. Within a calibration window $[t_0, t_1]$, one could perform multiple T-tests on the average return of the current state for each asset using the hypothesis:

$$H_0 : \mathbb{E}\left(r_{i,t+1} \mid \mu_t = \mu_{t_1}\right) = 0 \quad \text{against} \quad H_1 : \mathbb{E}\left(r_{i,t+1} \mid \mu_t = \mu_{t_1}\right) \neq 0.$$

Firstly, we calculated the T-statistics,

$$S_{i,\mu} = \frac{\mathbb{E}\left(r_{i,t+1} \mid \mu_t = \mu_{t_1}\right)}{\sqrt{\text{var}\left(r_{i,t+1} \mid \mu_t = \mu_{t_1}\right)}} \sqrt{n_{\mu_{t_1}}}$$

where $n_{\mu_{t_1}}$ is the number of timestamps labelled with μ_{t_1} within $[t_0, t_1 - 1]$. Then, we computed the p-values for each T-statistics and used the Benjamini–Hochberg procedure[5] to perform asset selection. However, since minute returns were close to zero, it barely rejected the null hypothesis and thus none of the assets was chosen. We therefore decided not to include the false discovery rate method in our strategy.

4 Strategies Performances

We now present the results of our strategies.

4.1 2007 and 2008 data

Given a 2007 dataset of minute-by-minute prices, we kept the values of the tickers with more than 90,000 observations and we forward-filled the remaining data. We then discarded rows with at least one null value and finally calculated the linear return on this filtered data. After the process, the remaining data contained $N = 81$ tickers. Therefore, we applied the Louvain clustering algorithm with calibration window length $T = \lfloor N/3 \rfloor$, following the usual rule of thumb [1, 3].

We chose a time period from $t = 1000$ to $t = 1000 + T - 1$, and we obtained $R_{N \times T}$ from the return matrix of the whole two years. We then calculated the correlation matrix and then used a function from Python package *bahc* to calculate the BAHC-filtered matrix(average of the 100 HCAL-filtered matrices). As shown in 4,

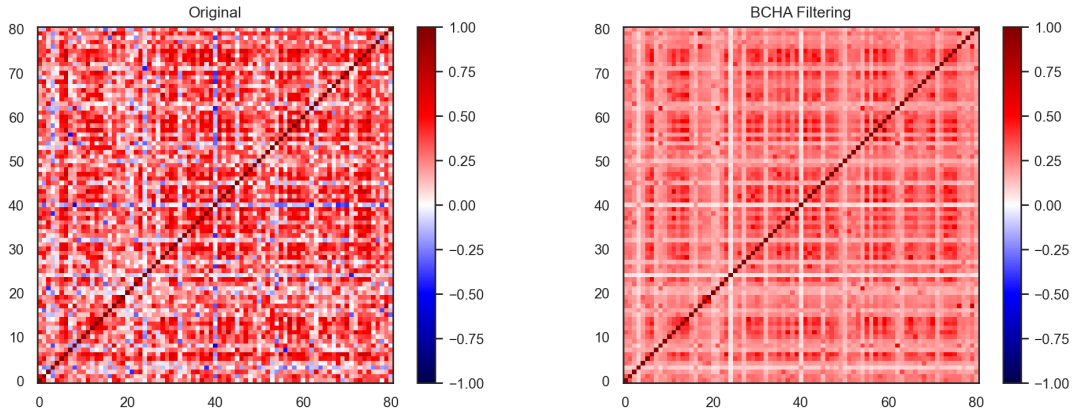


Figure 4: Correlation matrix of stocks returns (Original and after BCHA filtering)

The number of clusters presents certain patterns over time and most of the time there were only three clusters as seen in Figure 5:

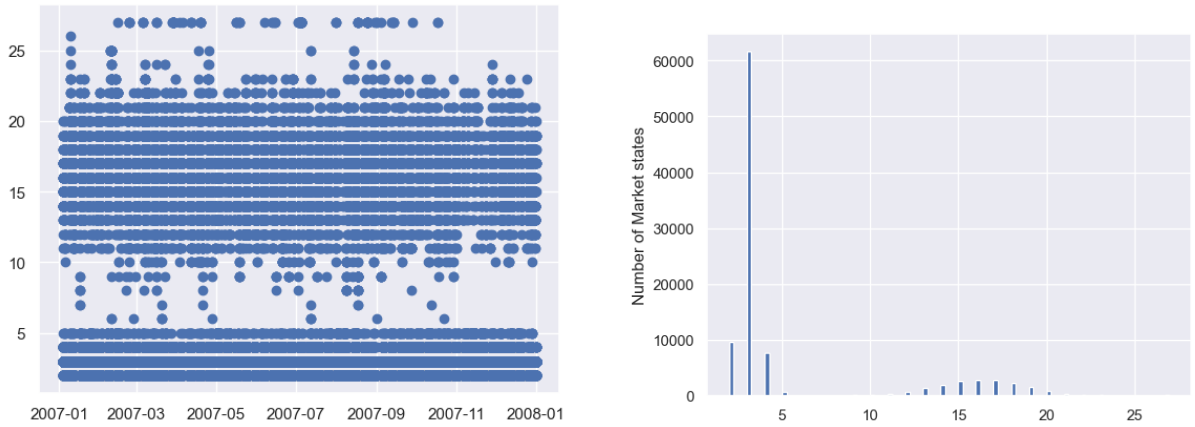


Figure 5: Scatter plot of the number of market states (Left) and Histogram of the number of market states (Right)

The similarity between two consecutive clusters can be measured by the Adjusted Rand Index [6]. For example, if we have two clusters $C1 = \{X_1, X_2, ..., X_r\}$ and $C2 = \{Y_1, Y_2, ..., Y_r\}$, then

$$\mathbf{ARI} = \frac{RI - E}{1 - E}$$

where

$$RI = (a + b)/\binom{N}{2},$$

$$E = \left[\sum_{i=1}^{r}\binom{a_i}{2}\sum_{j=1}^{s}\binom{b_j}{2}\right]/\binom{n}{2},$$

and a is the number of pairs of samples that are in the same cluster in both $C1$ and $C2$ while b is the number of pairs of samples that are in different clusters in $C1$ and $C2$. Finally, a_i is the number of elements in X_i and b_j is the number of elements in Y_j.
From Figure 6, one can observe that ARI is mainly distributed around 0. So it shows that the clustering of two consecutive calibration windows is not very stable, showcasing the non-stationarity of the market.

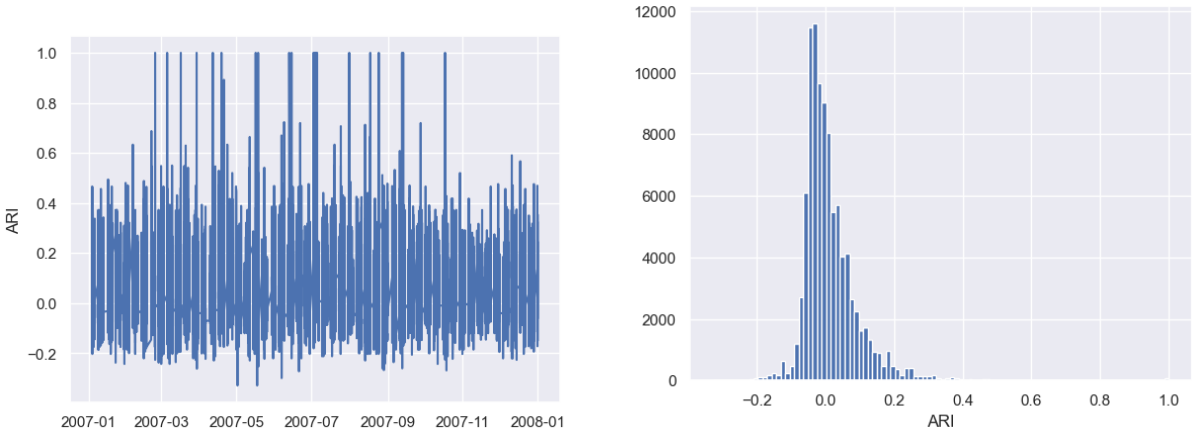


Figure 6: Scatter plot of the number of market states (Left) and Histogram of number of the market states (Right)

For the minute resolution data of 2007, we run the market state strategy with the three different performance measures (Mean, Median, HL Pseudo Median). We observe a good performance for the years 2007 to 2008 and only 2007.

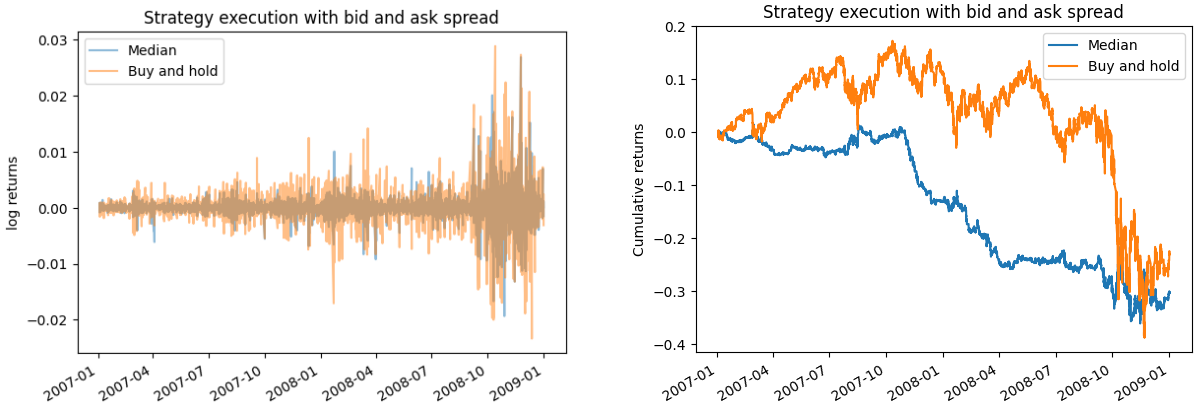
(a) 2007 - 2008 (b) 2007

Figure 7: Cumulative returns of the strategies using VWAP

The strategy with Hodge-Lehman Pseudo Median performs best among all the three measures but it's worth noticing that the Louvain method was published in 2008, so there is technology look-ahead bias for this backtest. What is more useful is to backtest the data on S&P 500 in 2010.

The single-state strategy would give new positions every minute and we buy and sell at the volume-weighted average price of the following minute. To be more realistic, we can execute the strategy with the last best bid/ask price of a given minute, namely selling at the best bid price and buying at the best ask price.

Since so far we backtested strategies with the VWAP, now we look to the performance of our strategies when executed with the best bid and ask prices (while still extracting signal from VWAP). Figure 8 shows that the strategy barely outperforms the buy-and-hold strategy during the 2008 financial crisis. This means that the signal of the predicted VWAP can't be applied to the real-time execution of strategies.



(a) Log returns (b) Cumulative returns

Figure 8: Performance of the strategy with the median signal considering the bid/ask spread

4.2 2010 data

Figure 9 corroborates our hypothesis that look-ahead technology bias is detrimental to backtesting. In 2010, after the Louvain method was invented, the trading strategy performed badly. This could however be tested on larger data samples and other financial periods.

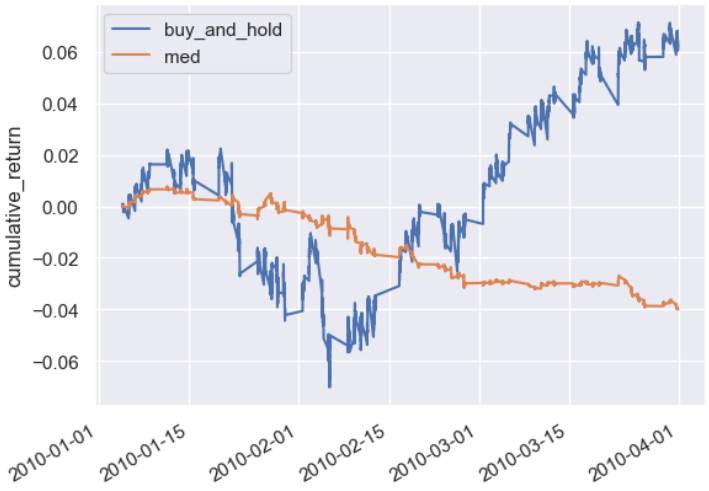


Figure 9: Cumulative returns of the median strategy in 2010-Q1

5 Limitations and Possible Improvements

Overall, the strategy only uses the current market state to predict future returns and it turns out that it does not outperform the passive market move after considering the bid and ask spread. To improve our strategies, we could use Marsili-Giada clustering[7] as an alternative to Louvian clustering and test if it improves the performance algorithm. Moreover, one could also try to design a more complicated strategy which would take into consideration a time series of market states instead of only the current one. Finally, one could try to find market states with mid prices instead of VWAP and then apply the strategy.

It would be interesting to further investigate how the publishing of the Louvain or Marsili and Giada algorithms affected their overall performance. Indeed, it is well known that performant strategies rapidly become deprecated creating a very competitive and stimulating environment in the world of quantitative research.

Finally, one could work in the optimisation of the computations both on the mathematical and the computer sides to always reduce the required time and the energy consumed.

References

[1] Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008:P10008, 2008.

[2] E.L. Lehmann and H.J.M. D'Abrera. *Nonparametrics: Statistical Methods Based on Ranks*. Prentice Hall, 1998.

[3] Prof. Damien Challet. EPFL FIN-525 Financial Big Data Lecture Slides (Autumn 2023).

[4] Christian Bongiorno and Damien Challet. Covariance matrix filtering with bootstrapped hierarchies. *PLOS ONE*, 16(1):e0245092, January 2021.

[5] Yoav Benjamini and Yosef Hochberg. Controlling the false discovery rate: A practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society: Series B (Methodological)*, 57(1):289–300, 1995.

[6] Adjusted rand index (ari), 2024. Accessed: 2024-02-01.

[7] Lorenzo Giada and Matteo Marsili. Algorithms of maximum likelihood data clustering with applications. *arXiv preprint arXiv:cond-mat/0204202*, 2002. Available at `https://arxiv.org/abs/cond-mat/0204202`.

A Appendix - Code snippets and Notebook

Hereafter are some relevant functions used in the notebooks which can be found in the zip folder attached to the submission. All the functions are available as well in the zip file.

A.1 Cleaning the data

Hereafter an example of the function used to save and clean the data in a parallelized way using Dask.

```python
#Delayed function to load and clean data for all selected dates for a given ticker. The other called functions respect a similar structure and can be found in upload_data.py

@dask.delayed
def load_all_dates(ticker, start_date = pd.to_datetime('2004-01-01'), end_date =
                   pd.to_datetime('2023-12-31'),
                   country = "US", dirBase="data/raw/equities/",suffix="parquet",
                   suffix_save=None, dirSaveBase="data/clean/equities/events",
                   saveOnly=False, doSave=False):
    """
    Loads all data from start_date to end_date for a given ticker
    date: date object with format 'YYYY-MM-DD'
    Returns: Pandas dataframe with all the events all the days
    """
    all_events = pd.DataFrame()
    all_dates = get_all_dates(ticker, dirBase = dirBase, country = country)
    all_dates = [date for date in all_dates if date >= start_date and date <= end_date]
    allpromises=[load_merge_trade_bbo(ticker, date, country = country, dirBase = dirBase, suffix = suffix) for date in all_dates]
    #remove None values
    allpromises = [x for x in allpromises if x is not None]
    try:
        all_events = pd.concat(allpromises, axis = 0)
    except:
        print("No data in the given period for ticker " + ticker)
        return None
    #adding info to dataframe
    all_events['ticker'] = ticker

    ## Saving file ##

    return all_events

#Main function to clean all the data
def load_all(start_date = pd.to_datetime('2004-01-01'), end_date =
            pd.to_datetime('2023-12-31'),
            tickers:list = None,
            show_parallelization_structure=False,
            country = "US", dirBase="data/raw/equities/", suffix="parquet",
            suffix_save=None, dirSaveBase="data/clean/equities/events",
            saveOnly=False, doSave=False):
    """
    Loads all data from start_date to end_date

    If tickers is None, loads all the data available
    If tickers is a list of tickers, loads only those tickers
    If show_parallelization_structure is True, saves figure of the parallelization structure of the computation

    Returns: Pandas dataframe with all the events for all tickers and the given days
    """
    if tickers is None:
        all_tickers = get_all_tickers(dirBase = dirBase, country = country)
    else:
        all_tickers = tickers

    allpromises = [load_all_dates(ticker, start_date = start_date, end_date = end_date, country = country, dirBase = dirBase, suffix = suffix) for ticker in all_tickers]

```

```
    all_events_list = dask.compute(*allpromises)
    #remove None values
    all_events_list = [x for x in all_events_list if x is not None]
    all_events = pd.concat(all_events_list, axis = 0)

    ## Saving file code ##

    return all_events
```

A.2 Preprocessing of the data

Hereafter an example of the function used to preprocess the clean data using Vaex

```
#Function to save the clean file with VWAP
def save_vwap_trade(dirBase = "E:/FinancialData/sp100_2004-8/clean/events/US",
    dirSaveBase = "E:/FinancialData/sp100_2004-8/clean/vwap/trade"):
    """
    Calculates and saves vwap (Volume Weighted Average Price) of the trade price over a
    1min tick) to parquet file.
    Requires vaex library.
    """
    tick = '1min'
    #extracting all file names:
    file_pattern = os.path.join(dirBase, '*-to-*-events.parquet')
    file_list = glob.glob(file_pattern)
    #loop through files:
    print('Calculating and saving {}-vwap for all available clean trade files'.format(
    tick))
    for file in tqdm.tqdm(file_list):
        #loading file
        alldata = vaex.open(file)
        #calculating vwap
        df_trade = alldata[~alldata['trade_price'].isna()] # we keep only the "trade"
    events.

        df_trade['xltime'] = df_trade['xltime'].astype('datetime64')
        df_trade['minute'] = df_trade['xltime'].dt.strftime('%Y-%m-%d %H:%M')
        #group by minute and ticker, then calculate VWAP
        vwap_df = df_trade.groupby(by=['minute', 'ticker'],
                                   agg={'vwap': vaex.agg.sum(df_trade['trade_price'] *
    df_trade['trade_volume'])
                                        / vaex.agg.sum(df_trade['trade_volume'])
                                        })
        #save to parquet file
        vwap_file_name = os.path.join(dirSaveBase,
                                      re.sub(r'events', 'vwap-trade-{}'.format(tick), os
    .path.basename(file)))
        vwap_df.export_parquet(vwap_file_name,use_deprecated_int96_timestamps=True)
```

A.3 Strategies

Hereafter the functions used to implement the strategies.

```
def market_state_strat(r): #matrix of log returns
    T=int(np.floor(r.shape[1]/3))
    ret=[1]
    for t in range(T+1,r.shape[0]):
        R = r.iloc[t-T:t]
        DF = LouvainCorrelationClustering(R.T)
        pre_state=DF.iloc[T-1][0]
        I = DF[DF[0]==pre_state].index.tolist()
        my_list = [x+1 for x in I[:-1]]
        ar=R.iloc[my_list].mean(axis=0)
        pos = np.sign(ar.values)
        ret.append(np.dot(pos, np.exp(r.iloc[t].values)-1)/len(pos)+1)

    df_chopped = r.iloc[T:]
    df_chopped['rolling_ret']=ret
```

```
    df_chopped['cumulative_perf']=df_chopped['rolling_ret'].cumprod()
    df_chopped['cumulative_perf'].plot()
    plt.xlabel('time')
    plt.ylabel('USD')
    plt.title('Cumulative_performance')

def market_state_strat_upgraded(r, clustering='Louvian', trend_measure='med_HL', seed=10): #matrix of log returns
    np.random.seed(seed)

    T=int(np.floor(r.shape[1]/3))
    ret=[0]
    pos=np.sign(np.zeros(r.shape[1]))
    for t in range(T+1,r.shape[0]):
        R = r.iloc[t-T:t]
        if clustering=='Louvian':
            DF = LouvainCorrelationClustering(R.T)
            cur_state=DF.iloc[-1][0]
        # else:
        #     clusters = aggregate_clusters(C, only_log_likelihood_improving_merges=False)
        I = DF[DF[0]==cur_state].index.tolist()
        my_list = [x+1 for x in I[:-1]]
        if my_list:
            if trend_measure=='med_HL':
                ar=med_HL(R.iloc[my_list])
            elif trend_measure=='med':
                ar=R.iloc[my_list].median(axis=0)
            else:
                ar=R.iloc[my_list].mean(axis=0)
            pos = np.sign(ar.values)
        ret.append(np.dot(pos, np.exp(r.iloc[t].values)-1)/len(pos))

    all_ret=pd.DataFrame({'Strat_A':ret}, index=r.index[T:])
    all_ret['BH']=np.concatenate(([0], np.dot(np.exp(r.iloc[T+1:])-1, np.ones(len(pos)))/len(pos)))
    all_ret['Strat_A_perf']=all_ret['Strat_A']+1.0
    all_ret['BH_perf']=all_ret['BH']+1.0

    return all_ret

def strat_eval(all_ret, m):
    #robust sharp ratio
    all_ret['Strat_A_perf'].cumprod().plot(label=m)
    all_ret['BH_perf'].cumprod().plot(label='buy_and_hold')
    sharpe_ratio=_calc_sharpe(all_ret['Strat_A'])
    plt.xlabel('time')
    plt.ylabel('USD')
    plt.title(f'Cumulative_performance for {m}, SR={sharpe_ratio}')
    plt.legend()
    plt.show()

def _calc_sharpe(returns):
    return returns.mean() / returns.std() * np.sqrt(252*390)

def med_HL(data):
    n = data.shape[0]
    if n>1:
        pair_list=[]
        for i in range(n):
            for j in range(i+1,n):
                pair_list.append((data.iloc[i] + data.iloc[j]) / 2)
        pairwise_averages = pd.DataFrame(pair_list)
        return pairwise_averages.median(axis=0)
    else:
        return data.median(axis=0)

def FDR(data, alpha=0.05, null_mean=0):
    pvals=[]
    for ticker in data.columns:
```

```
        _, p_value = stats.ttest_1samp(data[ticker], null_mean)
        pvals.append(p_value)
    rej, _,_,_ = multitest.multipletests(pvals, alpha=alpha, method='fdr_bh')

    return rej
```