Algorithmic trading with reinforcement learning

Capstone Proposal

Domain Background

(approx. 1-2 paragraphs)

The algorithm trading market has experienced significant growth rate owing to increasing automation process in trading by a large number of firms. Integrated financial markets help the local vendors into buying foreign assets with the reduced risks. Involvement of various international markets has directed to global distribution of savings and also aiding the countries in creating opportunities for portfolio diversification and risk sharing. Global algorithm trading market is expected to grow from US$ 8,790.7 Mn in 2016 to US$ 18,160.8, Mn by 2025 at a CAGR of 8.7% between 2017 and 2025.

Training Reinforcement Learning agents to trade in the financial markets can be an extremely interesting research problem. I believe that it has not received enough attention from the research community but has the potential to push the state-of-the art of many related fields. It is quite similar to training agents for multiplayer games such as DotA, and many of the same research problems carry over.

Personally, I have a background in Finance and Banking and I want to pursue this project to make a start in developing my own algorithmic trading algorithm and later may be a start-up!!

Academic research:

http://www1.mate.polimi.it/~forma/Didattica/ProgettiPacs/BrambillaNecchi15-16/PACS_Report_Pierpaolo_Necchi.pdf

Problem Statement

(approx. 1 paragraph)

I will use reinforcement learning to train an agent that will buy/sell/hold 2 stocks every-day, once a day, so that long term capital increases.

- Agent has an initial capital of $ X.
- Environment has 2 equities that agent can trade A and B.
- Price of these equities are defined as P_A^t and P_B^t.
- Agent can perform tasks: Buy, Sell or Hold for either or both of these equities

Datasets and Inputs

(approx. 2-3 paragraphs)

Dataset is gathered from Kaggle:

https://www.kaggle.com/borismarjanovic/price-volume-data-for-all-us-stocks-etfs/home

Content

The data is presented in CSV format as follows: Date, Open, High, Low, Close, Volume, OpenInt.

I will use following columns from data set: Date, Open, Close and Volume

The data is for a period of more than 5 years (various stocks have various data points).

I will be planning to use 2 equities from the data set- say for e.g Google (googl.us.txt) and Unilever (un.us.txt)

When I decide on the number of epochs to use (say 1000), I will divide the dataset in chronological order from a start date D. So:

- Training data will be from Day D to Day D+1000.
- Then I will do a validation from day D+1001 to D+1200
- Test will be D+1201 to D+1400

Solution Statement

(approx. 1 paragraph)

Reinforcement Learning (RL) is a general class of algorithms in the field of Machine Learning (ML) that allows an agent to learn how to behave in a stochastic and possibly unknown environment, where the only feedback consists of a scalar reward signal . The goal of the agent is to learn by trial-and-error which actions maximize his long-run rewards. However, since the environment evolves stochastically and may be influenced by the actions chosen, the agent must balance his desire to obtain a large immediate reward by acting greedily and the opportunities that will be available in the future. Thus, RL algorithms can be seen as computational methods to solve sequential decision problems by directly interacting with the environment.

As stated above, to make the model easy, my agent will only trade in 2 equities, say A and B. It can take either long or short position in either or both of the 2 stocks.

- It can only buy a stock (either A or B or both) if it has sufficient amount to buy the stock, so no leverage
- It can only sell the stock (A or B or both) if it has these stocks in its portfolio

Using reinforcement learning, the agent will decide when is it right moment to buy, hold or sell stock A or Stock B.

Benchmark Model

(approximately 1-2 paragraphs)

Benchmark model can be a model where the agent for the training period:

- Buys stocks A and Stock B, with half amount invested in each stock
- Holds these stocks for the whole period
- Sells these stocks at the end of the training period

Evaluation Metrics

(approx. 1-2 paragraphs)

Here are a few of the most basic metrics that traders are using. I will use the same for my evaluation:

Net PnL (Net Profit and Loss)

Simply how much money an algorithm makes (positive) or loses (negative) over some period of time, minus the trading fees.

$$PnL = (\text{Portfolio Value})^{t+1} - (\text{Portfolio Value})^{t}$$

Alpha and Beta

Alpha defines how much better, in terms of profit, your strategy is when compared to an alternative, relatively risk-free, investment, like a government bond. Even if your strategy is profitable, you could be better off investing in a risk-free alternative.

$$\text{Alpha} = R - R_f - \text{beta}\ (R_m - R_f)$$

Where:

R represents the portfolio return

R_f represents the risk-free rate of return

Beta represents the systemic risk of a portfolio.

R_m represents the market return

Beta is closely related, and tells you how volatile your strategy is compared to the market. For example, a beta of 0.5 means that your investment moves $1 when the market moves $2.

1.Obtain the weekly prices of the portfolio

2.Obtain the weekly prices of the market index (i.e. S&P 500 Index)

3.Calculate the weekly returns of the portfolio

4.Calculate the weekly returns of the market index

5. β is ratio of these 2 returns

Sharpe Ratio

The Sharpe Ratio measures the excess return per unit of risk you are taking. It's basically your return on capital over the standard deviation, adjusted for risk. Thus, the higher the better. It considers both the volatility of your strategy, as well as an alternative risk-free investment.

Sharpe ratio = (Mean portfolio return − Risk-free rate)/Standard deviation of portfolio return

Project Design

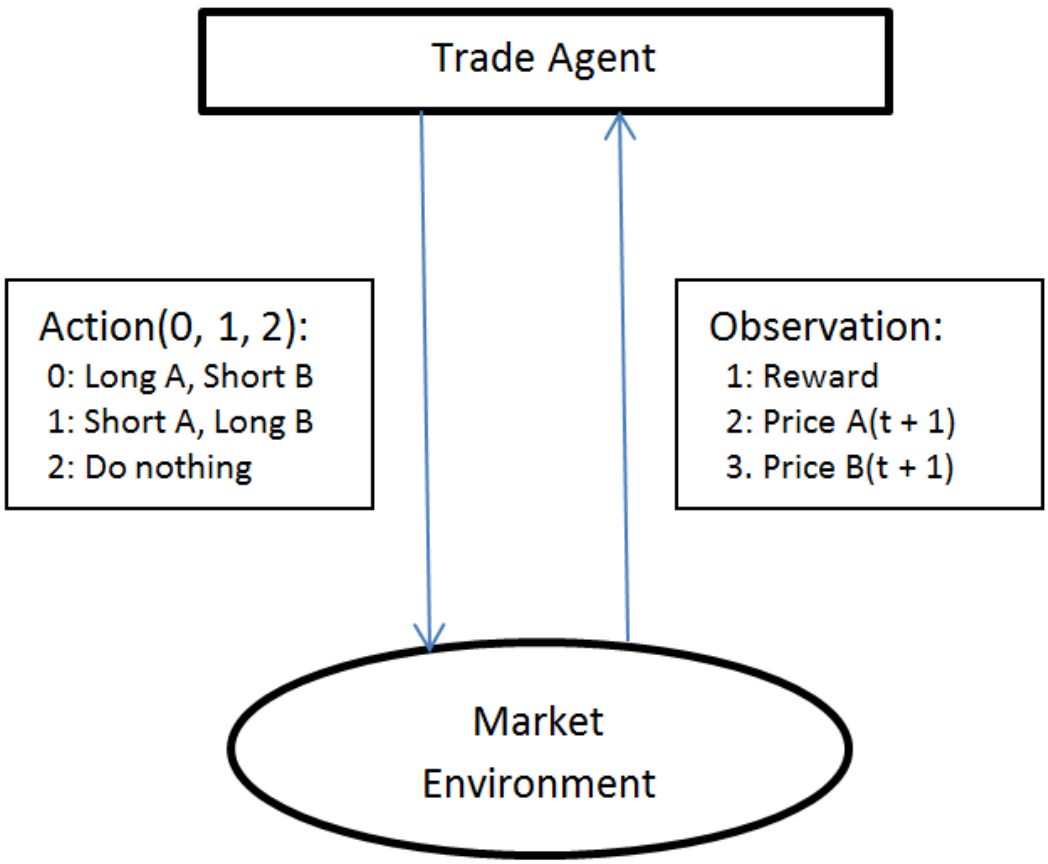
(approx. 1 page)

Please find the below steps in sequence to build the algorithm by reinforcement learning.

1. Data Analysis and visualization: Download the Dataset from Kaggle and do visualization and pre-analysis of data
2. Select 2 stocks as input for the model
3. Define Markov Decision process parameters:
    a. State (2 stocks, and their Prices),
    b. Actions (Buy, Hold, Sell stocks),
    c. Reward Function (This is described in next point)
    d. State transition matrix
    e. Decay factor: Defined between 0 and 1
4. Environment: This class will be designed with following methods:
    - Init : For initialization of the environment at the beginning of the episode.
        - State: Holds the price of A and B at any given time = t.
    - Step: The change in environment after one time step. With each call to this method, the environment returns 4 values described below:
        - next_state: The state as a result of the action performed by the agent. In our case, it will always be the Price of A and B at t = t + 1
        - reward: Gives the reward associated with the action performed by the Agent.
        - done: whether we have reached the end of the episode.
        - info: Contains diagnostic information.
    - Reset: To reset the environment after every episode of training. In this case, it restores the prices of both A and B to their respective means and simulates new price path.

5. Reward Function: This can be simple PnL, profit realized in a day or can be complex based on Sharpe ratio. It is very important to have right reward function
6. Agent can perform tasks buy, hold or sell stocks so that long term reward is maximised.

The agent is a MLP (Multi Layer Perceptron) multi-class classifier neural network taking in two inputs from the environment: Price of A and B resulting in actions : (0) Long A, Short B (1) Short A, Long B (2) Do nothing, subject to maximizing the overall reward in every step. After every action, it receives the next observation (state) and the reward associated with its previous action. Since the environment is stochastic in nature, the agent operates through a MDP (Markov Decision Process) i.e. the next action is entirely based on the current state and not on the history of prices/states/actions and it discounts the future reward(s) with a certain measure (gamma). The score is calculated with every step and saved in the Agent's memory along with the action, current state and the next state. The cumulative reward per episode is the sum of all the individual scores in the lifetime of an episode and will eventually judge the performance of the agent over its training. The complete workflow diagram is shown below:



7. Constraint: Agent has to buy only with amount of money it has and sell only the stocks it has (no short selling).

8. Select optimal policy for sequential decision problem using value iteration or Policy iteration. The policy for next action will be determined using Bellman Ford Algorithm as described by the equation below:

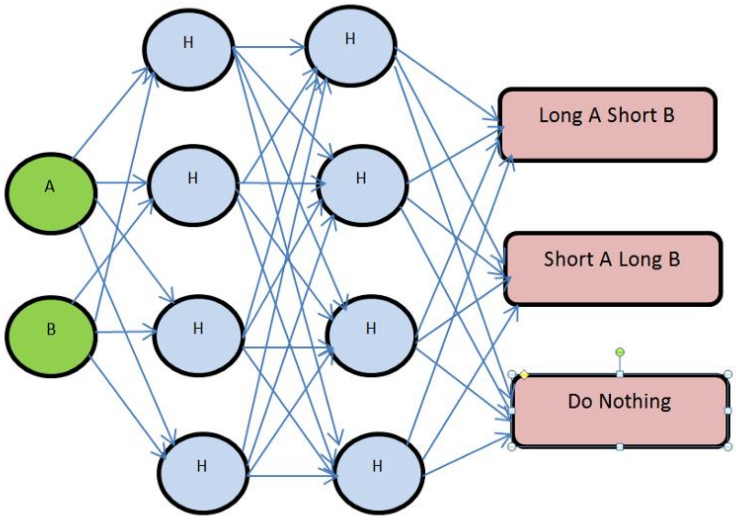
$$Q(s, a) \sim r + \gamma * Q'(s', a')$$

s: Agent's current state
a: current optimal action
γ: discount factor
r: Immediate reward from Q->Q'
s': next *optimal* state
a': *optimal* action in the next state

Through the Q-value, agent will also appreciate the long term prospects than just immediate rewards by assigning different Q values to each action. This is the crux of Reinforcement Learning. Since the input space can be massively large, we will use a Deep Neural Network to approximate the Q(s, a) function through backward propagation. Over multiple iterations, the Q(s, a) function will converge to find the optimal action in every possible state it has explored. Speaking of the internal details, it has two major components:

1. Memory: Its a list of events. The Agent will store the information through iterations of exploration and exploitation. It contains a list of the format: (state, action, reward, next_state, message)
2. Brain: This is the Fully Connected, Feed-Forward Neural Net which will train from the memory i.e. past experiences. Given the current state as input, it will predict the next optimal action.

To train the agent, we need to build our Neural Network which will learn to classify actions based on the inputs it receives. (A simplified Image below. Of course the real neural net will be more complicated than this.).



In the above image,

Inputs(2): Price of A and B in green.

Hidden(2 layers): Denoted by 'H' nodes in blue.

Output(3): classes of actions in red.

For implementation, I will be using Keras and Tensorflow.

9. Based on agent's performance change the reward function (if needed) and fine-tune hyper-parameters of the neural network. The neural net is trained with an arbitrarily chosen sample size from its memory at the end of every episode in real-time hence after every episode the network collects more data and trains further from it. As a result of that, the Q(s, a) function would converge with more iterations and we will see the agent's performance increasing over time until it reaches a saturation point.

Once the training and validation step is completely done, then I will check the model performance on test data (where Epochs were not earlier run on) and visualize performance of agent.

I hope that with above approach, I will be able to create my own reinforcement learning trading agent !!