Tradesimple

Date : April 14, 2021

Salim Eradi Shubham Sharan Stephanie Wang

Introduction

You are a new investor and you are not too familiar with the stock market. You don't want to lose all your money and you are not risk-averse to the point that you are better off opening a high interest saving account. You decide to choose a stock, but you notice its prices fluctuate every day due to demand and supply. That volatility affects your state of mind, provoking impulsive buying or selling. This means you need to constantly keep track of current trends and news which can be tedious and time consuming. Knowing when to buy and sell can be tricky? How do we do it :

The AI identifies when to buy and when to sell by:

- Providing a singular stock (TSLA) of a company because it simplifies the implementation of our agents without compromising the accuracy of the prediction
- Using the current sentiment based on news chatter to predict if the stock will go up or down in price.
- Using historic data to predict trends that contributes towards analyzing the risk vs return potential. (e.g. is the stock doing better/worse over time?)

We want our AI to best predict when to buy and sell the stock to maximize our return which is our profits. Atimes these are highly dependent on positive and negative news/trends in the media since historic data does not completely represent the future outcomes. Our goal is to maximize the confidence in the future return through deep reinforcement learning.

Motivation for the problem

Identifying future prospects of a company is hard to grasp from past endeavors only. The earliest signs come in the form of chatter in the media and amongst individuals. As students in our early 20's, we can leverage the power of compounding. This is a challenge that reinforcement learning agents can resolve since they can process larger quantities of data to make more informed decisions in comparison to a novice trader. The project is significant because there is a large group of new investors who are starting to invest and are either intimidated by the stock market or make poor buy/sell decisions. We are providing a program that will reassure the users that their invested money will be able to grow over the years.

Description of method(s) from artificial intelligence used

Bag of Words for Natural Language Processing

The Bag-of-Words (BoW) model is an easy way to represent text documents as features for training machine learning algorithms. It enables classifiers such as the Naive Bayes Classifier to process natural language. The model uses an array of words (aka the bag of words) and tries to identify each word's weight. BoW calculates the term frequency (tf) and inverse document frequency (idf). The term frequency is the number of occurrences of a particular word in a single document. The inverse document frequency is the score of how meaningful a word is. It compares the word to all the documents provided. The rarer the word, the higher the score. Common words including “a” and “the” will have smaller weights as they are not as meaningful to understanding the sentence. This is an important method since the computer will gain knowledge on which words represent a positive sentence and which represent a negative sentence.

$$tf_{t,d} = \frac{n_{t,d}}{Number\ of\ terms\ in\ the\ document},\ where\ t = term,\ d = document$$

$$idf_t = log(\frac{number\ of\ documents}{number\ of\ documents\ with\ term\ t})$$

Naive Bayes Classifier

Supervised Learning uses labeled datasets to train algorithms to classify data that it has not encountered. The purpose is to learn a function that can predict the label given an input by creating a matrix that associates various inputs to corresponding labels. The value pairs will be a comment provided by the dataset and an integer indicating the sentiment. This will allow the algorithm to classify whether a comment is positive, neutral or negative depending on the features extracted. Different weights are distributed to each feature. For instance, if two features were extracted from a comment and the feature labeled as positive has more weight, then the comment will be classified positive. This method is very useful when it comes to training the agent to develop an accurate feature matrix by assigning the appropriate weights to words.

$$Pair(X, Y)\ where\ X = input\ comment,\ Y = \{-1, 0, 1\}$$

$$f(X) = Y$$

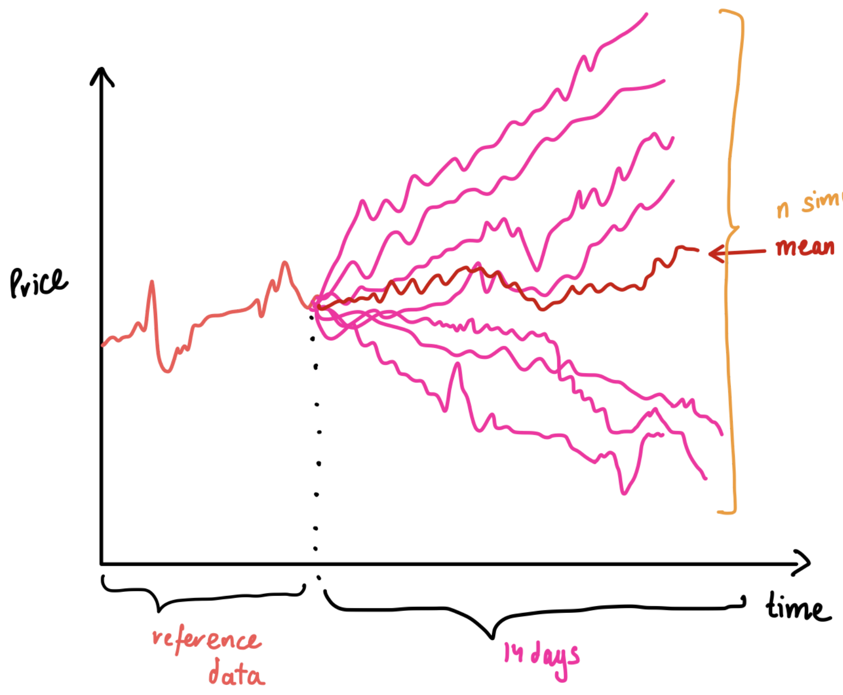
Labels	
1	positive
0	neutral
-1	negative

Naive Bayes Classifier uses conditional probability and applies the Bayes’ theorem with the assumption of strong independence between probabilities. Using the matrix created from feature extractions as evidence, the classifier predicts the hypothesis, being the sentiment of the provided sentence. With the help of the weights representing the probability, the classifier can make a prediction for each word in the sentence.

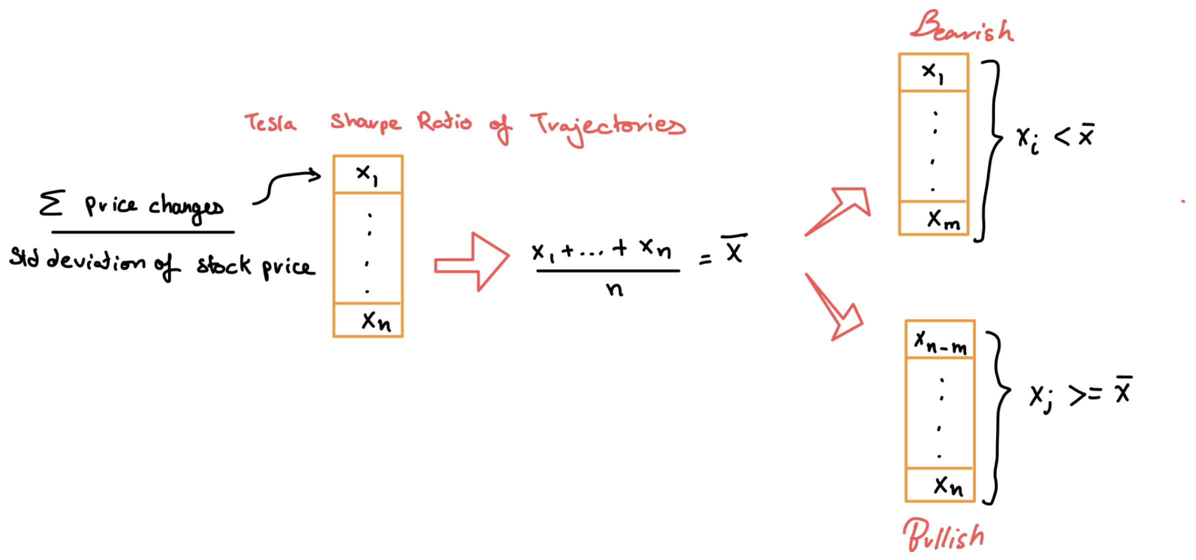
$$P(A|B) = \frac{P(B|A)\ P(A)}{P(B)}$$

Monte Carlo Simulation

Monte Carlo simulations are a great way to simulate future prices of an asset based on some past data as reference. So we end up with a bunch of future simulated stock trajectories from which we can determine the risk adjusted expected return called a sharpe ratio. Stock trajectories are generated by adding a $shock = max\ (0,\ std\ deviation\ of\ reference\ data)$ to the previous day's price. We also incorporate a drift factor which allows us to account for price movements caused by delayed public reaction or sudden changes.

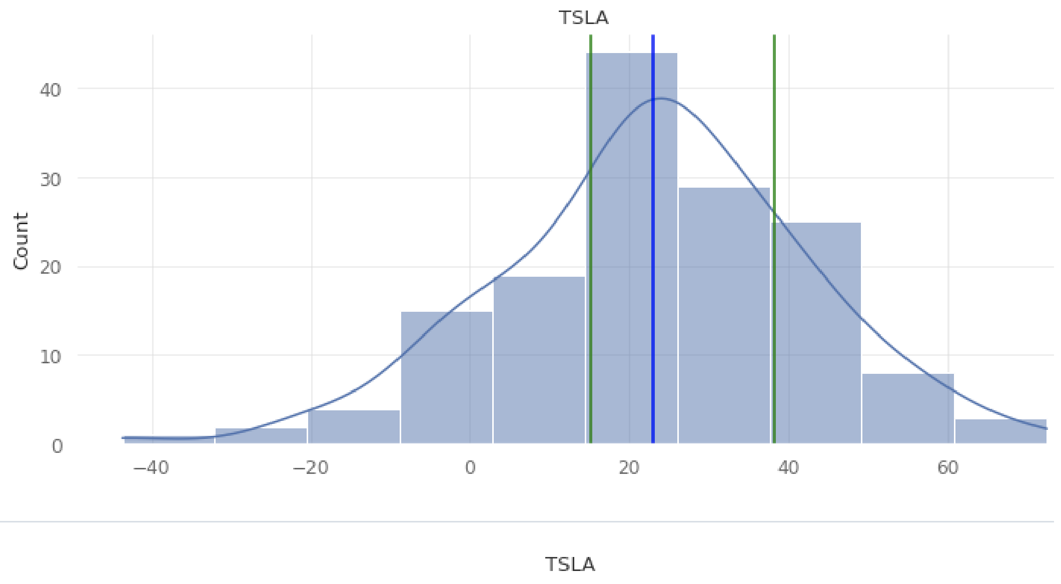


Once we have simulated n trials of our stock price, we will be computing the risk adjusted return of each trajectory and organise it in a sorted series. We split the trajectories into 2 frames; "bearish": whose sharpe ratio is less than the mean and "bullish": more than the mean to be .


Tesla Sharpe Ratio of Trajectories
Σ Price changes / Std deviation of stock price
x_1 ... x_n
$\frac{x_1 + \dots + x_n}{n} = \bar{x}$
Bearish
x_1 ... x_m
$x_i < \bar{x}$
x_{n-m} ... x_n
$x_j >= \bar{x}$
Bullish

However, we have modified this traditional definition of "bear" and "bull" as the simulation may not show a trajectory with a return of less than 0. Which is why we use the mean to get a sense of how much it might deviate from its original trajectory in a positive or negative manner.

By picking one randomly above the mean and one under, we will just return their respective sharpe ratios. Now why random? Let's look at an example:



The more prominent trajectories with specific sharpe ratios will show up more frequently hence the probability of picking a trajectory from a bin that has a larger count is higher. In one iteration the following sharpe ratios were picked. Also note the distribution is not centered at zero. The reference data provided is for the past 14 days. We hope the random pick is close to the true value that we might encounter.

Proximal Policy Optimization (PPO)

PPO uses the policy gradient method for reinforcement learning. PPO uses an on policy method as it picks the actions and follow's it own policy, and it does not store past experiences. It learns directly from what the agent encounters in our environment. It is trained on a batch of trajectories and when the update is made that information is discarded. The idea is to update the policy in a manner such that the new policy is not drastically different from the old one. PPO uses a clipping technique to avoid large changes and avoid being greedy. So the agent performs well not only for a sample it was trained on, but also for an unencountered instance. Our policy gradient loss function looks like this:

$$L^{CLIP}(\theta) = \widehat{E}[min(r_t(\theta)\, \widehat{A}_t,\ clip(r_t(\theta),\ 1 - \varepsilon,\ 1 + \varepsilon)A_t)]$$

Let's explain some of the notation. Where θis the policy parameter, r_t is the ratio of the probability under the new and old policy which can be computed by:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)}$$

It penalises changes to the policy where $r_t(\theta)$deviates from 1. Hence, we clip with anεfactor which clips the probability ratio provided by the equation above, making it constrained in the interval of $[1 - \varepsilon, 1 + \varepsilon]$. Here $\pi_\theta$is the policy which is a neural net where we use an ANN.

Another factor to be discussed is the estimated advantage at time t denoted by $A_t$which is computed with respect to the discounted rewards (what we know happened) and a value function (what did we expect to happen). Subtracting the two components results in the estimated advantage.

- **Discounted rewards:**

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

  Where $\gamma \in [0, 1]$ where 0 is prioritizing short term and 1 for long term. PPO requires little to no hyperparameter tuning, but still provides good results. Changing the value gamma allows for a focus on short-term goals for day traders (closer to 0), as well as long-term goals for growth investors (closer to 1).

- **Value Function:** Simply tries to estimate the discounted return from a specific time period onwards or the final reward at the end. The value function is represented as a neural net, which is updated using the experience it acquires with every iteration as this can be termed as a form of supervised learning where states are imputed and the neural net tries to predict what the discounted sum of reward could be.

  In our implementation we specify a policy model for PPO to use, and we will be using a Multi Layer Perceptron which is basically a feed forward ANN. This multi-layer perceptron uses a supervised learning technique called back propagation.

Our estimated advantage or the so-called advantage estimate value can be negative or positive.

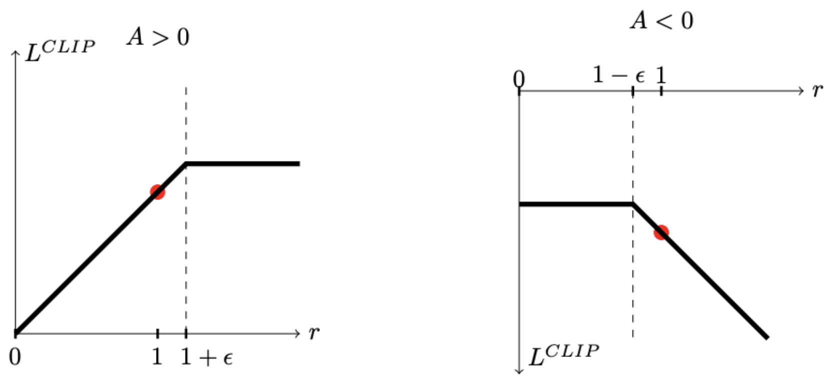


Image 1: clipping when the value is plus and when negative of a single time step[1]

[1] https://arxiv.org/pdf/1707.06347.pdf

L^{CLIP} is just a probability ratio.

Left Image
The clipping here takes place when the action taken is more probable with current policy as compared to the old one, so we clip it at a higher point where the gradient turns to zero. Why we don't want it to exceed this threshold is because we are just working with the sample and not the entire population of data. **As it goes up it becomes more probable.**

Right Image
Here we clip it when it comes close to zero, as the action under the current policy is not favorable, but we want it to be in a position where it is still able to take this action. **As it goes up it becomes less probable.**

Now let's break down the whole equation as we know it. Firstly we have an expected value tying it all up as we will be computing this over a batch of trajectories. This is taken over 2 terms:

- $r_t(\theta)\ \widehat{A_t}$: the normal policy gradient objective that promotes actions to be taken that yield a high positive advantage over the output from the value function.
- $clip(r_t(\theta),\ 1 - \varepsilon,\ 1 + \varepsilon)\ A_t$: the clipped version of normal policy gradients objective

The clipped value will not always be chosen. When A < 0 the gradient will tell us to walk the other direction and make the action less probable and this is when the unclipped normal policy gradient objective has a lower value than clipped version. In the end we get a likelihood as an output for each action which helps us decide which action is to be taken. That action is then reflected in our environment. Our environment utilizes that action to process the reward awarded at every step.

Description of the Implementation

Data Scraping

Overview

Web scraping is a method on how to extract data from websites to a json file. Importing information from websites allows for further analysis of data. In our scenario, we want to scrape information that consists of the stock ticker symbol (e.g. "TSLA"), the appropriate date within a two year time interval, and a small snippet of the article pertaining to that stock on that day. We scanned roughly 10 posts a day using Google News API.

The first consists of the training data which ranges from 09/31/2016 - 09/31/2018 to help train the NLP agent. The second consists of prediction data which ranges from 10/31/2019 - 10/31/2020 to help predict an upward or downward trend in the stock. We analyzed over 5,000 data points for training data and prediction data

Code Implementation

Libraries

1. I utilized three different libraries/API: “GoogleNews”, “json”, ”newspaper”.

Utilizing Google News API

2. Finding posts about a given stock, using the Google News API.
3. Having a loop iterating through the first 10 posts on that particular day.
4. Appending the results to an array in order to extract relevant information, since we only want specific pieces of data.

Creating JSON String

5. After extracting specific information needed such as the date, stock title and the comment.
6. We create a dictionary with the following information extracted by using the results array that is then converted into a json file.

Sentiment Analysis

Overview

The agent being implemented is a model-based reflex agent. The agent maintains an internal representation of the environment. The environment represents all the words the agent will be exposed to. It keeps track of a feature matrix with weights of each word in the corpus document. The various weights represent features of how likely a sentence has a positive, neutral or negative sentiment. The agent will return the predicted sentiment of a sentence based on the feature matrix (environment state) and the given sentence (current precept).

Some python libraries used to help facilitate the implementation of the NLP agent:

- **spaCy**
  A python library for natural language processing. The library is used to classify words and/or punctuations in a sentence into universal part-of-speech (POS) tags. These tags will be used for feature extraction to produce a sparse matrix used for classification.
- **Sklearn**
  A python library for machine learning. The library is used to create the Naive Bayes Classifier to help identify which category a comment belongs to; positive, neutral or negative under the assumption of a normal distribution.
- **VaderSentiment**

A python library with a rule-based sentiment analysis tool. Since there are no labeled datasets that represent comments about the stock market accurately, the library will be used to automate the creation of a labeled dataset for demonstration purposes.

Code Implementation

Creation of Dataset

Input: *string of file path containing json objects separated by line*
Output: *array of labeled comments*
The function reads the file provided by the argument and parses each line representing a JSON string. Only the comment is used for the analysis and training. The function uses the vaderSentiment library to create an analyzer to determine the sentiment of the sentence. The compound value provided by the analyzer is a metric that calculates the sum of all the lexicon ratings which have been normalized between -1 (negative) and 1 (positive). Values less than -0.5 will be considered as a negative sentence whereas values above 0.5 are considered as a positive sentence. Values in between will be considered as a neutral sentence. The final result will be appended to the comment as an integer.

Feature Extraction

Input: *array of labeled datasets for all documents*
Output: *sparse matrix (2D array) of features, string array of unique words (bag of words)*
The function attempts to extract all features and return a sparse matrix representing the association of features and comments. The rows represent each comment in the corpus document and the columns represent the features such as nouns and term frequency. Since each comment may not contain all features, the matrix will be sparse.

The bag of words is created by identifying all unique terms given by the corpus document from the labelled dataset. These unique terms are what composes the bag of words. If a word is being processed, but the word is not in the vocabulary, then the word is unknown. Since the agent has no weight for the unknown word, then it cannot provide any useful information for the sentence. Unlike the n-gram model that can predict letters, there is no way of predicting the weight of the word because the agent is not aware of the meaning. Therefore, the unknown words will be ignored for the prediction.

Once all unique terms are compiled, the feature matrix can be built by looping through each comment. For each word in each comment, the term frequency and inverse-document frequency is calculated and added to the feature matrix. In addition, some common sentence syntax are also counted as part of the matrix such as nouns, verbs and punctuations. This can potentially be important to identifying the sentiment of a sentence. It is important to note, the final column of the matrix represents the label of the true value provided by the labelled dataset.

Prediction

Input: *array of features associated to comments, array of labels, array of comments for prediction*

Output: *array of integer representing the predictions*
The sparse feature matrix is passed to the Naive Bayes Classifier. With the help of supervised learning, different weights are placed to each feature for the possible outcomes (-1, 0, 1). Since the matrix shows the features for each sentence, and each sentence is labeled, the classifier can add more weights to these particular features to the provided label. These weights represent the probability of a feature existing in a positive or negative sentence. The classifier uses the feature matrix as evidence to predict the sentiment of a sentence (hypothesis). When the weights are all assigned appropriately to each feature, the agent is ready to predict any set of sentences with the use of the Naive Bayes Classifier.

Monte Carlo Simulation Agent

The Monte Carlo simulation utilized a model-based reflex agent, where future trends are built on top of its internal representation of the environment. This representation helps us to predict possible future trends.

Function : def monteCarloSim(ticker, num_days, simulation_count,start, end):
Here we have 5 input variables:

- ticker: what stock are we concerned with
- num_days: How many days in the future are we concerned with.
- simulation_count: How many trials will we be running.
- start: start date of the reference data provided
- end: end date of the reference data provided

Implementation :

```
df['Bearish Sharpe Ratio'].loc[i], df['Bullish Sharpe Ratio'].loc[i] =
monteCarloSim("TSLA", 14, 150, start, i)
```

We pass in the last 14 days as reference data.

Step 1: Extract the opening stock prices that you might buy or sell on.

Step 2: Compute the volatility which is merely the standard deviation of the price differences of the stock prices. Pct_Change is the function we are utilising that is looking at the previous price to determine the difference over current price.

Step 3: We identify the last price, which is our start point or the first value in our output from Step 1. We also compute a drift factor using this equation:

$$Drift = Average\ Daily\ Return - \frac{Open\ Price\ Variance}{2}$$

This will be utilised as features when calculating the future price.

Step 4: We just need to prepare the data frame which houses all the future prices of different trajectories. Since trading only happens on weekdays, 14 refers to the next 14 trading days determined by start-end date difference.

Step 5: The first loop runs for the number of trajectories you are expecting to have. For each simulation we initiate a list which we will append the prices to. The loop nested inside runs for the number of days you are predicting the price for into the future. From here we compute a random shock for each day, this is a random number in the range from 0 to our standard deviation (volatility). So to determine the current price we use the following:

$$price = max\,[0,\ (lastprice \times e^{(drift\ + shock\,)})]$$

These prices are added to the specified trajectory and combined into a dataframe with all trajectories.

Step 6: We compute the sharpe ratio for all trajectories. From which we return the sharpe ratio of one trajectory that is greater then the mean and one smaller.

Proximal Policy Optimization

Overview

The proximal policy optimization utilizes a goal based agent, which determines the best action to maximize long term rewards/ profits. Since we are collecting a batch of trained trajectories we want to update its decision-making policy to choose the best action to buy and sell which determine if we take a long (stock goes up) or short position (stock dips). Our reinforcement learning agent is utilizing the data passed from the NLP agent and Monte Carlo Simulation. From which it learns what is the best action to take, so either buy (1) or sell (0); these are the discrete set of actions our agent can take. The reinforcement learning agent brings all attributes of our study together. We will be creating 4 custom environments each with its own unique set of signal features. As shown in the table below:

Table 1

	Model	**RL Stock Agent's Signal Features**	**NLP Agent**	**Monte Carlo Sim**	
		Price	**Sentiment**	**Bearish (Sharpe Ratio)**	**Bullish (Sharpe Ratio)**
Only past	Random	X			
	1	X			
Uses	2	X	X		

future and past	3	X		X	X
	4	X	X	X	X

Libraries

import gym_anytrading : this is the primary library to create an environment in which our agents will be running in. It comes with a bunch of attributes we will be exploiting when making our custom environment

from gym_anytrading.envs import StocksEnv: This is the environment we will be inheriting properties from. We will manipulate these attributes to form a custom environment for our ablation study.

from stable_baselines3 import PPO: Stable baselines is a library of reinforcement learning algorithms from which we will be using the PPO algorithm.This works really well with the gym environments as they are both developed by OpenAi.

Code Implementation

The initial step is to merge all data points into one dataframe. This dataframe will have the following columns:

- Index(['High', 'Low', 'Open', 'Close', 'Volume', 'Adj Close', 'Bearish Sharpe Ratio', 'Bullish Sharpe Ratio', 'Result'], dtype='object') - **Result is the sentiment column!**

For this study we will only focus on: 'Open', 'Bearish Sharpe Ratio', 'Bullish Sharpe Ratio', 'Result'

Common traits and differences between all Model Environments 1,2,3, and 4

- **Class Function CustomEnv:** We are simply inheriting some of the properties of the stocks environment. Initialised as:CustomEnv**{**1,2,3 or 4**}**(df=joined_df, frame_bound = (5, 55), window_size = 5). The parameters are:
    - **df:** The data frame has all the signal features we care about in our models.
    - **frame_bound:** A tuple which specifies the start and end of df.
    - **window_size:** This is the partial observable aspect of the environment as it has no clue what may happen in the future and uses the past 5 days as reference.

- **Functions Add_signals:** every model has its own add signals. The difference is the signals that are being passed into the reinforcement learning agent depending on which model we are focusing on. Our add_signals function returns the tuples of signal features and the prices. We pass the function in the CustomEnv's _process_data parameter so we don't have to use the native parameters the StocksEnv ships with:
    - **start:** `env.frame_bound[0] (5) - env.window_size  (5)` , The batch size of the trajectory that is visible to our agent is for the past 5 days of prices. A week's worth of stock price data, and we start from the first row.

- **end:** `env.frame_bound[1]` so we end at the last index
- **prices:** We will be focused only on the opening price, as that is the price the market opens at. This is what our profits will be computed with. The goal is to buy at low price and sell at high price, as to maximise profits.
- **signals:** This is the only feature that varies across all models as we keep the various features of interest to us in that specific model. Features like sentiment and sharpe ratio of a bullish (the stock goes up) and bearish (the stock goes down) scenario.

- **Reinforcement Learning :** Since, PPO can work relatively well without any hyperparameter tuning we will run it with the base function[2]:
  1. *class* **`stable_baselines3.ppo.PPO`**(*policy, env, learning_rate=0.0003, n_steps=2048, batch_size=64, n_epochs=10, gamma=0.99, gae_lambda=0.95, clip_range=0.2, clip_range_vf=None, ent_coef=0.0, vf_coef=0.5, max_grad_norm=0.5, use_sde=False, sde_sample_freq=- 1, target_kl=None, tensorboard_log=None, create_eval_env=False, policy_kwargs=None, verbose=0,* ***seed=None****, device='auto', _init_setup_model=True*)
     We will provide a policy and the environment in which to run the policy. No hyperparameters are altered and we do test it with different seeds. The policy we will be using is a multi layer perceptron which is simply a feed forward ANN. This multi-layer perceptron uses a supervised learning technique called back propagation for the learning.

  2. **`learn`(*total_timesteps*,** *callback=None, log_interval=1, eval_env=None, eval_freq=- 1, n_eval_episodes=5, tb_log_name='PPO', eval_log_path=None, reset_num_timesteps=True*)
     We will provide the total number of samples to train on. All other hyperparameters will not be altered with. So this is where we want to train our reinforcement learning agent on one set of data and ensure the predictions that we will be making will be on data that it has never encountered before.

  3. **`predict`(*observation*,** *state=None, mask=None, deterministic=False*)
     Here we are only passing in the observation from which it tells us what action to take and the next state *(not used)*. The action that it determined is then thrown into the step function.

  4. Step function is a function that takes an action in the environment. After taking this action it returns our state passed, reward, profit, a boolean that tells us if it has gone through all the data and information we see at each step as follows:
     Information : {'total_reward': 92.70201110839844, 'total_profit': 1.1586723906425247, 'position': 0}

[2] https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html#notes

Step 3 and 4 are in a loop as we iterate through the data points. This total reward tells us that 1 - 1.15867 = 0.15 means we made a profit of 15%.

The results or outcomes achieved

Sentiment Analysis
After training the agent with over 5,000 data points, the results show a 52% accuracy for 2,000 predictions. The results are not as great as expected, but can be justified with the implementation.

Firstly, it was difficult to find labelled datasets that represent the stock market well. When referring to the stock market, specific words are more oftenly used such as buying and selling. These words are important to determine the outcome. The program creates a labelled dataset from web-scraped stock news and a sentiment analysis python library, VaderSentiment. Since the library is well known, we made the assumption that the labels are accurate. Although, the prediction may not be as accurate as the true value if assessed by a human.

Secondly, the bag of words model uses an n-gram of size 1, a unigram. This means that all words are independent of each other and does not consider sentence structure. By modifying the bag of words to use a bigram or even a trigram can potentially provide better results as the queries are more structured. Sentences that use negation for instance might result as a positive sentence because there are more positive words than negative words. In this case, the negation word outweighs all other words, resulting in a false negative.

Finally, the implementation currently ignores all new words during the prediction phase. Since the feature matrix is built using Supervised Learning, the Bag of Words model is limited to only the words seen during the training phase. Any new words exposed afterwards will not be considered to the final result since there are no weights associated with these unknown words. These words can play a crucial part to the final outcome in deciding the sentiment of the sentence. Due to this omittance, the result can be greatly affected.

Reinforcement Learning
This agent incorporates combinations of Monte Carlo Simulation features, Sentiment and Opening stock prices to determine when to buy and sell.

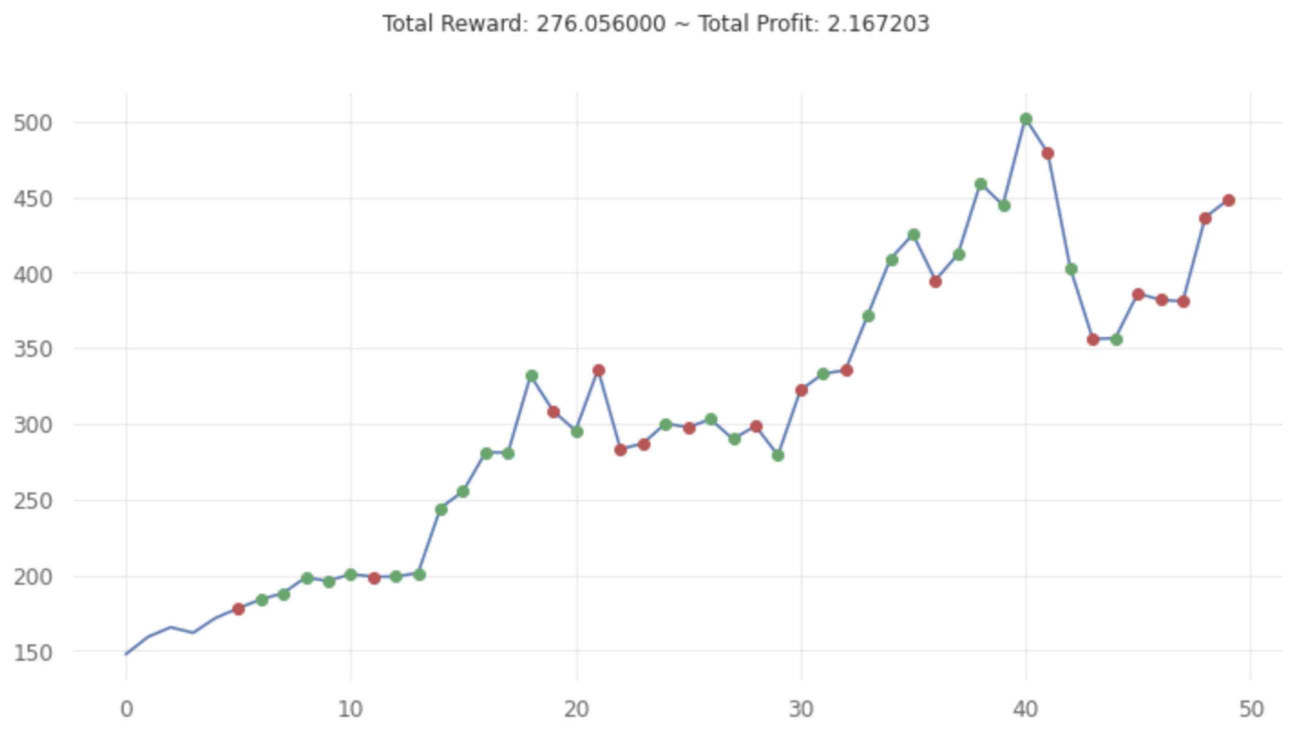
Table 2: time period of the data that is utilized

Agent Type	**Reference Data**	**Predicting from 2019**
MC Simulation Agent	1 year + time elapsed from 100 days	100 days
NLP Agent	Trained over 5000 data points	100 days

RL Agent	Last 5 days of reference	100 days

Table 3				
Model	**Reinforcement Results** *(No hyperparameter tuning)*			
	Trial 1	**Trial 2**	**Trial 3**	**Trial 4**
Random	0.888212		1.0588408	
1	2.167202	1.464201	2.1672026	1.464201684
2	2.171412	1.1496266	1.11240505	1.388048514
3	1.484244	1.7704778	0.98448482	1.106116873
4	1.3371	0.9331202	1.63090406	0.942097853

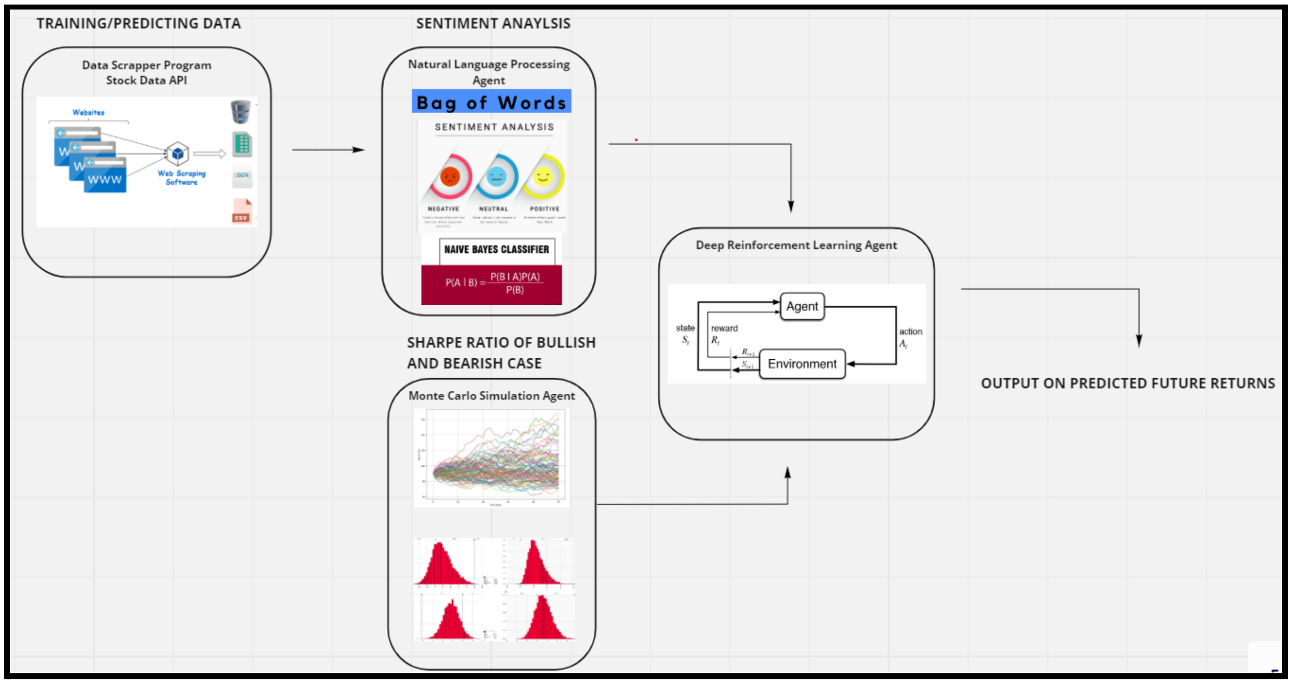
The max possible profit in this environment that can be achieved is 6.363 means we could have made a profit of 536% if bought and sold at the most appropriate times. Our agent visualises it's decision making in the following set of graphs (one shown below):



This image is from trial 3 model 1. Here the red dots say that the stock might fall in value (short) and green to say it might go up in value (long). Not every action taken is perfect but we do take some really good positions. Even though we show only a subset of the results in Table 3, we realised that there was always a model that performed better than model 4 which has all attributes. The model might need more data to train with or a larger window size for reference.

Discussion of the implications of the work

Diagram of Process



Are the conclusions consistent with the results?

Model 4 from Table 3 includes all agents to make the decision of buying and selling. All agents are prone to make mistakes because the agent is predicting with some probability. By including all agents, the final result is more likely to accumulate all errors, hence making the result further from the true value.

The models that use sentiment analysis tend to make the decision of buying more than selling atimes. The results for the NLP agent were skewed towards positive sentiment which associates the action of buying. This can be caused by the NLP agent for not fully understanding the sentence's meaning. The agent does not interpret the sentence properly because the intent may not be captured by sentiment alone. This conclusion is consistent with the way the NLP was implemented as a unigram which does not interpret sentence structure.

Are results/outcomes analyzed with respect to the objective of the project?

The hypothesis of our project was to understand if aspects like sentiment and forecasted trends will lead to better buy/ sell decisions with greater confidence. Purely because they give us a glimpse into the future as to where the stock prices might end up as past stock prices are only an indication of past performance. Hence the ablation study allows us to draw direct comparisons. It seemed though predicting these future actions is quite hard. Firstly, past data is more readily available and easy to process for any type of traders rather than to extract information from probable future indicators. Our model 1 does consistently well as compared to the ones which utilised future indicators but time to time the NLP or the Monte Carlo models do better but lack consistency. Our objective of being able to be better informed has not been completely automated but definitely does provide more insight to aid in decision making.

Directions for future work

There are many ways our project can be used for the future in ways that involve modifying our work to enhance performance, consistently and efficiently.

Our data scraper could have utilised the full article instead of a small snippet, due to API constraints.

Including a larger vocabulary in the Bag of Words could provide a better prediction of the sentiment for the NLP agent. The downside is that it increases the size of the feature matrix since the bag of words would be bigger which would result in a larger consumption of memory. This is the tradeoff taken for the NLP agent, but with more resources, the implementation can potentially be exploited.

We utilized a unigram bag of words model, meaning every word is independent of each other. Looking at every word individually does not provide full sentence structure, but to get better results, a bigram or trigram could be utilized to fix the problem.

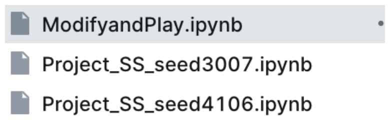
In addition to the NLP agent, unknown words were omitted from predicting the outcome. Since these words can provide crucial information to the final result, they should not be dismissed. Rather than only using Supervised Learning to train the agent, perhaps the agent can incorporate, in addition, Unsupervised Learning method. By using Unsupervised Learning, the agent can discover patterns and information allowing it to predict the meaning of unknown words. With the consideration of the unknown words, the agent can potentially provide a more accurate prediction to the sentiment of the overall sentence.

So far we have been working with only one stock and one media platform, but in reality people prefer to have a diverse portfolio. So we would like to scale up the solution to incorporate multiple stocks and multiple social media platforms to feed data into the NLP agent.

For future work, we could run more simulations in the Monte Carlo Simulation and include more features (volume, pe ratios, etc) that explain the price variances in stocks. So far in our reinforcement learning agent, we did little hyperparameter tuning as it required a more in-depth knowledge of this implementation since reinforcement learning agents are quite sensitive to such parameters. We can explore different policies to identify the optimal one that best suits the problem in hand, we are currently using the MLPPolicy and we would like to explore others including CNN based Policy or even make our own custom policy, as Stablebaselines3 library supports it.

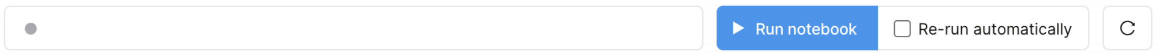
User Manual

1. **Open the deepnote link provided:** executable
2. You can select "Modify and Play" even though all of the 3 notebooks here all have the same code just different seeds:



ModifyandPlay.ipynb
Project_SS_seed3007.ipynb
Project_SS_seed4106.ipynb

3. Just click on the Run notebook, and the code should start running. Note there are some sections which might take longer than expected.



Statement of contributions

Shubham Sharan

- Focused on the implementation of :
  - The Monte Carlo Simulation to provide key features for the reinforcement learning agent
  - The Proximal Policy Optimization and creating the custom gym environment's which incorporate features from the Sentiment Analysis and Monte Carlo Simulation
- Write up for:
  - Monte Carlo Simulation and Reinforcement Learning descriptions and implementation
  - Monte Carlo Simulation and Reinforcement Learning results and outcome
  - User Manual

Stephanie Wang

- Worked on the implementation for the Natural-Language Processing agent which includes the following:
  - Built a labelled dataset in relation to stocks for Supervised Learning
  - Built a vocabulary for the Bag of Words Model to accurately make a decision with the usage of a feature matrix
  - Used the Naive Bayes Classifier to predict the result
- Write up for the AI methods used for the NLP agent, as well as the implementation and conclusion portion of the NLP agent.

Salim Erradi

- Contributed to writing the introduction
- Write up for AI agent for Monte Carlo Agent and Deep Reinforcement Learning Agent
- Implemented web scraper API program to be given to NLP agent which includes prediction data and training data
- Write up for web scraper API program

Appendix

Code Appendix

NLP AGENT: https://deepnote.com/project/9d3bb7f3-9fd2-4050-9ebf-2ac08233f9a5
RL Agent:
https://deepnote.com/project/tradesimple-MZIDbEIFSISz-nbTAJr4sg/%2FModifyandPlay.ipynb
Data Scraping: https://github.com/COMP4106/Data-Scrapper

Research Material Appendix

1. https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html#notes
2. https://pypi.org/project/gym-anytrading/
3. https://openai.com/blog/openai-baselines-ppo/
4. https://www.investopedia.com/terms/m/montecarlosimulation.asp
5. https://arxiv.org/pdf/1707.06347.pdf