Stock forecast system with Elliott Wave pattern recognition and adaptive trading strategy

Motonari ITO
SUNet ID: motonari

Sundararaman Shiva
SUNet ID: shivavs

In this project, we have built a system to advise an optimal stock market trading policy; when and how many stocks you should buy or sell. We ran the system over various stock symbols and evaluated the result.

The system consists of two components; reflex model based stock price predictors and state model based trading policy optimizers. The modularity allowed us to evaluate and optimize the performance of each component independently.

The result indicates there is probably a weak correlation between the historical stock price behavior and the future stock prices. However, it was hard to build a sensible policy optimizer with the given set of data and training time.

1 Introduction

Stock forecast has been studied and practiced with various degree of success. The technical analysis is a methodology based on the historical stock market prices [11]. The fundamental analysis tries to predict based on the business's financial statement [10]. Data mining over the Internet with sentiment analysis also became popular recently [2].

Elliott Wave Principle (EWP) is a classical technical analysis method [4, 8]. It is a hypothesis that stock market price can be modeled as a sequence of waves which shapes follow some defined rules. EWP suggests we can predict the future market price more accurately than a random chance by recognizing the wave pattern. This is distinct from other stock price prediction methods in that it relies solely on the historical price changes and doesn't use external information such as market sentiment or industrial news.

EWP has been criticized for the poor performance [1]. Notably, for a given stock historical data, the rules yields many different interpretation of the wave shapes. This uncertainty makes the future prediction hard, if not impossible, while one can claim the accuracy of the theory *after* the fact. In a sense, EWP is so powerful and complex model that it easily falls into over-fitting.

We, however, believe the essence of EWP is valid; future price is influenced by past price pattern. Intuitively, some often predict the price to go up if the past price has strong upward trend. Others may predict downward trend if the past price shows inverted-V shape. While such prediction may not be an inherent property of the stock market, the fact that many people believe that way affect the market. Therefore, it is probable that a reflex based machine learning algorithm can predict a future price based on the past prices.

There is also several attempt to predict the stock price based on the sentiment analysis over various data source on the Internet [5]. In the project, we tried to use New York Times Community API [9] to retrieve the customer comments and use them as a hint to a predictor.

Given we have a sensible predictor, it is still an open question when and how to trade stocks to optimize the asset because the predictor is inherently imperfect. For example, it may not be smart to sell the entire stocks asset immediately just because a predictor says price might go down. The trading decision should be educated by the actual performance of predictors.

Our approach is to use a state based learning algorithm to find the optimal trading strategy. Intuitively, as it runs the predictors on the historical data, the trader will learn the peculiarity of each predictor.

2 System

The system consists of two parts: predictors and traders.

A predictor predicts a future price change of a particular stock. Each predictor uses different input data and inference algorithm. We evaluate the performance of each implementation and plug-in some of the best predictor to the final system.

A trader uses the prediction and learns the optimal trading policy: when to buy/sell how much stocks.

There are five predictors (SimpleNNPredictor, LinearPredictor, SentimentPredictor, PatternPredictor, and CheatPredictor) and two traders (RoteQTrader and QTrader).

For the purpose of the discussion, we use the following definitions. More variables will be defined as needed.

$$p_i := \text{Stock price of the } i\text{-th day}$$

$$priceChange(p_{old}, p_{new}) := \frac{p_{new} - p_{old}}{p_{old}}$$

2.1 Predictors

All the predictors are configured with a hyper parameter D, which indicates the future date delta to predict.

They also implement three operations.

extractFeatures For a given date, the function returns the feature vector for the prediction.

train The function is invoked with a feature vector and the target value, which is the price change after D days.

predict The function is invoked with a feature vector and returns the predicted price change after D days.

2.1.1 SimpleNNPredictor

The predictor uses the multilayer perception implementation from scikit-learn [7]. It extract the feature by looking back the prior stock price.

The look back date is defined as a vector $\boldsymbol{b}$:

$$\boldsymbol{b} \leftarrow [89, 55, 34, 21, 13, 8, 5, 3, 2, 1]$$

In the training phase, it looks back the stock prices and calculate the price changes X compared to the current price. Suppose the current date index is denoted as i, the input to the algorithm is:

$$X_i \leftarrow \left\{priceChange(p_{i-j}, p_i) : j \in \boldsymbol{b}\right\}$$

The target value is is the actual stock price change for the given future date: $priceChange(p_i, p_{i+D})$.

The algorithm uses two hidden layers; 3 and 2 nodes each.

2.1.2 LinearPredictor

The predictor is same as SimpleNNPredictor except that the underlying algorithm uses a linear regression with stochastic gradient descent, also from scikit-learn [7].

2.1.3 SentimentPredictor

The predictor uses New York Times Community API [9] to retrieve the customer comments of each news article from Jan 1, 2010 to Nov 10, 2016.

There are 62000 comments in the period. We group the comments by keywords related to the stock symbol. For example, we use comments which contains any of aapl, apple, iphone, ipad, mac, ipod, or ios to predict AAPL stock price.

Symbol	Keywords	Comments
aapl	aapl, apple, iphone, ipad, mac, ipod, ios	463
bp	bp, dudley, oil, major	1873
cop	conocophillips, lance, oil, major	1853
cost	costco, brotman, wholesale	41
cvx	chevron, oil, gas	1029
dj	dow, jones, dj	76
hd	home, depot, hd	1395
ibm	ibm, watson, ginni, rometty	24
ko	coca, cola, ko	18
low	lowe, lowes	2
nke	nike, nke	6
qcom	qualcomm, qcom	0
rut	russell, rut	25
tgt	target, tgt	191
wmt	wallmart, wmt	3
xcom	xcom, xtera	0

Table 1: Sentiment analysis comment keywords

Table 1 shows the keywords we use and the number of comments used for each symbol.

Then, it uses Stanford Core NLP [6] to find the sentiment. The algorithm returns a tuple of sentimentValue and sentiment for each sentence in the comment. For example, a very positive sentence may return (3, 'positive'). A weakly negative sentence would return (1, 'negative').

We estimate the sentiment of the comment by averaging the sentiment values.

```
def comment_sentiment(comment):
  total = 0
  score = 0
  for each sentence in comment:
    if sentiment == 'positive':
      score += sentimentValue
      total += sentimentValue
    elif sentiment == 'negative':
      score -= sentimentValue
      total += sentimentValue
    else:
      total += sentimentValue

  return float(score) / total
```

Then, for a given date, we look back the last 30 days of the sentiment for the stock and create a feature vector of size 30.

We use a linear regression with stochastic gradient descent from scikit-learn [7] to train the predictor. The target

n-days ago	price (dollar)
today	100
1	98
2	95
3	95
5	93
8	92
13	96
21	99
34	100

Table 2: Example stock price change

value is is the actual stock price change for the given future date: $priceChange(p_i, p_{i+D})$.

2.1.4 PatternPredictor

The predictor uses a table look up method.

It extracts the feature vector $\phi(x)$ by looking back the prior stock price. The look back date is defined as a vector $\boldsymbol{b}$:

$$\boldsymbol{b} \leftarrow [0,1,2,3,5,8,13,21,34]$$

It looks back the stock prices and map the price changes of each interval to $+1$ (= price went up) or -1 (= price went down). When there is no price change, we pick $+1$ or -1 randomly.

For example, suppose we have the stock price history in table 2, the the feature vector would be $\phi(x) = [-1,-1,-1,+1,+1,0,+1,+1]$.

The feature vector is used as a key of the table look up. Since there are eight intervals where each interval gets one of two values, there will be 256 entries in the table.

For the target value, the future price p_{i+D} is mapped to $+1$, -1, or 0 compared to the current price p_i. Note that we use 0 when there is no price change.

In the training phase, the table is updated to the average of the target value y as follows.

$$\eta \leftarrow \frac{1}{\text{\# of updates to the entry}}$$
$$\text{table}[\phi(x)] \leftarrow (1-\eta)\text{table}[\phi(x)] + \eta y$$

In the prediction phase, we simply look up the table and return the estimated target value.

2.1.5 CheatPredictor

The predictor returns the future price changes by actually looking at the data. It is used for the testing purpose.

2.2 Trader

If we had a perfect predictor, the optimal strategy would have been to buy before the stock price goes up and to sell before it goes down. However, no predictor is perfect.

To learn the optimal trading policy given the uncertain predictions, we use a reinforcement learning.

For the sake of discussion, we define some additional variables here.

$o_i :=$ The number of stocks owned on i-th day.
$c_i :=$ the maximum number of stocks we could buy with the current cash on i-th day.
$m_i :=$ the predicted slope by performing a least square polynomial fit over the predicted future price changes.
$r_i :=$ the sum of residuals of the predicted slope above.

The goal is to find a trading policy to maximize our asset value u after the sequence of trades. Suppose the last day index is n, the asset value u_n is defined:

$$u_n = p_n(o_n + c_n)$$

We have two traders; RoteQTrader and QTrader. QTrader uses a function approximation while RoteQTrader doesn't.

2.2.1 Model (RoteQTrader)

We model the problem as MDP where we don't know the transition function.

State For i-th day, the state is defined to have a tuple

$$(o_i, \lfloor c_i \rfloor, I[m_i > 0])$$

Initial State Before running MDP starting at day index $start$, we initialize the state as follows.

$o_{start} = 0$
$c_{start} = 10$
m_{start} is initialized by a predictor based on the first day.

Action The action is an integer in the range: $[-o_i, \lfloor c_i \rfloor]$. The negative value means to sell owned stocks for that amount, and the positive value means to buy stocks for that amount.

Transition On taking a action, the state moves to the next day. Note that the system doesn't know the next state beyond today.

Reward The reward is the difference of asset value before and after the state transition.

$$Reward = p_i(o_i + c_i) - p_{i-1}(o_{i-1} + c_{i-1})$$

Intuitively, we want to have more stocks when the stock price is high and we want to have more cash when the stock price is low.

EndState $o_i < 1$ or the current date hits end of the period.

2.2.2 Model (QTrader)

The model is same as RoteQTrader except these differences.

State For i-th day, the state is defined to have a tuple (o_i, c_i, m_i, r_i).

Initial State Before running MDP starting at day index $start$, we initialize the state as follows.

$o_{start} = 0$
$c_{start} = 10$
m_{start} and r_{start} are initialized by a predictor based on the first day.

2.2.3 Algorithm

RoteQTrader uses Q-learning with epsilon-greedy learning. QTrader uses a function approximation and epsilon-greedy learning.

We use the following parameters and definitions.

$$\begin{aligned} \varepsilon &:= 0.9 (\text{epsilon-greedy threshold}) \\ \gamma &:= 0.9 (\text{discount factor}) \end{aligned}$$

In addition, for QTrader, we denote the weights to learn as $\boldsymbol{w}$. Consequently, $\hat{Q}_{opt}(s,a;\boldsymbol{w})$ is defined as follows where $\phi(s,a)$ is the feature extractor.

$$\hat{Q}_{opt}(s,a;\boldsymbol{w}) := \boldsymbol{w} \cdot \phi(s,a) \qquad \text{For QTrader}$$

For i-th day, we pick an action from the range: $[-o_i, \lfloor c_i \rfloor]$. Based on whether a random number $[0.0, 1.0]$ is greater than ε, we pick exploration or exploitation.

$$\pi_{act}(s) = \begin{cases} \arg\max_{a \in actions} \hat{Q}_{opt}(s,a) & \text{probability } 1-\varepsilon \\ \text{random choice from actions} & \text{probability } \varepsilon \end{cases}$$

On taking an action, the trader gets a reward.

$$\begin{aligned} Reward &= u_{i+1} - u_i \\ &= p_{i+1}(o_{i+1} + c_{i+1}) - p_i(o_i + c_i) \end{aligned}$$

Then MDP moves to the next state, which represents the stock market and the current asset of the next day.

$$\begin{aligned} c_{i+1} &\leftarrow (c_i - action)\frac{p_i}{p_{i+1}} \\ o_{i+1} &\leftarrow o_i + action \\ m_{i+1}, r_{i+1} &\leftarrow \text{predict(i+1)} \end{aligned}$$

The function predict invokes predictors and returns the future stock market trend.

```
def predict(i):
  delta = [0.0]
  prediction = [0.0]
  for p in predictors:
    phiX = p.extractFeature(i)
    prediction += [p.predict(phiX)]
    delta      += [p.D]
  slope, residual =
      linear_fit(delta, prediction)
  return slope, residual
```

For QTrader, we tested two different feature extractors for the function approximation; simple and complex.

The simple feature extractor uses just one feature; a product of the predicted future price slope and the action value. We use this just for testing purpose.

$$\phi_s(s_i, a_{i+1}) := [m_i \times a_{i+1}]$$

The complex feature extractor uses some combinations of the simple feature. We use this to predict the actual stock behavior.

$$\phi_c(s_i, a_{i+1}) := [m_i \times a_{i+1}, m_i^2 \times a_{i+1}, m_i \times a_{i+1}^2, m_i^2 \times a_{i+1}^2]$$

RoteQTrader updates Q_{opt} table as follows.

$$\begin{aligned} \eta &\leftarrow \frac{1}{\text{\# of updates to } \hat{Q}_{opt}(s,a)} \\ \hat{Q}_{opt}(s,a) &\leftarrow \hat{Q}_{opt}(s,a) - \eta\left(\hat{Q}_{opt}(s,a) - reward + \gamma \hat{V}_{opt}(s')\right) \end{aligned}$$

QTrader updates the weights as follows.

Symbol	Start Date	End Date
aapl	1980-12-12	2016-11-11
bp	1977-01-03	2016-11-11
cop	1981-12-31	2016-11-11
cost	1986-07-09	2016-11-11
cvx	1970-01-02	2016-11-11
dj	1985-01-29	2016-11-11
hd	2011-11-14	2016-11-11
ibm	1962-01-02	2016-11-11
ko	1962-01-02	2016-11-11
low	2011-11-14	2016-11-11
nke	1980-12-02	2016-11-11
qcom	1991-12-13	2016-11-11
rut	1987-09-10	2016-11-11
tgt	1980-03-17	2016-11-11
wmt	2011-11-14	2016-11-11
xcom	1970-01-02	2016-11-11

Table 3: Yahoo Finance Stock Data

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta[\hat{Q}_{opt}(s,a;\boldsymbol{w}) - (reward + \gamma\hat{V}_{opt}(s'))\phi(s,a)$$

Note that

$$\hat{V}_{opt}(s') \leftarrow \max_{a \in actions(s')} \hat{Q}_{opt}(s',a)$$

2.2.4 Test

In test phase, we run the learning algorithm on a new data with the following modification.

Always choose the optimal action, that is to set $\varepsilon = 0$
Skip weight update step.

Then, see if how much money earned or lost by looking at the final asset value.

3 Performance

3.1 Predictors

We use stock price data from Yahoo Finance [3]. It provides day-to-day closing stock price for the period shown in table 3.

We measure the performance of predictors by whether it predicts upward/downward trend correctly on the test data. For example, if the predictor predicts upward trend correctly, we call it as true positive.

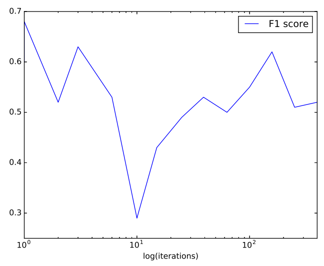

Fig. 1: SimpleNNPredictor F1 score by the number of iterations

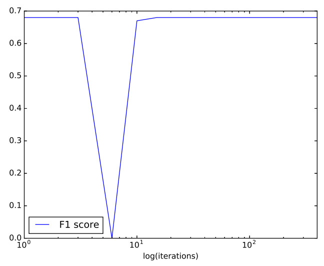

Fig. 2: LinearPredictor F1 score by the number of iterations

Each predictor is trained over all the stock symbols up to 2015-11-11. The rest of the one year data is used for testing.

In one training iteration, for each stock symbol, we pick a date randomly and run through the predictor for the next 128 days. Since we have 16 stock symbols, one iteration feeds $128 * 16 = 2048$ training samples (note: they are not necessarily unique) to the predictor.

3.1.1 SimpleNNPredictor

Table 4, 5, and 6 shows prediction performance of SimpleNNPredictor after 398 iteration. We can observe the accuracy is slightly better than 50%.

However, figure 1 shows that the predictor performance, in terms of F1 score, does not improve over the training iteration.

3.1.2 LinearPredictor

Table 7, 8, and 9 shows prediction performance of LinearPredictor after 398 iteration. Although we can observe the accuracy is slightly better than 50%, the confusion matrix indicates strong bias in the prediction.

Figure 2 shows that the predictor performance, in terms of F1 score, does not improve over the training iteration.

3.1.3 PatternPredictor

Table 10, 11, and 12 shows prediction performance of PatternPredictor after 398 iteration.

This predictor shows sensible performance. The accuracy is almost always slightly more than 50%.

Figure 3 shows that the predictor performance, in terms of F1 score, improves as we increase the training iteration.

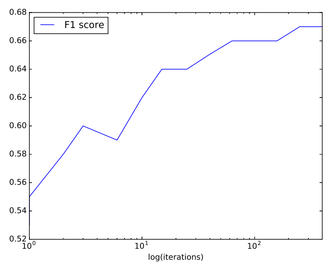

Fig. 3: PatternPredictor F1 score by the number of iterations

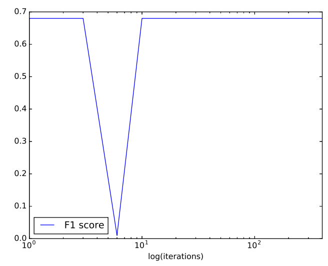

Fig. 4: PatternPredictor F1 score by the number of iterations

		Predicted			
		+	−	Accuracy	0.51
Actual	+	0.18	0.17	F1 Score	0.43
	−	0.32	0.33		

Table 4: SimpleNNPredictor, $D = 1$, 398 iterations

		Predicted			
		+	−	Accuracy	0.49
Actual	+	0.20	0.30	F1 Score	0.44
	−	0.21	0.29		

Table 5: SimpleNNPredictor, $D = 3$, 398 iterations

3.1.4 SentimentPredictor

Table 13, 14, and 15 shows prediction performance of PatternPredictor after 398 iteration. Although we can observe the accuracy is slightly better than 50%, the confusion matrix indicates strong bias in the prediction.

Figure 4 shows that the predictor performance, in terms of F1 score, does not improve over the training iteration.

3.2 Trader

We measured the performance of the trader by comparing the asset value between the beginning and the end of the episode.

First, we split the data set to training set (till 2015-11-11) and test set (after 2015-11-11).

Then, we trained the predictors on the training set. Based on the predictor performance analysis in figure 3, we use 100 iteration for the predictor training.

		Predicted			
		+	−	Accuracy	0.46
Actual	+	0.29	0.22	F1 Score	0.52
	−	0.32	0.17		

Table 6: SimpleNNPredictor, $D = 7$, 398 iterations

		Predicted			
		+	−	Accuracy	0.36
Actual	+	0.35	0.00	F1 Score	0.52
	−	0.64	0.01		

Table 7: LinearPredictor, $D = 1$, 398 iterations

		Predicted			
		+	−	Accuracy	0.50
Actual	+	0.50	0.00	F1 Score	0.67
	−	0.50	0.00		

Table 8: LinearPredictor, $D = 3$, 398 iterations

		Predicted			
		+	−	Accuracy	0.51
Actual	+	0.51	0.00	F1 Score	0.68
	−	0.49	0.00		

Table 9: LinearPredictor, $D = 7$, 398 iterations

		Predicted			
		+	−	Accuracy	0.44
Actual	+	0.26	0.10	F1 Score	0.48
	−	0.46	0.18		

Table 10: PatternPredictor, $D = 1$, 398 iterations

		Predicted			
		+	−	Accuracy	0.50
Actual	+	0.45	0.06	F1 Score	0.64
	−	0.44	0.06		

Table 11: PatternPredictor, $D = 3$, 398 iterations

		Predicted			
		+	−	Accuracy	0.52
Actual	+	0.49	0.02	F1 Score	0.67
	−	0.46	0.02		

Table 12: PatternPredictor, $D = 7$, 398 iterations

		Predicted			
		$+$	$-$	Accuracy	0.36
Actual	$+$	0.36	0.00	F1 Score	0.52
	$-$	0.64	0.00		

Table 13: SentimentPredictor, $D = 1$, 398 iterations

		Predicted			
		$+$	$-$	Accuracy	0.50
Actual	$+$	0.50	0.00	F1 Score	0.67
	$-$	0.50	0.00		

Table 14: SentimentPredictor, $D = 3$, 398 iterations

		Predicted			
		$+$	$-$	Accuracy	0.51
Actual	$+$	0.51	0.00	F1 Score	0.68
	$-$	0.49	0.00		

Table 15: SentimentPredictor, $D = 7$, 398 iterations

The trader calculate m_i, the future price changes slope, by the predictions of predictors. We use a various number of a same predictor with different D value for the experiment. For example, in one test case, we use the combination of three `PatternPredictor` with $D = 1$, $D = 3$, and $D = 7$.

With the trained predictors, we run `QTrader` on the training set. In the training, we pick a start date index randomly and run an episode up to 90 days. The process is repeated for the specified number of times.

3.3 Correctness

Before trying a complex scenario, we verified the correctness of the trader implementation. We use `CheatPredictor` and the simplified feature extractor $\phi_s(s,a)$ so that the performance is not impacted by the predictor and we can reason the resultant weights easily.

In this simplified scenario, even with one iteration, the trader quickly learned the weight value of [0.0873]. Recall that $\phi_s(s,a)$ is defined as $[m_i \times a_{i+1}]$. Intuitively, the weight value can be interpreted as follows.

1. When the predicted future price is upward ($m_i > 0$), the optimal action to maximize $\hat{Q}_{opt}(s,a)$ is the max value of action; buy as many stocks as possible.
2. When the predicted future price is downward ($m_i < 0$), the optimal action to maximize $\hat{Q}_{opt}(s,a)$ is the min value of action; sell as many stocks as possible.

Therefore, the trader implementation appears to be sensible.

	Gain (US$)		
Symbol	$D = 1$	$D = 7$	$D = [1,3,7]$
dj	9177.92	9033.52	15234.80
qcom	-8.07	75.95	249.50
rut	1154.00	-200.12	2838.05
wmt	214.27	63.82	59.76
hd	131.67	162.74	163.42
low	-123.54	38.83	-6.46
tgt	-81.63	-79.02	-100.02
cost	-180.67	115.74	-82.95
nke	55.58	-86.02	-10.95
ko	-30.42	-37.65	7.62
xom	-4.58	144.83	174.51
cvx	-60.90	274.49	180.37
cop	-157.38	46.34	-101.95
bp	60.41	86.08	46.37
ibm	196.59	65.05	218.54
aapl	-42.90	118.20	141.19
Sum	10300.35	9822.78	19011.80

Table 16: Trader with PatternPredictor, 100 iterations

3.4 Running on the test set

According to the predictor analysis, `PatternPredictor` is the most sensible predictor in our portfolio. We have conducted these test cases. The expectation is that short term prediction should be better than the long term prediction. Also, giving multiple prediction points should inform the trader better.

Use one instance of the predictor with $D = 1$.
Use one instance of the predictor with $D = 7$.
Use three instances of the predictor with $D = [1,3,7]$.

Table 16 shows the result. We indeed made a money as net, though we lost in some symbols.

4 Conclusion

We have successfully confirmed, with the result of `PatternPredictor`, the stock market behavior can be predicted by the prior price changes.

The predictor uses table look up algorithm by translating the price changes into a sequence of $+1$ or -1. We tried more generalized algorithms (`SimpleNNPredictor` and `LinearPredictor`) but they did not perform well.

It is, in a sense, supported by Elliott Wave Theory, which focuses on the wave shape instead of the exact prices. However, by simplifying the stock market behavior into a table of

256 entries, it is probable that we miss some important features; even `PatternPredictor` performs just silightly better than 50% accuracy anyway. We might be able to design a better predictor by using more complex model (more hidden layers and/or features) – it's a future research work.

The other predictor, `SentimentPredictor`, may have potential but didn't perform well in the current configuration. We suspect it's due to the lack of data – the API provided just 62000 comments over the six years period. Besides, there are not so many article and user comments about stock market in New York Times; they mostly talk about politics and social matter. Unfortunately, we could not find any other convenient free data source. For example, Twitter API does not allow us to access older tweets than a few weeks ago.

Despite the challenge of the predictors, the trader works reasonably well with `PatternPredictor`, especially when we use three of them together to predict the price of different future dates. It is especially encouraging to observe that, as an Apple employee, it made more than $140 over last year when AAPL did not perform well.

5 Code and Data

In the attached zip file, we have the following data and code.

plotPredictorPerformance.py Plots predictor performance table and graph.

plot_trader_perf.py Plots trader performance table and graph.

predictor.py The predictor implementations.

trader.py The trader implementation.

testPredictor.py Tests predictor implementations.

testTrader.py Tests trader implementation.

predictor_perform_* Predictor performance data obtained by testPredictor.py. Note that the date in the file name indicates when the data was obtained. Old data may be generated by a program code with bugs.

trader_perf_{iteration}_{D} The trader performance data with iterations and D values.

nytimes/download.py Downloads New York Times article comments through the API.

nytimes/nyt_comment_{date}.json The downloaded comments.

nytimes/nytimes.py Reorganize comments into one json file where the key is a stock symbol.

nytimes/trend.json Output file from the above script.

nytimes/stanford_nlp.py Add sentiment data into the file above.

nytimes/trends_with_sentiment.json Output file from the above script.

data/*.json Stock data from Yahoo! Finance.

References

[1] David Aronson. *Evidence-based technical analysis : applying the scientific method and statistical inference to trading signals*. John Wiley & Sons, Hoboken, N.J, 2007.

[2] Philip Ball. Counting google searches predicts market movements, 2013. [Online; accessed 11-December-2016].

[3] Yahoo! Finance. Yahoo! finance. [Online; accessed 12-December-2016].

[4] A. J. Frost. *Elliott wave principle : key to stock market profits*. New Classics Library, Chappaqua, N.Y, 1981.

[5] Xiao-Jun Zeng Johan Bollen, Huina Mao. Twitter mood predicts the stock market. In *Journal of Computational Science 2(1), March 2011*, pages 1 – 8, 2011.

[6] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60, 2014.

[7] scikit learn. scikit-learn. [Online; accessed 12-December-2016].

[8] StudyOfCycles.com. Study of cycles – where direction and timing matter most, 2014. [Online; accessed 11-December-2016].

[9] The New York Times. The new york times developer network. [Online; accessed 11-December-2016].

[10] Wikipedia. Fundamental analysis — wikipedia, the free encyclopedia, 2016. [Online; accessed 12-November-2016].

[11] Wikipedia. Technical analysis — wikipedia, the free encyclopedia, 2016. [Online; accessed 29-October-2016].