Stock Trading with Machine Learning

Daniel Hunegnaw , Kai Luo | EE P 596 Advanced Machine Learning | 02-27-22

UW EE 596: Stock Trading with Machine Learning

Contents

UW EE 596: Stock Trading with Machine Learning

Introduction

In this report, we will present the implementation of the baseline cross moving average strategy for stock trading.

Project Structure

The project is implemented in python using visual studio code. A private repository is created in github. We have planned to make the repository public after the project is completed. Fig 1 below show the current structure of the project. Below is provided a brief description of each folder and file.

Folder structure

`stock-ml-mini-project:` The root folder of the project

`docs`:- where documentations and reports of the project reside

`docs-report -final.docx:-` the final project report document

`src` :- where the source code of the project resides

`TradingStrategy.py:-` class which contains all the functionality to do paper trading with trained model

`program.ipynb:-` a jupyter file for running model training and back testing. To run the project using the program.ipynb, you need to provide values for **Api_key** and **Secret_key** variables.

`PaperTrader.py:-` class/a job for running paper trading everyday when the market is open

`start.py` : this does the same task as in `program.ipynb` but you can use this to run it from the command line using the command as in the Fig 2

`Common:-` Is a folder that contains common utility classes

`ApiClient.py:-` a utility class used to call the ALPACA API

`DatetimeUtilty.py:-` used to check if the market is open for trading for a given datetime

`MovingAverageCalculator.py`: contains functions for calculating Simple and Exponential Moving Average

`MA`:- Contains class files related to moving average functionality

`BaseStrategy.py`: a base class which currently contains functions for:

- plotting dataframe data, moving averages, buy and sell signals
- for generating buy and sell signals based on crossing moving averages

`ExponentialMovingAverageStrategy.py`: a class which contains functions for

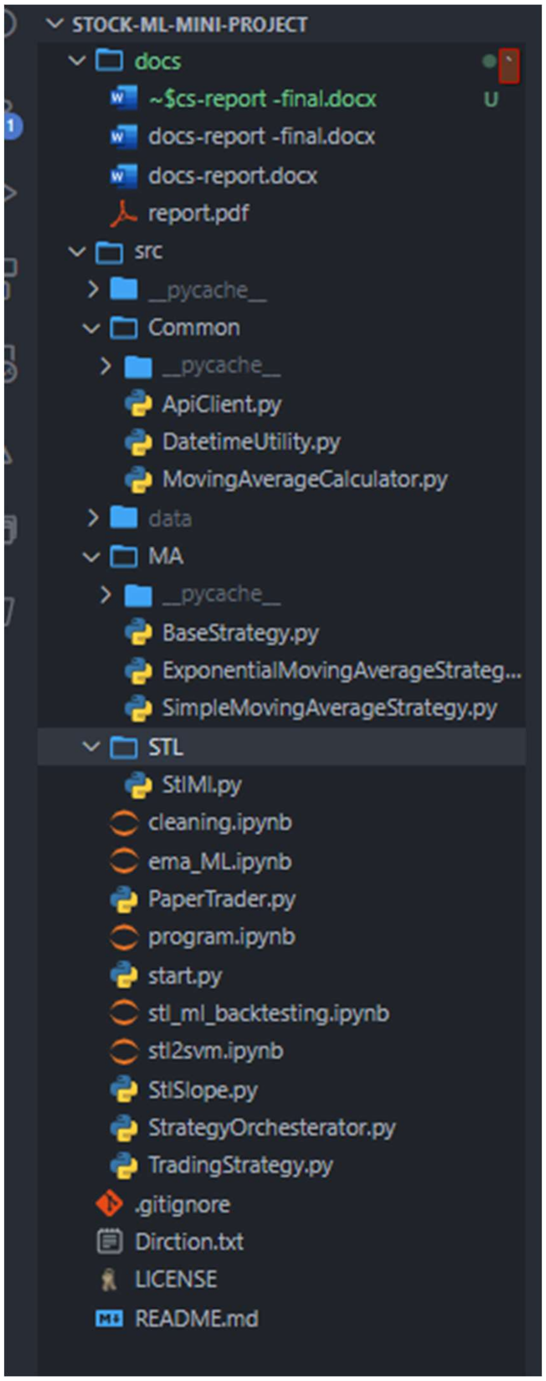
- generating data for exponential crossing moving average based strategy
- creating features and labels based on local min and max of the moving average
- training models
- backtesting models

`SimpleMovingAverageStrategy.py`: contains function for generating data for crossing simple moving average-based strategy

`STL`:- Contains class files related to STL functionality

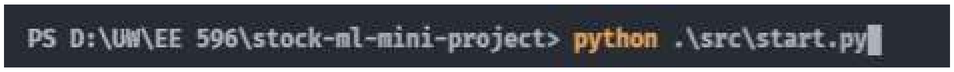
`StlMl.py`: a class which currently contains functions for:

- generating features and labels based on STL from raw
- train STL ML training model
- predict models for every buy, sell and hold possible
- regulate models for sanitize extra buy and sell and prevention of excessive losses
- produce profit model
- plot model for close price and prediction
- backtesting model



```
STOCK-ML-MINI-PROJECT
  docs
    ~$cs-report -final.docx        U
    docs-report -final.docx
    docs-report.docx
    report.pdf
  src
    __pycache__
    Common
      __pycache__
      ApiClient.py
      DatetimeUtility.py
      MovingAverageCalculator.py
    data
    MA
      __pycache__
      BaseStrategy.py
      ExponentialMovingAverageStrateg...
      SimpleMovingAverageStrategy.py
    STL
      StlMl.py
    cleaning.ipynb
    ema_ML.ipynb
    PaperTrader.py
    program.ipynb
    start.py
    stl_ml_backtesting.ipynb
    stl2svm.ipynb
    StlSlope.py
    StrategyOrchesterator.py
    TradingStrategy.py
  .gitignore
  Dirction.txt
  LICENSE
  README.md
```

Fig 1. Project structure



```
PS D:\UW\EE 596\stock-ml-mini-project> python .\src\start.py
```

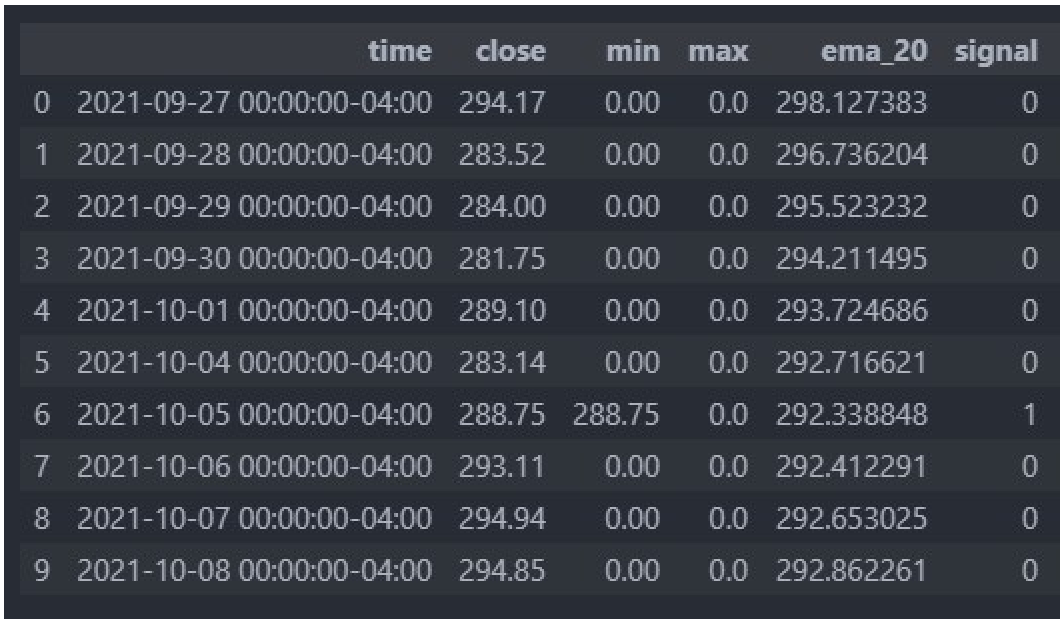
Fig 2. Running the project from command line

Stock ML Model with EMA

Feature and Label Extraction

The following are the steps used to create features and labels

1. Calculated the 20-days exponential moving average
2. Based on the moving average local *min* and *max* are calculated with sliding window of 4 (default) on the close price
3. The local *min* and *max* of values are used to generate the label the *signal* column. A signal value of 1 means 'buy' and a signal value of -1 means 'sell'
4. Then the data is divided into int train and test (7 months of data for train, and the last 5 months of data for backtesting
5. Modeling training is done on the data which looks like as shown in Fig 3.



	time	close	min	max	ema_20	signal
0	2021-09-27 00:00:00-04:00	294.17	0.00	0.0	298.127383	0
1	2021-09-28 00:00:00-04:00	283.52	0.00	0.0	296.736204	0
2	2021-09-29 00:00:00-04:00	284.00	0.00	0.0	295.523232	0
3	2021-09-30 00:00:00-04:00	281.75	0.00	0.0	294.211495	0
4	2021-10-01 00:00:00-04:00	289.10	0.00	0.0	293.724686	0
5	2021-10-04 00:00:00-04:00	283.14	0.00	0.0	292.716621	0
6	2021-10-05 00:00:00-04:00	288.75	288.75	0.0	292.338848	1
7	2021-10-06 00:00:00-04:00	293.11	0.00	0.0	292.412291	0
8	2021-10-07 00:00:00-04:00	294.94	0.00	0.0	292.653025	0
9	2021-10-08 00:00:00-04:00	294.85	0.00	0.0	292.862261	0

Fig 3. Except of the dataframe with features columns Min, Max, and label Signal

Performance Evaluation backtest

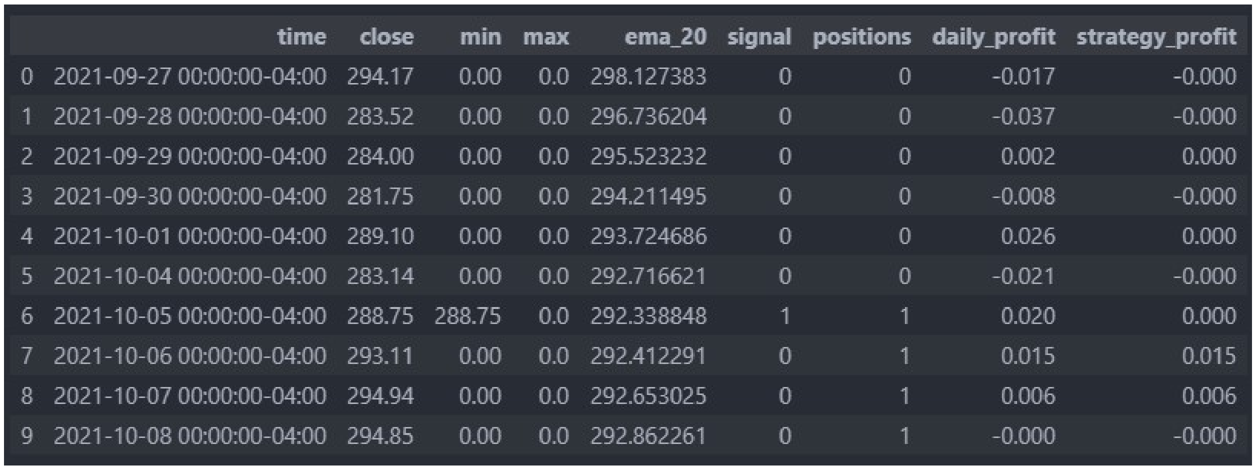
The data selected for this project includes one year stock data of FB, MSFT, NFLX, AMD, and GOOG. Of these data, 7/12 of the data used for training and the remaining(5/12) is used for backtesting.

In order test the performance using log returns, we assumed to buy just a single unit of the stock based on the signal. For this purpose, after the model was trained, we introduced a column "*position*" whose value of 1 represents one unit of the stock and 0 represents no position on the stock

To evaluate the performance of our machine-based model, we compare the return of the ML strategy with the naïve approach of buy and hold for a certain period , in this case for 5 months. To this end, for the last 5 months, we calculate the log returns , `ln(Pt+1/Pt)` , for each day and put it in the column "*daily_profit*". The return for the ML based strategy is derived from the "*daily_profit*" and "*positions*" column using the formula

```
strategy_profit = daily_profit * positions
```

Fig 4. below shows the final column structure of the data after the backtesting is done. The signal column the test data contains the predicated values



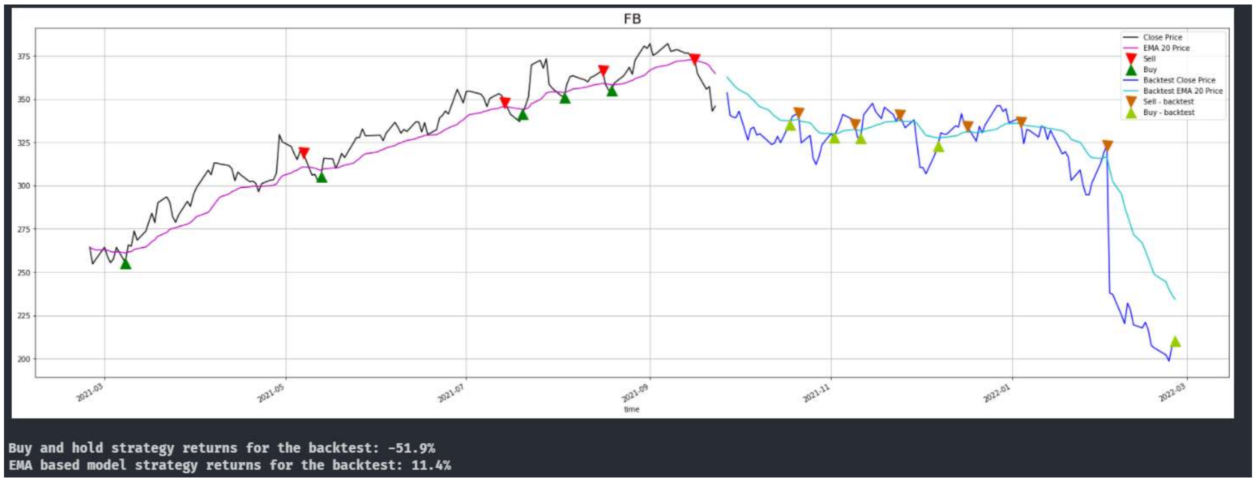
	time	close	min	max	ema_20	signal	positions	daily_profit	strategy_profit
0	2021-09-27 00:00:00-04:00	294.17	0.00	0.0	298.127383	0	0	-0.017	-0.000
1	2021-09-28 00:00:00-04:00	283.52	0.00	0.0	296.736204	0	0	-0.037	-0.000
2	2021-09-29 00:00:00-04:00	284.00	0.00	0.0	295.523232	0	0	0.002	0.000
3	2021-09-30 00:00:00-04:00	281.75	0.00	0.0	294.211495	0	0	-0.008	-0.000
4	2021-10-01 00:00:00-04:00	289.10	0.00	0.0	293.724686	0	0	0.026	0.000
5	2021-10-04 00:00:00-04:00	283.14	0.00	0.0	292.716621	0	0	-0.021	-0.000
6	2021-10-05 00:00:00-04:00	288.75	288.75	0.0	292.338848	1	1	0.020	0.000
7	2021-10-06 00:00:00-04:00	293.11	0.00	0.0	292.412291	0	1	0.015	0.015
8	2021-10-07 00:00:00-04:00	294.94	0.00	0.0	292.653025	0	1	0.006	0.006
9	2021-10-08 00:00:00-04:00	294.85	0.00	0.0	292.862261	0	1	-0.000	-0.000

Fig 4 Showing returns (daily_profit) for buy-and-hold, and strategy_profit for the EMA ML strategy

Results

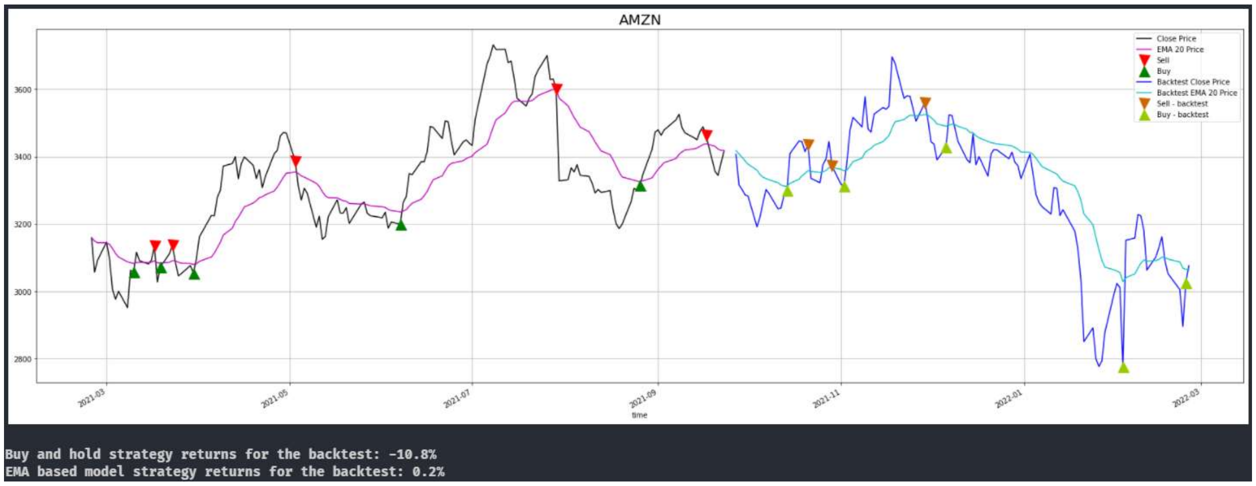
The results for each stock are shown below in the screenshots. Each of the Figs (5, 6, 7, 8,9) shows the following

1. 7 months of training data showing close price (black), 20-days exponential moving average(magenta), labels sell (red triagle), buy (green triangle)
2. 5 months of backtest data showing close price (blue), 20-days moving average (cyan), predicted labels – sell (light red) and buy (light green)
3. Shows at the bottom of the fig the returns of each strategy



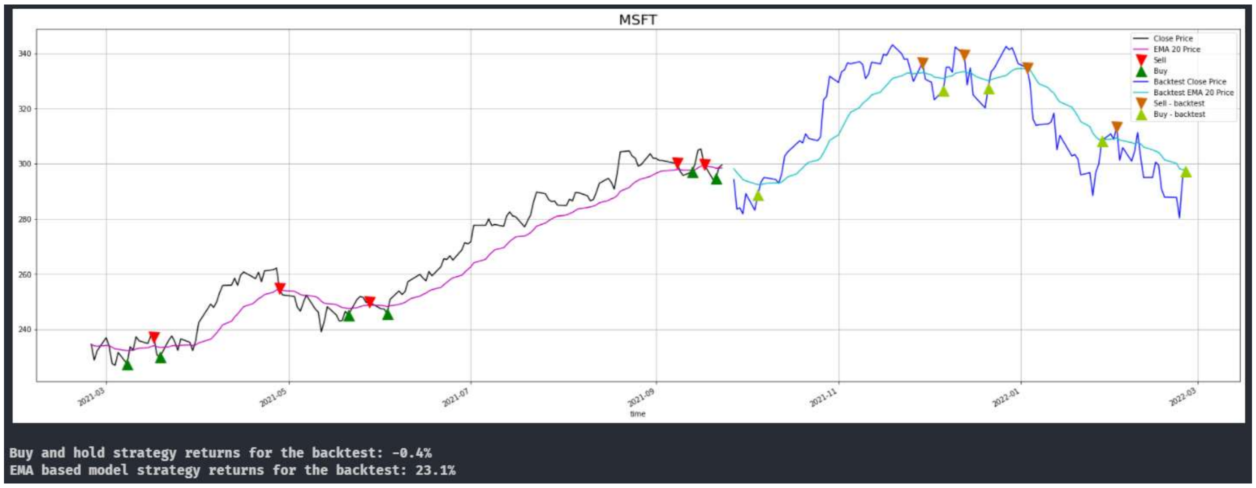
```
Buy and hold strategy returns for the backtest: -51.9%
EMA based model strategy returns for the backtest: 11.4%
```

Fig 5. Train and backtest data results for FB



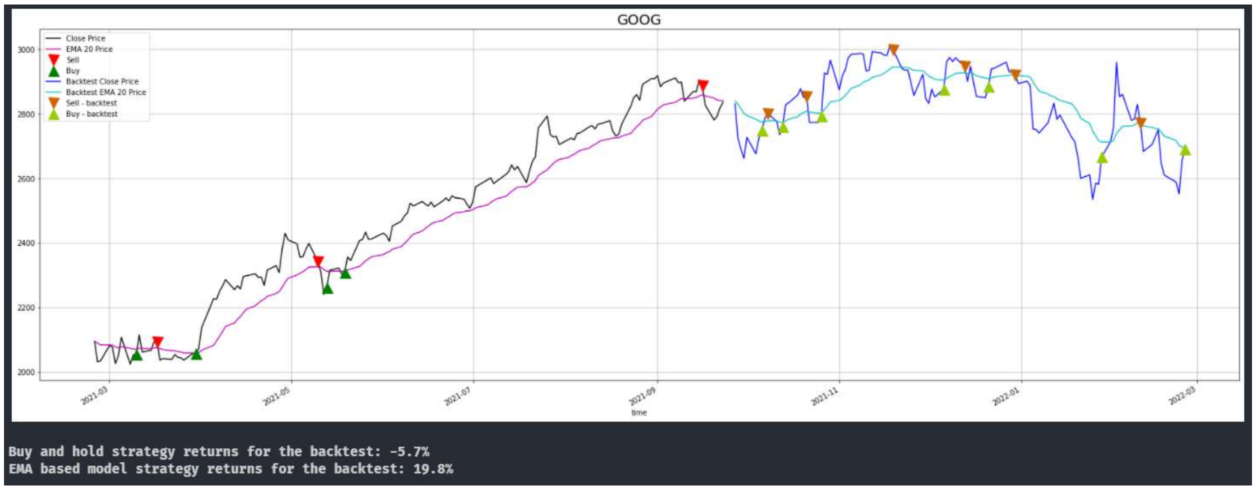
```
Buy and hold strategy returns for the backtest: -10.8%
EMA based model strategy returns for the backtest: 0.2%
```

Fig 6. Train, backtest results for AMZN



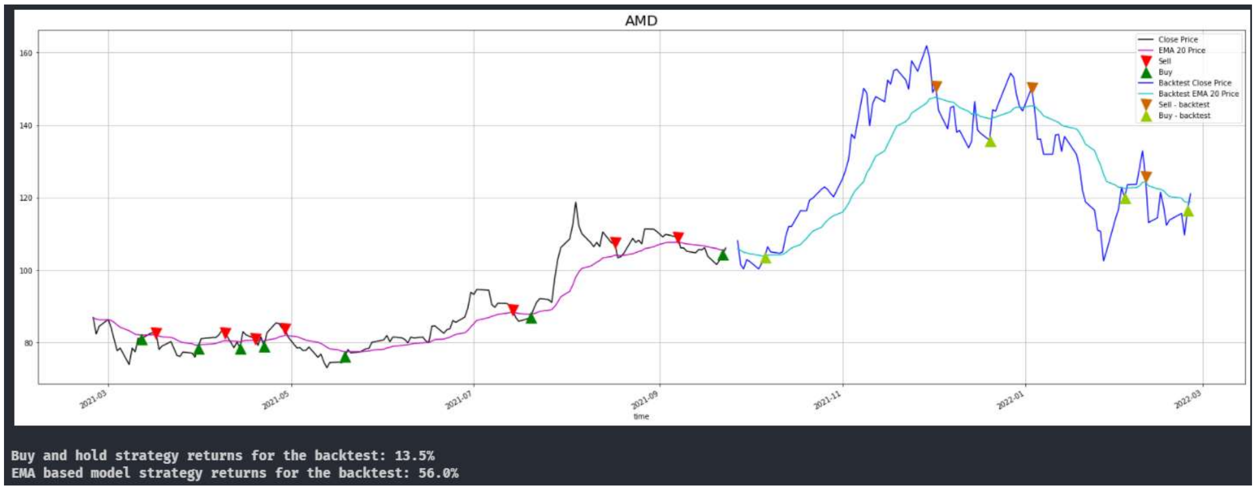
```
Buy and hold strategy returns for the backtest: -0.4%
EMA based model strategy returns for the backtest: 23.1%
```

Fig 7. Train, backtest results for MSFT



```
Buy and hold strategy returns for the backtest: -5.7%
EMA based model strategy returns for the backtest: 19.8%
```

Fig 8. Train, backtest results for GOOG



```
Buy and hold strategy returns for the backtest: 13.5%
EMA based model strategy returns for the backtest: 56.0%
```

Fig 9. Train, backtest results for AMD

Stock ML Model with STL

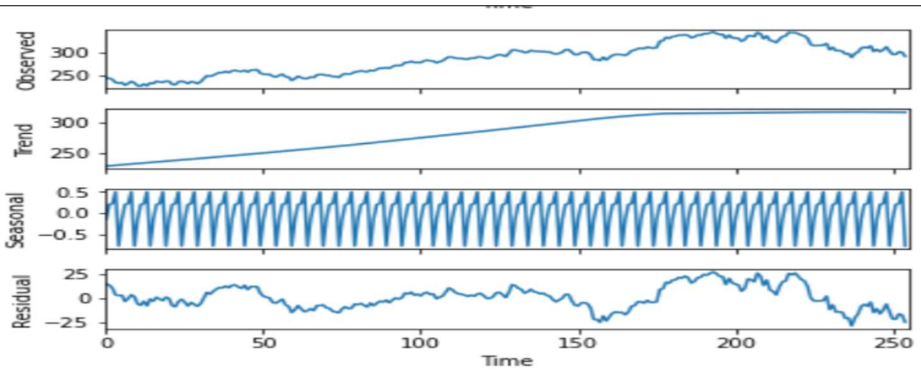
Process steps

The following are the steps used to create features and labels

1. Split 255 days (one year) data set to train and test
2. Calculated the Seasonal and Trend decomposition using Loess(STL)
3. Based on the residual histories of last n day, last close and open price as features
4. Based on the mean of residual divide to three parts as sell/hold/buy as labels
5. Train and test with linearSVD ML
6. sanitize prediction limited buy and sell point and prevention of excessive losses
7. Profit calculation based on *ln(sell price /buy price)*

Performance Evaluation backtest

1. Split 255 days (one year) data set to train and test
2. Train for 7 months and test for 5 months
3. Transfer each to get features and labels
4. Train and test with STL ML



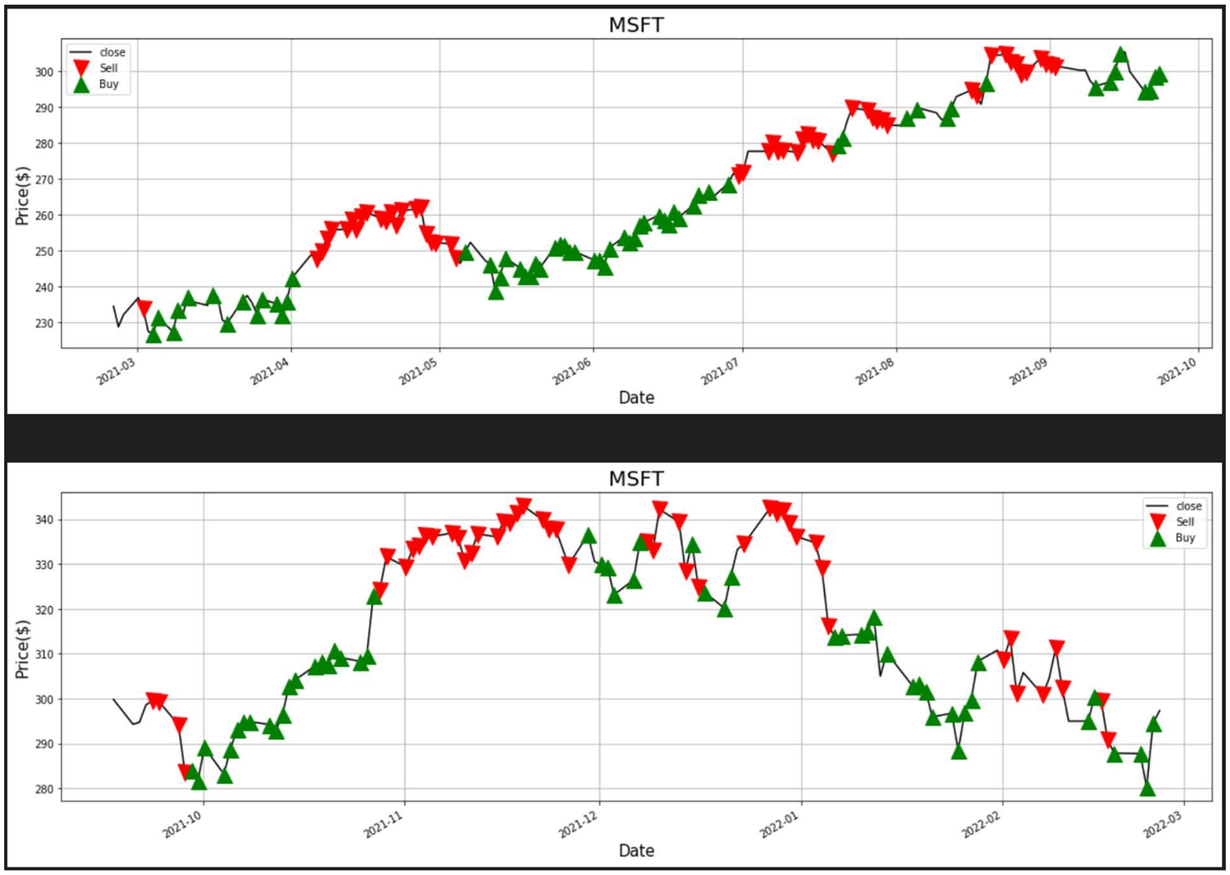
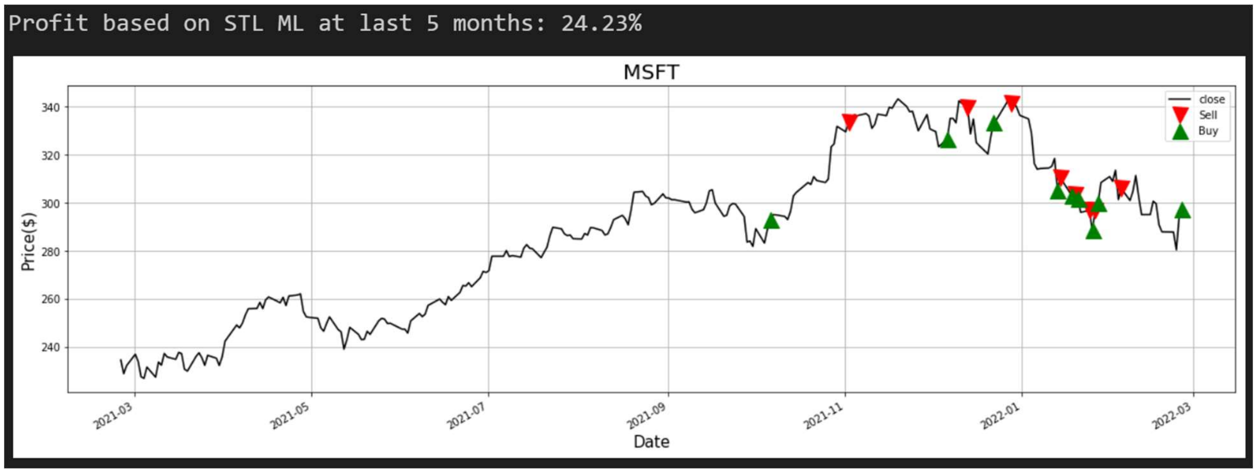


Fig 9. Train, test labes for GOOG

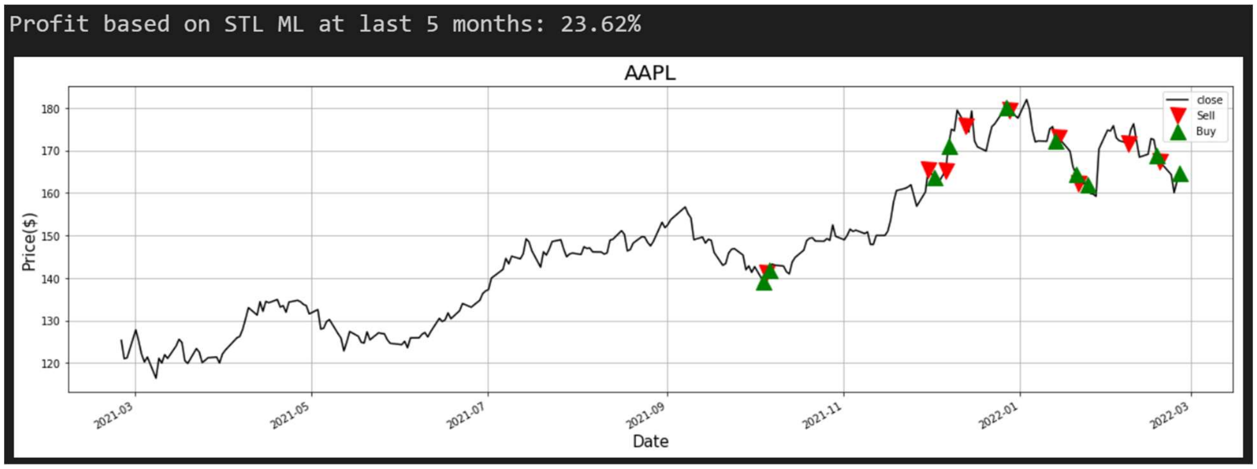
Results

The results for each stock are shown below in the screenshots. Each of the Figs (10, 11,12, 13,14) shows the following



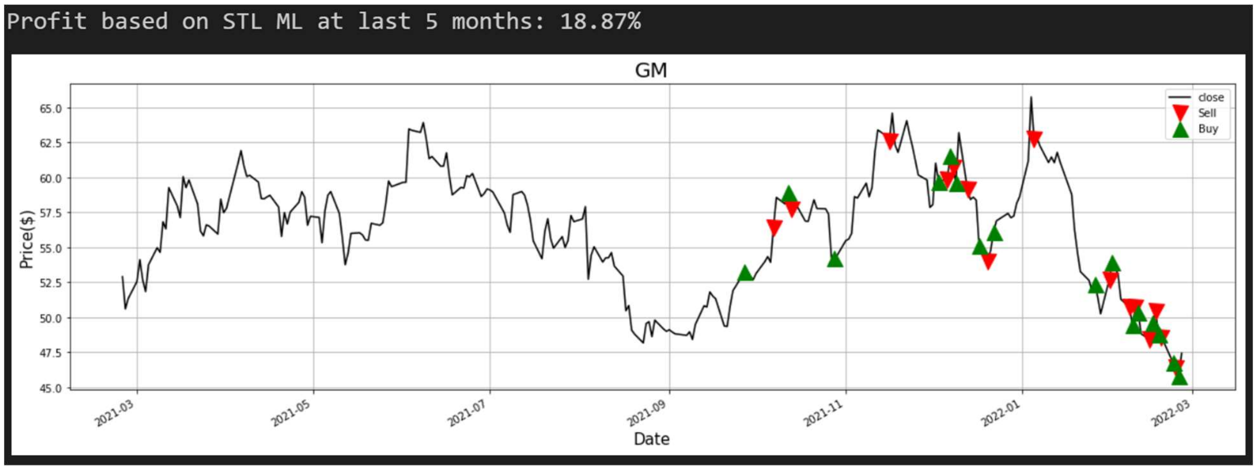
```
Profit based on STL ML at last 5 months: 24.23%
```

Fig 10. result for MSFT



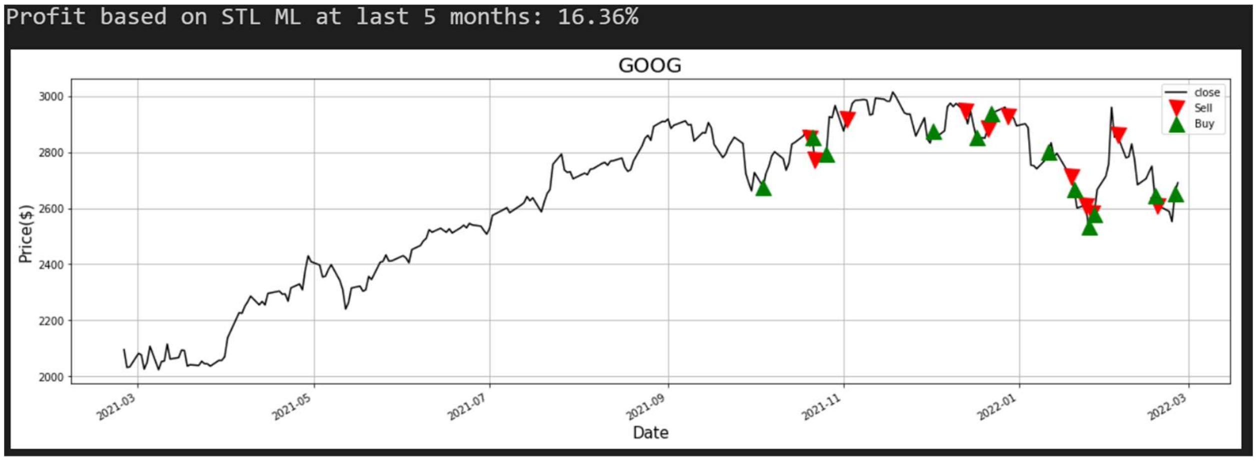
```
Profit based on STL ML at last 5 months: 23.62%
```

Fig 11. result for AAPL



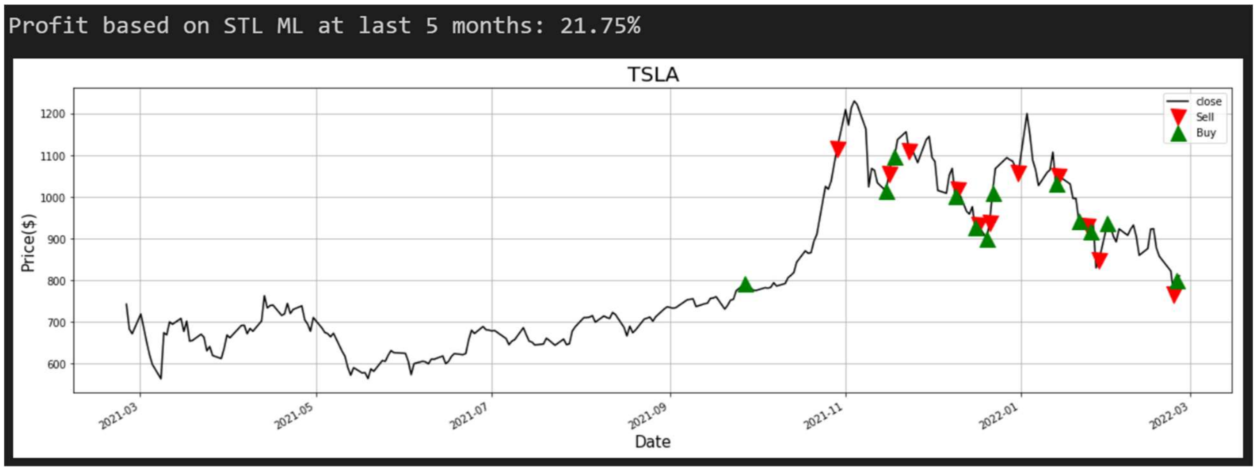
```
Profit based on STL ML at last 5 months: 18.87%
```

Fig 12. result for GM



```
Profit based on STL ML at last 5 months: 16.36%
```

Fig 13. result for GM



```
Profit based on STL ML at last 5 months: 21.75%
```

Fig 14. result for TSLA

Total profit: 104.8%

Comparing Strategies

	MSFT	AAPL	GM	GOOG	TSLA	mean	total
STL	24.23	23.62	18.87	16.36	21.75	21.75	104.83
	FB	MSFT	AMZN	AMD	GOOG	mean	total
EMA	11.4	23.1	0.2	56	19.8	19.8	110.5

EMA wins comparison, but STL gtes more stable at average. Also, STL sometimes will fail to predict result (only come with 0).

Github Project Location

Throughout the project, github is used a code and collaborative repository. It was just private repo , we just made it public

https://github.com/danielhunex/stock-ml-mini-project

Conclusion

In the case of performance metrics, comparing buy-and-hold strategy for a short period (5 months) of time might not be a good baseline (or a naïve approach) to compare with the EMA ML based strategy; however, it has satisfactorily proved how the strategy well performed.

STL is typically used for anomaly detection, which breaks the sequence into three components: seasonality, trend, and remainder. Essentially a standardized version of the original series, so this is what we monitor for exceptions. Using STL for stocks is a novel but logical thing to do. Essentially, if a company is healthy, its stock should rise or fall regularly. When something unexpected happens, the company's stock will react faithfully. Abnormal reactions can be detected by residual forces. Then by setting the residual threshold, people can sort out and understand the trend of the stock. Sort out the label by setting the lower zone for buy, the middle zone for watch, and the upper zone for sell. Finally, through them to machine learning, a pipeline of stock anomaly detection can be established. This applies to the majority of stock price cycle obviously abnormal fluctuations of the small range of stocks.

References

https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.argrelextrema.html

https://www.freecodecamp.org/news/algorithmic-trading-in-python/

https://algotrading101.com/learn/alpaca-trading-api-guide/

https://medium.com/wwblog/anomaly-detection-using-stl-76099c9fd5a7