CS221 Fall 2016 Project [p-progress]

SUNet ID: motonari
Name: Motonari ITO
Collaborators: Sundararaman Shiva

1 Scope

1.1 Elliott Wave Principle

Elliott Wave Principle (EWP) 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 method in that it relies sololy on the historical price changes and doesn't use external information such as market sentiment or industrial news. While we could improve the prediction by using those methods complementary, for this project, we focus on EWP approach.

While EWP has several rules about wave shapes, we don't use them in the original form (except in the baseline algorithm explored in `p-proposal`.) Instead, we use Reinforcement Learning to find a new set of rules which predict the price more accurately.

1.2 Input and Output

The system learns the stock price change pattern from a historic price obtained from Yahoo Finance (`https://finance.yahoo.com/quote/AAPL/history?p=AAPL`) and advises the optimal action (buy or sell) for today.

For example,

Input Historic stock price data as an array and the currently own stocks as a dictionary: {PurchaseDate, NumberOfStocks}.

- AAPL stock data (Dec 12, 1980 - Oct 22, 2016)
- {Nov 12 2015: 121 stocks, Apr 1 2016: 53 stocks, ...}

Output Optimal buy / sell today. For example, {Sell: { Nov 12 2015: 11 stocks, Apr 1 2016: 5 stocks }, Buy: 12 stocks }.

2 Model

We model a stock market as an MDP where we neither know the transitions nor reward functions.

2.1 State

The state is the historical stock price change and the set of currently own stocks. As of the current implementation, they are:

- Let p_i be closing price $N[i]$ days ago, where $N \in [87, 54, 33, 21, 13, 8, 5, 3, 2, 1]$. We use a set of Boolean indicating the price went up or down between each day, which is $\{I[p_i < p_{i+1}]\}$, as a state. ($I[..]$ is an indicator variable.)

  Note that we use Fibonacci numbers because EWP suggests there are some correlation between the stock market behavior and Fibonacci numbers.

- The current stocks owned. This is an array of tuple of (difference between purchased price and today's price, number of stocks).

2.2 Action

On a given day, we choose one or more of these operations.

- Don't do anything
- Buy stock(s).
- Sell stock(s). It specifies which stocks to sell in the currently owned stocks array.

2.3 Reward

We get reward according to the number of sold stocks and the current price.

2.4 Transition

After the action, we move to the next day, which has a new state based on the stock price and the prior action.

When the state reaches today, the system reports the action of the day which would results in the maximum reward.

3 Algorithm

We use Q-learning with Epsilon-greedy and function approximation. We use the following variables.

$$\begin{aligned}
\epsilon &:= \text{Parameter for epsilon-greedy policy}\\
\phi(s,a) &:= \text{feature extractor}\\
p_i &:= \text{stock price of i-th day}\\
\boldsymbol{w} &:= \text{weight to learn}\\
\hat{Q}_{opt}(s,a;\boldsymbol{w}) &:= \boldsymbol{w}\dot{\phi}(s,a)
\end{aligned}$$

3.1 Learning

In learning phase, we loop through the stock price of each day.

Choose an action

For each day, we examine the state and obtain the available actions. For example, while we can always buy a stock, we cannot sell a stock if we don't own one.

Based on whether a random number $[0.0, 1.0]$ is greater than ϵ, we pick exploration or exploitation.

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

Calculate reward

Based on the action, we calculate the reward.

$$reward = \begin{cases} -p_i \times \text{number of stocks to buy} & \text{“buy” action} \\ p_i \times \text{number of stocks to sell} & \text{“sell” action} \\ 0 & \text{no action} \end{cases}$$

Find next state

Let $s' :=$ state. Based on the action, we update s'.

- When we bought stocks, add `(0, number of stocks)` to the currently own stock array. It means we have stocks with zero price difference.
- When we sold stocks, subtract the number of stocks.

Then, we update s' for the next day. The historical prices are shifted. The price differences in the currently owned stocks are updated based on the new stock price.

Update weights

We update the weights as follows.

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

3.2 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 $\epsilon = 0$
- Skip weight update step.

Then, see if how much money earned or lost by looking at the sum of all rewards.

4 Preliminary Implementation

In the current preliminary implementation, we have made the following simplification.

- Feature extractor $\phi(s, s)$ returns `list(priorPattern) + [len(currentAssets)] + [action]`, which contains only the total number of current assets and no generalization on the prior stock price change.
- Actions are limited to buy or sell only one stock per day.

We run the algorithm over various stock over one year up to today. In short, we lost a lot of money. Also, in some stocks, the algorithm thinks don't buy or sell any stocks is the optimal choice.

dj	-1486.736328
gdx	0.000000
qcom	-61.536490
rut	268.510009
wmt	212.577556
hd	0.000000
low	-12.438588
tgt	0.000000
cost	0.000000

nke	0.000000
ko	0.000000
xom	0.000000
cvx	-18.072437
cop	-192.961476
bp	0.000000
ibm	-13.043052
aapl	4.597696