University of Mumbai

Bitcoin Investing Assistant

Submitted at the end of semester VI in partial fulfillment of requirements

For the degree of

Bachelors in Technology

by

Md. Ausaf Rashid
Roll No: 1913090

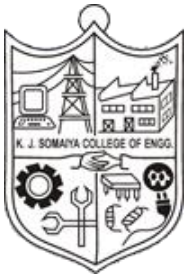
Jatin Nainani
Roll No: 1913093

Nirman Taterh
Roll No: 1913095

Guide:

Ankit Khivasara

Chaitali Kulkarni



Department of Electronics and Telecommunication Engineering
K. J. Somaiya College of Engineering, Mumbai-77
(Autonomous College Affiliated to University of Mumbai)

Batch 2019 –2023

K. J. Somaiya College of Engineering, Mumbai-77

(Autonomous College Affiliated to University of Mumbai)

Certificate

This is to certify that this report entitled **Bitcoin Investing Assistant** submitted by Md. Ausaf Rashid, Jatin Nainani and Nirman Taterh at the end of semester VI of TY B. Tech is a bonafide record for partial fulfillment of requirements for the degree of Bachelors in Technology in Electronics and Telecommunication Engineering of University of Mumbai

_____ _____

Guide/Examiner1 Examiner2

Date:

Place: Mumbai-77

K. J. Somaiya College of Engineering, Mumbai-77

(Autonomous College Affiliated to University of Mumbai)

DECLARATION

We declare that this written report submission represents the work done based on our and / or others' ideas with adequately cited and referenced the original source. We also declare that we have adhered to all principles of intellectual property, academic honesty and integrity as we have not misinterpreted or fabricated or falsified any idea/data/fact/source/original work/ matter in my submission.

We understand that any violation of the above will be cause for disciplinary action by the college and may evoke the penal action from the sources which have not been properly cited or from whom proper permission is not sought.

_____ **Signature of the Student** _____ **Roll No. 1913090**	_____ **Signature of the Student** _____ **Roll No. 1913093**
_____ **Signature of the Student** _____ **Roll No. 1913095**	

Date:

Place: Mumbai-77

Abstract

Trading is one of the most important aspects of commerce in human history.To consistently make profit, traders take every negative or positive trade they make as a learning opportunity. Predicting and analyzing financial indices has been of interest in the financial community for a long time, but recently, there has been a wide interest in the Machine Learning community to make use of and benefit from the more advanced techniques, such as reinforcement learning. Reinforcement learning (RL) has been able to surpass many of the Machine Learning benchmarks in a lot of fields. In this project we explore the plausibility of using RL in an algorithmic trading system. Our approach provides a feature rich environment for the reinforcement learning agent to work on. Firstly, we use the daily closing prices and volume of bitcoin in the market. As our aim is to provide long term profits to the user, we took into consideration some of the most reliable technical indicators. We have also developed a custom indicator to evaluate the current value of the prices. This aims at providing better insights of the Bitcoin market to the user. The Bitcoin market follows the emotions of the traders, so another factor of our trading environment was the overall daily Sentiment Score of the market on Twitter. The agent was tested for a period of 685 days which included the volatile period of Covid-19. It was able to provide recommendations reliable enough to give an average profit of 69%. Finally, the agent was able to recommend the optimal actions to the user through a website. The user can also access the visualizations of the important indicators to help them fortify their decisions.

Keywords: Bitcoin, Cryptocurrency, Reinforcement Learning, Sentiment Analysis, Technical Analysis, Trend trading

Contents

List of Figures and Formulae

1 Introduction

1.1 Background

A cryptocurrency is a digital virtual currency that is secured by cryptography. It uses cryptographic functions to facilitate financial transactions and form a system to store & transfer value. It leverages blockchain technology in order to be decentralized. Its most important feature is that it's not controlled by a central authority like the government or bank to uphold or maintain it. Bitcoin is the first decentralized cryptocurrency, which was released as open-source software in 2009.

A central bank-controlled fiat currency has value because it's issued by a monetary authority and is widely used in an economy. On the contrary, Bitcoin's value doesn't come from a centralized authority, but because of its scarce nature, and select use cases. But, on the most fundamental level, both fiat and Bitcoin (or any other currency) derive their value because of trust. As long as its users trust their government or the protocol, the currency will have value. Let's look at the important characteristics of Bitcoin which helps it derive its value and trust.

Decentralization is one of the key features and motivations behind cryptocurrencies. It gives users complete control over their funds, and since its blockchain is permissionless, anyone can participate in the network, increasing its overall security and adoption. As more and more users and nodes connect to this decentralized network, the more value it gets, and the more secure its blockchain becomes. It's important to note that no single node can take decisions on everyone's behalf.

Bitcoin has a fixed and limited supply of 21 million BTC. The last coin will be mined around 2140 and after that, no new coin will be minted. This characteristic is similar to that of precious metals like gold. As the supply of Bitcoin decreases every few years, and as users lose or burn coins, the supply side consistently decreases over time. Along with that, its adoption among individuals and corporations has been growing. So an increased demand along with a decreasing supply ensures that it's an excellent store of value.

There are almost no other systems or protocols which offer as much security as Bitcoin. If the suitable operational security is followed and the private key is stored safely, it is practically impossible to steal funds or to compromise the system. This is because of the virtue of being highly decentralized and having no single-point high-value targets for the attackers.

1.2 Motivation

It helps give the investor a perspective of the overall market situation and accordingly enables them to plan their investment and allocate an appropriate percentage of their financial portfolio to cryptocurrencies and Bitcoin.

While there are numerous solutions and trading bots that try to generate profits by trading and through short term patterns, there is a need for an investing assistant which tries to maximise profits by considering long term trends and real-time user sentiments.

It is important to note that more than 98% percentage of short-term traders do not make a profit net of fees. This is because it is complicated to time the market and make the right trading decision every time. This is because there are too many variables involved in predicting the short-term price of any asset, more so in the case of Bitcoin since it is one of the most volatile financial assets in the market.

The critical difference between a trading bot and an investing assistant is: In the case of the trading bot, the final trade decision is made by the bot itself, while in the investing assistant, the final investment decision is on the investor.

1.3 Scope of project

In this project, we primarily focus on the price of Bitcoin. This is because it forms around 50%-70% of the total market capitalization of all cryptocurrencies. The price action of Bitcoin generally dictates the overall crypto market condition and investor sentiment. In fact, Bitcoin dominance which is defined as the ratio between Bitcoin's market cap to total crypto market capitalization, is an important metric that indicates the overall greed in the market. It is observed that when bitcoin is in a consistent uptrend, the entire crypto market and all the prominent altcoins follow the uptrend, and when Bitcoin undergoes a downtrend, most of the altcoins see bearish momentum.

Bitcoin is one of the very few cryptocurrencies which has withstood the test of time and consistently been the coin with the highest market capitalization and lowest risk. This is owing to its extent of decentralization, strong open source community support and its first-mover advantage in the crypto space.

Therefore, considering all these factors, we have kept the scope of our project limited to bitcoin's price. That being said, these indicators and recommendations can be useful for providing a perspective on the price of most of the prominent altcoins too.

In the past year, Bitcoin's price has seen some correlation with the traditional stock markets, especially the S&P-500 and Nasdaq Index. In this project, we haven't taken them as a factor to generate a recommendation. It could be argued that as the market cap of crypto increases, it will be subject to the same macro and micro economical factors as the stock market and therefore have some degree of correlation. Also, we have restricted ourselves to the USD valuation of Bitcoin, as that's a world reserve currency

1.4 Brief description of project undertaken

In this project, we have two objectives primarily. The first one is to help the investors visualize the current market with the help of various price indicators vis, RSI, EMA, SMA and Custom Index based on the bull market support band.

The second part involves generating a buy, sell or hold recommendation after taking into account the aforementioned price indicators, the current market position of the investor and the overall sentiment of the users and the investor.

We aim to take a balanced overview of the various price indicators, user sentiments and investor's market position, to generate an optimized recommendation that will maximize risk-adjusted returns over the long term.

1.5 Organization of the report

This report is structured as follows. Chapter 2 introduces the features and problems of current solutions in the field. In Chapter 3, we analyze our objectives and explore the fine details of Reinforcement Learning and Sentiment Analysis. In Chapter 4, we delve into the implementation steps of our system, descripting the data and features used in the process. Finally, in Chapter 5, we present our conclusions.

2 Literature Survey

Behavioural economists like Daniel Kahneman and Amos Tversky established that decisions, even ones involving financial consequences, are impacted by emotion and not just value alone [5] . Dolan et al.'s work in "Emotion, Cognition, and Behavior" further supports that emotions highly impact decision-making [2] . Insights from these researchers open up the possibilities to find advantages through tools like sentiment analysis as it indicates that demand for a good, and therefore price, may be impacted by more than its economic fundamentals. Panger et al. found that Twitter sentiment correlated with people's general emotional state [11] .

Other researchers have specifically studied the efficacy of sentiment analysis of tweets. Pak et al. showed that separating tweets into positive, negative, or neutral categories could result in effective sentiment analysis [10] . Having established that emotions influence decisions, that social media can impact decisions, and that sentiment analysis of social media can accurately reflect the larger population's opinions towards something. Dennis et al. collected valence scores on tweets about the companies in the S&P 500 and found that they correlated with stock prices [14] . De Jong et al. analyzed minute-by-minute stock price and tweet data for 30 stocks in the DOW Jones Industrial Average and found that 87% of stock returns were influenced by the tweets [4] .

With the introduction of cryptocurrencies, similar work has been done to see if such methods effectively predict cryptocurrency price changes. In the paper "Predicting Bitcoin price fluctuation with Twitter sentiment analysis" by Stenqvist et al., the authors describe how they collected tweets related to Bitcoin and Bitcoin prices from May 11 to June 11 in 2017. First, tweets were cleaned of non-alphanumeric symbols (using "#" and "@" as examples of symbols removed). Then tweets that were not relevant or determined to be too influential were removed from the analysis. The authors then used VADER (Valence Aware Dictionary and Sentiment Reasoner) to analyze the sentiment of each tweet and classify it as negative, neutral, or positive. Only tweets that could be considered positive or negative were kept in the final analysis [13]. The sentiment analysis done in this project builds off everything above but is unique, and we solve the problem of giving the tweets a weight.

Isaac et al. [8] used 10 second and 10-minute BTC price data from OKcoin and Coinbase API to accurately predict the bitcoin prices for short term periods. They focused on 25 features relating to the Bitcoin price and payment network over the course of five years. With this, training was done for Binomial GLM, SVM and Random Forests and these were compared with each other through metrics like sensitivity, specificity, accuracy and precision. The random forest model was able to beat all the models in all metrics except precision, where Binomial GLM bested it.

Zhengyao et al. [3] used a trading period of 30 minutes. Their primary focus was on portfolio management rather than price prediction, and the trading experiments were done in the exchange Poloniex. The Reinforcement Learning framework used a deterministic policy gradient algorithm. Agent was the software portfolio manager performing trading actions in a financial market environment. The reward function aimed to maximize the average logarithmic cumulative return R. For policy functions, three different methods were tested. CNN, RNN and LSTM. Performance metrics were: Accumulative portfolio value, Sharpe ratio. Although with a high commission rate of 0.25% in the backtests, the framework is able to achieve at least 4-fold returns in 50 days.

Otabek et al. [12] created a recommendation system for cryptocurrency trading with Deep Reinforcement Learning. Data was taken from cryptodatadownload and focused on three currencies Bitcoin (BTC), Litecoin (LTC), and Ethereum (ETH). The environment was responsible for accounting for stock assets, money management, model monitoring, buying, holding, selling stocks, and calculating the reward for actions taken. While the agent ran every day, observing the environment, to choose an action with the policies learned during the training. The environment monitoring, determining actions with policies, recording and calculating rewards with discounted rewards, computing the gradient, and updating the network of systems with gradients—these are jobs that the agent is responsible for. The experiment on Bitcoin via DRL application shows that

the investor got 14.4% net profits within one month. Similarly, tests on Litecoin and Ethereum also finished with 74% and 41% profit, respectively.

Fengrui et al. [7] regarded the transaction process as actions, returns as awards and prices as states to align with the idea of reinforcement learning. A Deep Reinforcement Learning Algorithm - Proximal Policy Optimization (PPO) was used for high-frequency bitcoin trading. They first compared the results between advanced machine learning algorithms like support vector machine (SVM), multi-layer perceptron (MLP), long short-term memory (LSTM), temporal convolutional network (TCN), and Transformer by applying them to the real-time bitcoin price and the experimental results demonstrated that LSTM outperforms. It was then decided to use LSTM as the policy for the PPO algorithm. The approach was able to trade bitcoins in a simulated environment with synchronous data and obtained a 31.67% more return than that of the best benchmark, improving the benchmark by 12.75%.

Gabriel et al. [1] created an agent that learns to trade the XBTUSD (Bitcoin versus US Dollars) perpetual swap derivatives contract on BitMEX on an intraday basis. The cryptocurrency agent realized a total return of 350%, net transaction costs over five years, 71% of which is down to funding profit. The annualized information ratio that it achieves is 1.46. The echo state network provides a robust and scalable feature space representation.

Joonbum et al. [6] developed an action-specialized expert model designed specifically for each reinforcement learning action: buy, hold, and sell. Models were constructed by examining and defining different reward values that correlate with each action under specific conditions, and investment behaviour is reflected with each expert model. To verify the performance of this technique, the profits of the proposed system are compared to those of single trading and common ensemble systems. In addition, sensitivity was checked with three different reward functions: profit, Sharpe ratio, and Sortino ratio. All experiments were conducted with S&P500, Hang Seng Index, and Eurostoxx50 data. The model was 39.1% and 21.6% more efficient than single and common ensemble models, respectively. From the above literature survey, we can conclude a few points. There is a distinct lack of a consistent environment, leading to some really restrictive while others are too free and ill-defined. Most of the agents are restricted to a single type of market. The variety of preprocessing techniques used led to the question of whether the improvement in the metric was the result of the model or the data fed. Most models are treated as a complete black box with a lot of hyperparameter tuning. Perhaps some form of explainable AI might find some use here to convince the investors and help them understand on what basis our model recommends actions.

3 Project Design

3.1 Introduction

There are numerous challenges involved in building a robust and accurate reinforcement learning model that's a good enough representation of a cryptocurrency market's environment. Cryptocurrency price actions are extremely volatile and depend on a large amount of real-world and statistical factors. We will start by outlining the overall flow and architecture of the model. After that, we will move on to discuss the implementation of various features like Twitter sentiments, technical indicators and the custom index. Finally, all these features will be utilized in the reinforcement learning model. The model will be evaluated using robust evaluation metrics like entropy loss, policy loss and value loss. The final product is hosted on a website for the user to get the recommendations and gain insights from the visualizations.

3.2 Problem Statement

Our problem statement is to maximise risk adjusted returns over the long term, i.e., 2-4 years using periodic investment in Bitcoin. We also need to provide optimal trade action recommendation to the user via a website.

3.3 Block Diagram

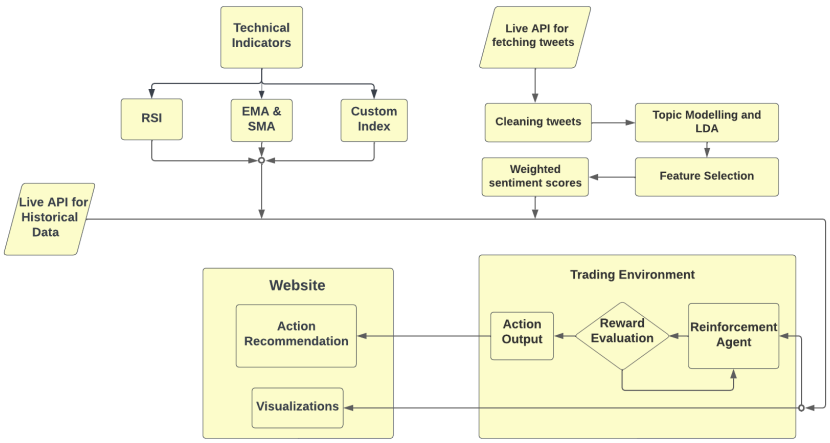


Figure 1: *Flow Chart*

The above diagram describes the flow of the project. The process starts with acquiring historical financial data of Bitcoin. This is achieved with the help of a Live API. Our solution takes novelty in its ability to take a balanced overview of the market situation. Furthermore, it focuses on feeding rich, handcrafted and tested features to the learning algorithm.

The input data is used to calculate popular technical indicators. Specifically, the daily closing prices of Bitcoin are used to evaluate Moving averages, Strength Indices, and our custom index. These indicators are appended to the input data as features.

This is then followed by evaluating the average sentiment score on a daily fidelity. This is done by fetching tweets related to Bitcoin using the Twitter API. From these, the weighted sentiment score is calculated and is then added to the input data as another feature.

The project aims to assist Bitcoin investors and not just provide blind recommendations to them. This is why the project also provides visualizations for the Bitcoin data along with the indicators

and the sentiment scores we evaluated. This would help the user to make their own decisions based on these visualizations and not just follow our final verdict.

After gathering all the features, it is given to our trading environment to simulate bitcoin trades. Our reinforcement Agent acts on this environment to produce optimal recommendations based on the market situation of the environment. Finally, this recommended action is displayed to the user through the website interface.

3.4 Objectives

We aim to maximize risk-adjusted returns over the long term using periodic investment in Bitcoin, leveraging machine learning. We take into consideration Human Generated Data Points(from Twitter), Price data, several technical price Indicators and our Custom Indexes. We also use the overall daily market sentiment as a feature.

We use reinforcement learning to implement trade-bot like architecture using the aforementioned data points and variables as features. The agent recommends the user whether to Buy, Sell or Hold Bitcoin, taking into regard the current market situation. We provide a website to display the recommendation and visualize technical indicators.

3.5 Sentiment Analysis

Just like other assets, the price of Bitcoin is directly related to market supply and demand. The market prices can change for reasons, including public opinion and social media. Hence, analyzing the market's sentiment to predict the potential of Bitcoin is very important. The Market sentiment is the collective attitude of traders and investors toward Bitcoin. Market sentiment does have the power to influence market cycles for Bitcoin. Still, favourable market sentiment does not always lead to positive market conditions.

Social media, especially Twitter, has been the main source of information about cryptocurrencies. Understanding the author's opinion from a piece of text is the objective of sentiment analysis. Most researchers recognize the predictive power of social media sentiment for Bitcoin prices trading over the short (one week) and long span (up to three months). Even the number of tweets made correlates with the trading volume of Bitcoin. Investors tend to overreact to news leading to a market trend where the market shifts initially with the sentiment and is then slowly fixed.

Market sentiment analysis is an essential part of many Bitcoin trading strategies. Like technical analysis, it is usually a good idea to make decisions using a mixture of all the information available. Being able to quickly understand the impact of tweets and the public opinion on price direction can provide useful purchasing and selling insights to cryptocurrency users. Several existing Bitcoin price predictors do not consider 'real time' public opinion to predict prices. Hence, we want to predict whether the user should buy or sell or hold his Bitcoin assets depending if the value of bitcoin will increase or decrease in the subsequent day based on 'real time' public opinion.

We approach this task by collecting Bitcoin Related Tweets with feature engineering, followed by conducting sentiment analysis on tweets. The output of this sentiment analysis gives us average sentiment scores on a daily fidelity which is then passed onto the RL model as a feature.

After scraping the tweets using the API, the first task is performing pre-processing techniques on the tweets. This includes cleaning the tweets, followed by Topic Modeling and LDA, which would help in advertisement and spam handling. Our goal is to apply sentiment analysis to our collected tweets to measure the subjective emotion or opinion of these tweets regarding Bitcoin. Due to our Tweet data set being unlabelled, we explored using Topic Modeling to help automate data labelling. After this, we chose to use VADER for analysis which returns a compound polarity score between 0 to 1. We then perform feature engineering and selection, which would then help to create a weighted sentiment score. To prevent the loss of information by taking the average sentiment, we have generated weights to create weighted sentiment scores.

3.5.1 Latent Dirichlet Allocation (LDA)

LDA is an effective topic modelling technique to extract topics from the given corpus. It is one of the most popular topic modelling methods. Each document is made up of various words, and each topic also has various words belonging to it. The aim of LDA is to find topics a document belongs to, based on the words in it. It assumes that documents with similar topics will use a similar group of words. This enables the documents to map the probability distribution over latent topics and topics are probability distribution.

Let us say we have the corpus with the following five documents:
Document 1: I want to watch a movie this weekend.
Document 2: I went shopping yesterday. New Zealand won the World Test Championship by beating India by eight wickets at Southampton.
Document 3: I don't watch cricket. Netflix and Amazon Prime have very good movies to watch.
Document 4: Movies are a nice way to chill however, this time I would like to paint and read some good books. It's been so long!
Document 5: This blueberry milkshake is so good! Try reading Dr. Joe Dispenza's books. His work is such a game-changer! His books helped to learn so much about how our thoughts impact our biology and how we can all rewire our brains.

Any corpus, which is the collection of documents, can be represented as a document-word (or document term matrix), also known as DTM. After preprocessing the documents, we get the following document word matrix where D1, D2, D3, D4, and D5 are the five documents, and the words are represented by the Ws, say there are eight unique words from W1 to W8.

Hence, the shape of the matrix is 5 * 8 (five rows and eight columns):

Document Word Matrix

	W1	W2	W3	W4	W5	W6	W7	W8
D1	0	1	1	0	1	1	0	1
D2	1	1	1	1	0	1	1	0
D3	1	0	0	0	1	0	0	1
D4	1	1	0	1	0	0	1	0
D5	0	1	0	1	0	0	1	0

Figure 2: *Word Matrix*

The corpus is mainly the above-preprocessed document-word matrix, in which every row is a document, and every column is the tokens or the words. LDA converts this document-word matrix into two other matrices: Document Term matrix and Topic Word matrix, as shown below:

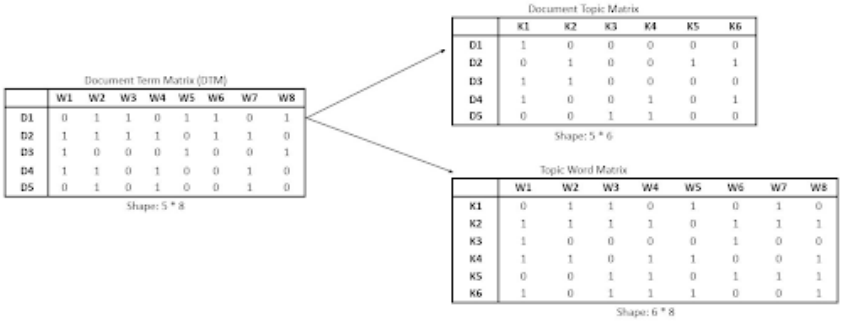


Figure 3: *Topic, term and word matrix*

The Document-Topic matrix already contains the possible topics (represented by K) that the documents can contain. Here, suppose we have 5 topics and had 5 documents, so the matrix is of dimension 5*6. The Topic-Word matrix has the words that those topics can contain. We have 5 topics and 8 unique tokens in the vocabulary; hence the matrix had a shape of 6*8.

3.5.2 VADER

VADER (Valence Aware Dictionary for Sentiment Reasoning) is a model used for text sentiment analysis that is sensitive to the polarity (positive/negative) of emotion. VADER text sentiment analysis uses a human-centric approach, combining qualitative analysis and empirical validation by using human raters and the wisdom of the crowd.

Primarily, VADER sentiment analysis relies on a dictionary that maps lexical features to emotion intensities called sentiment scores. The sentiment score of a text can be obtained by summing up the intensity of each word in the text. Lexical feature means anything that humans use for textual communication. In a typical tweet, we can find not only words but also emoticons like ":-)", acronyms like "LOL", and slang like "meh". These colloquialisms get mapped to intensity values in VADER sentiment analysis.

Lexical features are not the only things in the sentence which affect the sentiment. There are other contextual elements, like punctuation, capitalization, and modifiers which also impart emotion. VADER sentiment analysis takes these into account by considering five simple heuristics. The effect of these heuristics is, again, quantified using human raters.

The first heuristic is 'punctuation'. If we compare "I like it." and "I like it!!!", the second sentence has more intense emotion than the first and therefore must have a higher VADER sentiment score. The second heuristic is capitalization. "AMAZING performance." is more intense than "amazing performance.", so VADER takes this into account by incrementing or decrementing the sentiment score of the word.
The third heuristic is the use of degree modifiers. Take, for example, "hecking cute" and "sort of cute". The effect of the modifier in the first sentence is to increase the intensity of cute, while in the second sentence, it is to decrease the intensity. VADER maintains a booster dictionary which contains a set of boosters and dampeners.
The fourth heuristic is the shift in polarity due to "but". Often, "but" connects two clauses with contrasting sentiments. VADER implements a "but" checker. Basically, all sentiment-bearing words before the "but" have their valence reduced to 50% of their values, while those after the "but" increase to 150% of their values.
The fifth heuristic is examining the tri-gram before a sentiment-laden lexical feature to catch polarity negation. Here, a tri-gram refers to a set of three lexical features. VADER maintains a list of negator words.

VADER sentiment analysis combines a dictionary of lexical features to sentiment scores with a set of five heuristics. The model works best when applied to social media text.

3.6 Reinforcement Learning

Reinforcement Learning is a part of Artificial Intelligence that tries to replicate the learning methods of humans and other animals. Their primary focus of the agents of Reinforcement System is to take actions in an environment to maximize reward.

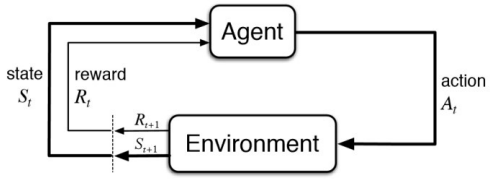


Figure 4: *General Block Diagram of RL*

The Environment is the Agent's universe where it exists and interacts. The agent can interact with the environment by taking some action but cannot influence/ change the rules of the environment through any means.

Beyond the agent (the learner) and the environment, we can identify the main elements of a reinforcement learning system:

- State
- Action
- Policy
- Reward function
- Value function
- Discount Factor

The state defines a set $\boldsymbol{S}$ of agent states that it can observe. An action defines a set of possible actions $\boldsymbol{A}$ that can be performed in a state $\boldsymbol{S}$. A policy defines how the learning agent behaves at any moment. In other words, it is a mapping of the states of the environment to the actions to be taken in those particular states. It is the heart and brain of an agent because it determines the primary behaviour. A reward function defines the goal in a reinforcement learning problem. It aims at mapping each state of the environment to a unique number, a reward, indicating the desirability of that state. For many problems, the consequences of an action are not immediately apparent but only after several other actions have been taken. The selected action can not only affect the immediate reward received by the agent but also the reinforcement it could get in later actions.

So to overcome the above problem, value functions were introduced. A reward function indicates what is preferable in that moment, a value function specifies what is desirable in the long run. The value of a state is the estimated amount of reward an agent can get if it starts from that state. Without rewards, there could be no value to evaluate and the primary focus of value functions is to get more rewards. Action choices are based on value judgments. The rewards are given directly by the environment of the agent, but the values must be estimated and re-estimated from the observation sequences made by an agent throughout its existence. Hence, we can say that the most important factor of all RL algorithms is a method of efficiently estimating values.

The concept of a discount factor was introduced in response to problems faced from compensation operations. After the agent takes an action in each state, it gets compensation. As time passes, the value of reward decreases, introducing depreciation. Depreciation has a value between 0 and 1, and the amount of compensation the agent receives over time is reduced.

Before we delve into Reinforcement learning anymore, we first need to cover the systems that make up a Reinforcement Learning Algorithm starting with Markov Decision Processes (MDP).

3.6.1 Markov Decision Process

The goal of any MDP is to find a policy, denoted as π (pi), that yields the optimal reward in long-term. As discussed before, policies are simply a mapping of each state $\boldsymbol{s}$ to a distribution of actions $\boldsymbol{a}$. For each state $\boldsymbol{s}$, the agent should take action $\boldsymbol{a}$ with a certain probability. However, policies can also be deterministic.

The Markov Property states that the next state can be calculated solely by the current state, that is to day that no 'memory' is required. This applies to how the agent traverses the Markov Decision Process. Some optimization methods use past learning to fine-tune policies. This is not a violation of the Markov property.

The Markov Property applies not only to Markov Decision Processes but anything Markov-related like a Markov-Chain.

Finally, we can formally describe a Markov Decision Process as m = (S, A, P, R, gamma), where:

- S represents the set of all states.
- A represents the set of possible actions.
- P represents the transition probabilities.
- R represents the rewards.
- Gamma is known as the discount factor

The set of Bellman Equations are central to Markov Decision Processes. It outlines a framework for determining the optimal expected reward at a state s.

1. **Bellman Expectation Equation:** As discussed, the value function represents the expected value of a state. The value function is the sum of the reward to be received when the agent moves to the next state and is influenced by the agent's policy. This Bellman equation expresses the relationship between the value function of the present state and the value function of the next state.

$$\mathrm{v}_{\pi'}(\mathrm{s}) = \sum_{\mathrm{a}\in\mathrm{A}} \pi(\mathrm{a}|\mathrm{s})(\mathrm{R}_{\mathrm{t}+1} + \gamma \sum_{\mathrm{s}'\in\mathrm{S}} P^{\mathrm{a}}_{\mathrm{ss}'} \mathrm{v}_{\pi}(\mathrm{s}'))$$

Figure 5: *Bellman Expectation Equation*

$$\sum_{a\in A} \pi(\mathrm{a}|\mathrm{s})$$

Figure 6: *Probability policy to do the action*

$$\sum_{s'\in S} \mathrm{P}^{\mathrm{a}}_{\mathrm{ss}'}$$

Figure 7: *State transition probability matrix*

Rt+1 is the reward and Σ is the discount factor.

2. **BELLMAN OPTIMALITY EQUATION:** The aim of Reinforcement learning is to find the optimal policy for the problem defined by the MDP. The policy is determined using the value function, and the policy that gives the highest expectation for all other policies is the optimal policy. The Bellman optimality equation is the policy that receives the optimal value using the value function. The following is the Bellman optimal equation:

$$v_*(s) = \max_a E_\pi[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s]$$

Figure 8: *Bellman Optimality Equation*

Where max_a Eπ is the maximum expected value among the policies that agents can receive.

Reinforcement learning solves the problem defined by MDP using both, the Bellman expectation equation and the Bellman optimal equations.

3.6.2 Learning Algorithms

The scope of reinforcement learning largely depends on the environment and the type of data it is dealing with. In this report, we will be covering two significant types of learning algorithms that we have used in our model that is Q-learning and Policy Gradients.

1. **Q learning:**

Q-learning uses an off-policy method to separate the acting policy from the learning policy. As a result, even if the action selected in the next state was mediocre, the information was not included in the updating of the Q-function of the current state, and the dilemma is that it is a wrong choice. However, since Q-learning uses off-policy, it solves the dilemma. Equation for the Q-value is as follows:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[R + \gamma \max Q(s0\ ,\ a\ 0\ ) - Q(s, a)]$$

Figure 9: *Q-Value*

Where α is the learning rate and has a value between 0 and 1. R is a reward and γ is the reduction rate of the reward as time passes. The Q-value Q (S, A) of the action for the current state S is updated with the sum of the existing value Q (S, A) and the equation which determines the best action in the current state. Q-learning is continued by updating the Q-value for each state continuously using the above equation.

If an agent selects an action through a policy in the starting state, it moves to the next state. This process is repeated several times so that the overall Q-value converges to a specific value. Q-learning combines dynamic programming and Monte Carlo methods, which have been used to solve the Bellman equation. This approach has become the basis of many reinforcement learning algorithms because, unlike other methods, Q-learning is simple and exhibits an excellent learning ability in single-agent environments. However, in Q-learning, a value is updated only once per action. Therefore, it is difficult to effectively solve complicated problems in a large state-action environment because these many state actions might not have been experienced previously.

Moreover, in a multi-agent environment with two or more agents, a large state-action memory is required, which leads to problems. For this reason, basic Q-learning algorithms are disadvantageous because they cannot accomplish effective learning in a multi-agent environment.

2. **Policy Gradient**
   In this learning algorithm, agents directly learn the policy function π. Policy gradient (PG) algorithms, which can perform gradient ascent on π directly. The advantage of PG algorithms is that they are likely to converge with the optimal solution, so they are more widely used than value optimization algorithms.

   The trade-off is that PG has relatively low consistency. They have a more significant difference in their performance compared to value optimization techniques such as DQN, so PGs usually require a higher number of training samples.

   It's known as Policy-Based Reinforcement Learning because, as a result, we will parameterize the policy directly. Here is how our policy function will look:

$$\pi_Q(s, a) = P[a \mid s, Q]$$

Figure 10: *Policy Gradient - Policy Function*

Here:
s - state value; a - action value; Q - model parameters of the policy network.

Policy score function:
To measure how good our policy is, we use a function called the objective function that calculates the expected reward of the policy. In an episodic environment, we first take the starting value. Then, we calculate the mean of the return from the first time step ***G1***, which is called the cumulative discounted reward for the entire episode:

$$J_1(Q) = E_\pi[G_1 = R_1 + \gamma + \gamma^2 R_3 + ...] = E_\pi(V(s_1))$$

Figure 11: *Cumulative discounted reward*

Where $[G_1 = R_1 + \gamma + \gamma^2 R_3 + ...]$ is the cumulative discounted reward starting at start state and,
$E_\pi(V(s_1))$ is the value of state 1.

Policy gradient ascent:
The next step in PG is to find the parameter Q that may maximize this score function. To maximize the score function, we use gradient ascent on policy parameters.

Policy gradient ascent is the inverse of the gradient descent. Descending the gradient, we take the direction of the sharpest decrease in the function. As we ascend the slope, we take the fastest path of function increase. We choose the ascent algorithm as the function to optimize is a score function not an error function.

So, we want to find the best parameters Q* that maximize the score:

$$Q^* = \text{argmax} E_{\pi Q}\left[\sum_t R(s_t, a_t)\right]$$

Figure 12: *Score*

Here, arguments that are after argmax, starting from E, are equal to J(Q), so our score function can be defined as:

$$J(Q) = E_{\pi}[T(\tau)]$$

Here:
E - expected given policy; τ - Expected future reward.

Both of these methods have considerable drawbacks. The problem with Deep Q-learning is that their predictions assign a score for every possible action, within each time step, given the current state. This method becomes exponentially more resource-expensive as the number of actions increases. As for policy gradients, they suffer from noisy gradients and high variance.

That's why we move to a 'hybrid method': Actor-Critic.

3.6.3 Advantage Actor Critic

The Actor-Critic algorithm is a Reinforcement Learning agent that combines value optimization and policy optimization approaches. More specifically, the Actor-Critic combines the Q-learning and Policy Gradient algorithms. The resulting algorithm obtained at the high level involves a cycle that shares features between:

- Actor: a PG algorithm that decides on an action to take
- Critic: Q-learning algorithm that critiques the action that the Actor selected, providing feedback on how to adjust. It can take advantage of efficiency tricks in Q-learning, such as memory replay.

The advantage of the Actor-Critic algorithm is that it can solve a broader range of problems than DQN, while it has a lower variance in performance relative to REINFORCE. That said, because of the presence of the PG algorithm within it, the Actor-Critic is still somewhat sampling inefficient.

4 Implementation and experimentation

4.1 Data Description

4.1.1 Bitcoin Historical Data

We obtained the data on past prices from Cryptocompare as it regularly reviews exchanges, carefully monitoring the market to deliver the most accurate pricing indices. Bitcoin prices are known to be volatile, and the cryptocurrency market is still evolving. The main difference between BTC prices and usual stock prices is that it changes on a much greater scale than local currencies.

The overall data collection period is from January 2016 to April 2021 on a daily fidelity which gives us a total of 1918 values. This dataset consists of six different attributes, namely, Date, High, Low, Open, Close, and Volume. High and Low signifies the highest and lowest price at which Bitcoin was traded for that day. Close signifies the last recorded traded price of Bitcoin for that day. Volume here refers to the Bitcoin trading volume, which indicates how many Bitcoins are being bought and sold on specific exchanges on the day. Rows like High and Low are irrelevant for our analysis and can be hence dropped later.

4.1.2 Bitcoin related Tweets

Twitter is a minefield of information due to the volume and variety of users. This has resulted in Crypto Twitter being significantly influential on cryptocurrency trading decisions. Thus, we collected tweets to perform sentiment analysis on. We used the Twitter API to query tweets by applying to the developer role. However, the rate limit is limited to 10 calls per second, so fetching tweets for a period of 2016/1 to 2021/04 was time-consuming.

'bitcoin,' 'BTC,' '#Bitcoin,' and '#BTC' were used as the keyword filters for fetching tweets. Also, all tweets were fetched in the English language. This gave us a total of 4,265,266 tweets to work with. The data contains many columns, including the date of the tweet, username, tweet location, tweet ID, number of replies, number of retweets, number of favorites, the text of the tweet, list of mentions, hashtags, and permalinks. Some of the columns are irrelevant to our analysis and will be dropped later.

4.2 Implementation of Technical Analysis

4.2.1 Trend Indicators

Simple Moving Average:
SMA is simply the average price over the specified period. The average is called "moving" because it is plotted on the chart bar by bar, forming a line that moves along the chart as the average value changes. A simple moving average smooths out volatility and makes it easier to view the price trend of Bitcoin. If the simple moving average points up, this means that Bitcoin's price is increasing. If it is pointing down, it means that Bitcoin's price is decreasing. The longer the time frame for the moving average, the smoother the simple moving average. A shorter-term moving average is more volatile, but its reading is closer to the source data.

The shorter the time span(n) used to create the average, the more sensitive it will be to price changes. The longer the time span(n), the less sensitive the average will be to the price changes because more lag is introduced between the SMA and the Bitcoin prices. When it comes to our project, we aim at maximizing profits in the long run. So, we took a longer time span of 21 weeks to calculate our SMA to make sure that it is not sensitive to all the little changes in price but also does not overlook the major price changes. A time span of 21 weeks gives us a very balanced value of SMA, which is well suited for our project.

$$\textbf{SMA} = \frac{(A_1 + A_2 + \ldots\ldots + A_n)}{n}$$

where, A_n = the price of Bitcoin at period n
n = the number of total periods

Figure 13: *Formula for SMA*

Exponential Moving Average:
Exponential Moving Average (EMA) is similar to Simple Moving Average (SMA), measuring trend direction over a period of time. However, whereas the SMA simply calculates an average of price data, the EMA applies more weight to data that is more current. Because of its unique calculation, the EMA will follow prices more closely than a corresponding SMA.

EMA is used to determine trend direction and trade in that direction. When the EMA rises, you may want to consider buying when prices dip near or just below the EMA. When the EMA falls, you may consider selling when prices rally towards or just above the EMA.
The formula for EMA is:

$$EMA_{Today} = (Value_{Today} \times (\frac{Smoothing}{1 + Days})) + EMA_{Yesterday} \times (1 - (\frac{Smoothing}{1 + Days}))$$

Figure 14: *Formula for EMA*

where, EMA = Exponential Moving Average
Smoothing = smoothing factor

Although there are many options to choose from when considering the smoothing factor, we opt for a value of 2. This value gives more credibility to the most recent data points available. The more we increase the smoothing factor value, the more influence the most recent data will have on the moving average. So on testing for different values, we found the value of 2 to give just enough credibility to recent data. The EMA has a shorter delay than the SMA within the same period. Hence we decided to go for a period of 20 weeks for EMA calculations.

Relative Strength Index:
The Relative Strength Index (RSI) is a momentum oscillator that measures the speed and change of price movements. The RSI oscillates between zero and 100. Traditionally the RSI is considered overbought when above 70 and oversold when below 30. Signals can be generated by looking for divergences and failure swings. RSI can also be used to identify the general trend.

In an uptrend or bull market, the RSI remains in the 40 to 90 range, with the 40-50 zone acting as support. During a downtrend or bear market, the RSI tends to stay between the 10 to 60 range, with the 50-60 zone acting as resistance.

The RSI is computed with a two-part calculation that starts with the following formula:

$$RS = \frac{Avg.Gain}{Avg.Loss}$$

$$RSI = 100 - \frac{100}{1 + RS}$$

Figure 15: Formula for RSI

The average gain or loss used in the calculation is the average percentage gain or loss during a look-back period. The formula uses a positive value for the average loss. Periods with price losses are counted as 0 in the calculations of average gain, and periods when the price increases are counted as 0 for the calculation of average losses.

The standard is to use 14 periods to calculate the initial RSI value. Once there are 14 periods of data available, the second part of the RSI formula can be calculated. The second step of the calculation smooths the results.

4.2.2 Custom Indicator - BMSB

BMSB index is a Bollinger band that helps us understand the current valuation of bitcoin against USD, vis., undervalued, overvalued, or fair valued. It ranges from -100 to 100, with the negative side corresponding to extreme undervaluation and the positive side indicating extreme overvaluation with respect to recent price movements.

Bull Market Support band is a type of Bollinger band that consists of 20w SMA and 21w EMA. Its significance comes from the previous bull runs where the price was bouncing off or riding from a distance the support band until the end of the market cycle.

Usually, when the price of Bitcoin falls below BMSB, there is a bearish momentum in the market, and when the price of Bitcoin holds support or stays above the BMSB, it indicates the onset of a bull market. That being said, being over-extension (+70 to +100 range) does not necessarily imply a bull run, and under-valuation (-70 to -100 range) does not necessarily imply a bearish movement. In fact, it can indicate the end of a bull run and bear market, respectively. Therefore, in our project, when the BMSB index tends to -100, we show it as a buying opportunity, and when it tends to +100, we show it as a time to take profits.

Algorithm:
Let us introduce a few of the crucial variables that will enable us to formulate the algorithm for the BMSB Index: We define Combined Average (μ) as the average of the current 21 weeks EMA and 20 weeks SMA. We define K_p as the Price Coefficient. The price coefficient decides the extent of the price range from which the Bitcoin price is considered to be normal. The normal price range varies from $\mu(1 - K_p)$ to $\mu(1 + K_p)$. If the price of bitcoin is within the normal range, we calculate the BMSB Index as:

IF,

$$[\ p > \mu(1 - K_P) \quad \&\& \quad p < \mu(1 + K_P)\ ]$$

THEN,

$$x = \left(\frac{p - \mu}{\mu}\right)\ \frac{K_S}{K_P} \times 100$$

where Ks is defined as the scaling coefficient. The scaling coefficient decides the extent on the BMSB Index scale for the normal price range. It varies from 100(0-K_s) to 100(0+K_s) and corresponds to the $\mu(1 - K_p)$ to $\mu(1 + K_p)$ price range. If the price of bitcoin is outside the normal range, we calculate the BMSB Index as:

ELSE IF,

$$[\ p < \mu(1 - K_P)\ ]$$

THEN,

$$x = \left[\frac{p(1 - K_S)}{\mu(1 - K_P)} - 1\right] \times 100$$

ELSE,

$$[\ p > \mu(1 + K_P)\ ]$$

THEN,

$$x = \left[1 - \frac{\mu\ (1 + K_P)\ \ (1 - K_S)}{p}\right] \times 100$$

In this project, we have chosen K_p as 0.3 and K_s as 0.9 arbitrarily.

4.3 Implementation of Sentiment Analysis

4.3.1 Preprocessing

Cleaning:
After scraping the tweets, we had to drop some columns that we deemed irrelevant for the sentiment analysis. This included the rows containing the tweet locations, tweet id, the list of mentions, hashtags, and permalinks. We kept the columns that we deemed useful, which include the date, username, number of replies, retweets, favorites, and the text of the tweet. Before we are able to start doing any form of sentiment analysis, the tweets collected have to be cleaned.

Sample tweets before cleaning:

"Who is the most underappreciated industry mover and shaker of bitcoin?
pic.twitter.com/q3UdXfH0e2"
"Bitcoin price index https://www.worldcoinindex.com/coin/bitcoin #USD #EUR #CNY
#GBP #RUBpic.twitter.com/UDhg2Sbp8H"
"Bitcoin Climbs Above 11,316.7 Level, Up 6% https://yhoo.it/2YV6wKZ #bitcoin #crypto
#blockchain #btc #news #cryptocurrency pic.twitter.com/mPv3rd4kLn #eth #ltc #xrp"

These tweets contain a large amount of noise, such as hashtags, URLs, and pictures, and some may even contain non-English characters which our Sentiment Analyzer cannot process. Using regex expressions, all these noises were removed. Preprocessing is a very important aspect of sentiment analysis; if we were to pass these raw tweets to our analyzer, chances are it will perform very poorly and take up much more time

Sample Tweets after cleaning:

"Who is the most underappreciated industry mover and shaker of bitcoin "
"Bitcoin price index"
"Bitcoin Climbs Above Level Up"

After processing the tweets, we managed to cut down our CSV file size by 33.3%, which allows us to spend more time tweaking our model while preserving the important parts of the tweets.

Next, we set all the characters to lower cases and also removed stopwords in our tweets. Stop words are commonly used words such as "a," "the," and "as," which provide no valuable information in Sentiment analysis. As mentioned earlier, we do not want to waste time processing invaluable words, so we made use of the Natural Language Toolkit(NLTK) to get rid of them. This was enough for VADER as it was specially tuned for social media content.

Topic Modeling and LDA for Advertisement and Spam handling:
In our first attempt on sentiment analysis, our sentiment analysis did not yield good results as it gave advertisement tweets a high positive sentiment score

E.g. "Free bitcoin faucet with bonuses Get free bitcoins now" : +0.8807 (Using Vader)

This is an issue as our prediction model was unable to differentiate between a useful tweet and ads. Hence, we identified and tagged these ads by using Latent Dirichlet Allocation (LDA) to cluster and identify possible tweet topics. This is useful as it allows us to gain insight into the type of Bitcoin topics people discuss on Twitter and, within these topics, to identify an ad topic cluster. The reason for choosing LDA is that it is most effective in generalizing to new documents easily.

To train our LDA model, the following parameters were varied:

- Corpus: Bag of Words generated from the pre-processed tweets
- id2word: Dictionary mapping of word ID to words in tweets
- Number of topics: The number of topic clusters to be generated.

The model parameters were varied by using different numbers of tweets and the number of topic clusters. Each tweet is represented as a distribution over topics, and each topic is represented as a distribution over words. To evaluate our model parameters, we used the following:

1. **Visual analysis of clusters with pyLDAvis:**

Through the visual analysis of 10, 6,3,20 topic clusters, we found that 20 topic clusters gave the best result. In the case of 3 topic clusters where none of the clusters overlap in the Intertopic Distance Map in Fig 16, indicating that there are 3 clear, distinct topics, the topics were too broad and vague to use them to identify an ad cluster. The histogram on the right represents the most probable topic word for that cluster of tweets. Hence, we increased the cluster size until we found that our clusters gave us unique, most probable topic clusters.

As seen here in Fig 17, we can see that the various possible Bitcoin topics are: Bitcoin, market, price, BTC, mining, analyst, new, day, transaction, news, time, analysis, blockchain, dont, eth, payment, exchange, million, social media, resistance. Based on the cluster topics, we suspected that the unusual 'don't' cluster(Fig 18) (cluster 14 on pyLDAvis, topic 13) is likely to be the ad cluster.

2. Running new unseen documents through our model Advertisement Tweets Examples 1. 'is airdropping den to participants sign up here' 2. 'claim daily bitcoins at for free' 3. 'bitcoin black v bitcoin get free coins value ' 4. 'get free bitcoin' 5. 'claim free bitcoin now link moon bitcoin is a bitcoin faucet with a difference you decide how often to claim april at pm'

To confirm our hypothesis, we used tweets that we know are ads (above) and ran it through the LDA model. We wanted to find the model parameters that will result in all 5 tweets' LDA results returning topic 13 is the most probable topic. Hence, we are able to establish that topic 13 is our ad cluster and our ideal model parameters are to use all the tweets with 20 topic clusters. To tag the ad tweets, we used all tweets and ran it through our LDA model with 20 clusters, and filtered out all Topic 13 tokenized tweets and extracted the words to form an "ad word list" which is used in the preprocessing stage to tag the ad tweets.

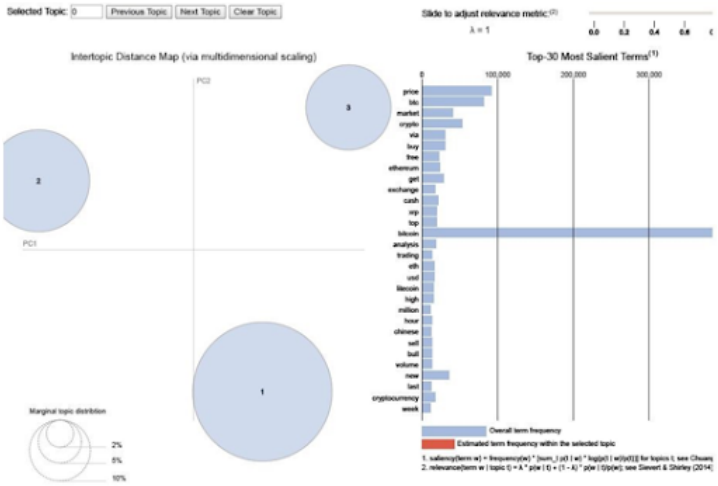


Figure 16: *pyLDAvis interactive visualization of Topics generated from LDA. Model parameters: # of tweets used =All, # of clusters = 3. On the left, is a PCA Plot, where the size of area of the circle refers to the importance of each topic over the entire corpus, distance refers to the similarity. On the right, the histogram shows the top 30 most probable topic terms.*

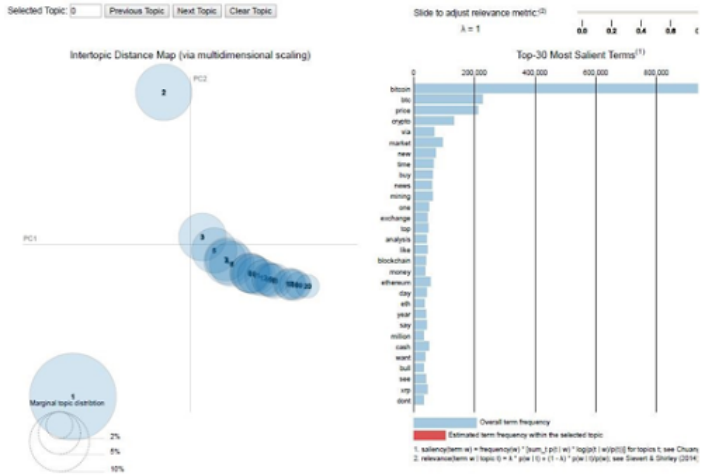


Figure 17: *pyLDAvis interactive visualization of Topics generated from LDA. Model parameters: # of tweets used =All, # of clusters = 20.*

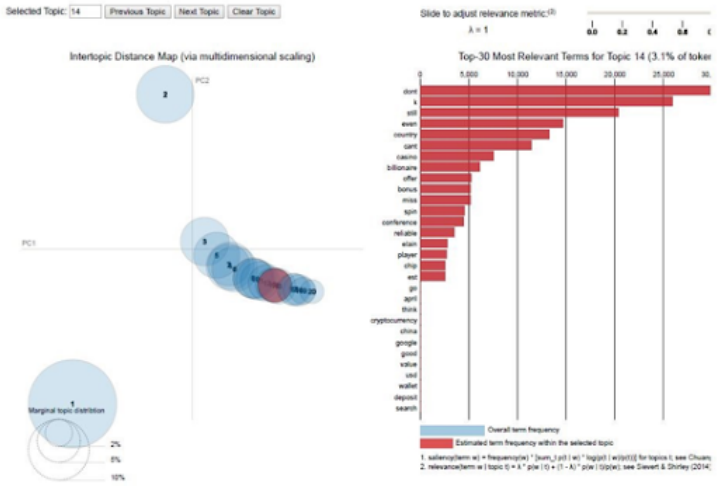


Figure 18: *pyLDAvis for suspected ad topic cluster*

4.3.2 Sentiment Analysis

Our goal is to apply sentiment analysis to our collected tweets to measure the subjective emotion or opinion of these tweets regarding Bitcoin. Due to our Tweet data set being unlabelled, we attempted various ways of labeling the data, such that we could use document vectors or TF-IDF models for sentiment analysis. As our dataset required for training the sentiment analysis model is very large, it is beyond our project time and manpower to manually label the sentiment of each tweet. We explored using Topic Modeling, as mentioned before, to help automate data labeling.

We then compare between VADER and TextBlob, which are lexicon-based approaches for sentiment analysis. VADER's sentiment function returns a compound polarity score. The score lies in the range of [0, 1], where 1 refers to positive sentiment. TexBlob's sentiment function returns a polarity score that lies in the range of [-1, 1], which we have scaled to [0, 1] to match VADER's range.

Based on observation, we selected a few tweets to compare if VADER or TextBlob performs better. We noticed that the sentiment score for advertisements and spam tweets was mostly quite positive. Hence, we filtered out such tweets, which has been described in detail in the Topic Handling section. From the tests we conducted on the selected few tweets, VADER works better with things like slang, emojis, etc., which is quite prevalent on Twitter, and it is also more sensitive than TextBlob. So, we choose to go with VADER.

After the sentiment of each Tweet is computed, we take the average sentiment for daily fidelity. However, simply taking the average sentiment results in the loss of information. For example, a prominent user's positive tweet (score of +1.0) may have greater influence over public opinion as compared to a common users' negative tweet (score of 0). In this case, the simple average sentiment is neutral (0.0), yet the effect on public opinion should be positive. To address this issue, we decided to use weighted Sentiment analysis scores.

4.3.3 Feature Selection and Weighted Sentiment Score

Our collection of features consists of Tweets text, Time of Tweet, Number of Retweets, Number of Replies, Number of Favorites, Advertisement score, Tweet Volume, and Sentiment Analysis Score (VADER). To determine which features we want to use for our models, we did feature engineering and selection to use to create the Weighted Sentiment Analysis score.

To prevent the loss of information by taking the average sentiment, we have generated weights

to create weighted sentiment scores. Each tweet will have a weighted score. Figure 19 shows the Weight Rules we have established

	Rule	Weight Score
1.	**Prominent Users:** We generated a list of prominent Cryptocurrency Twitter users. Just to name a few accounts, @APompliano, @BitcoinMagazine, @BTCNewsletter, @coindesk, @scottmelker	If the Tweet is made by users on the list, +1 to the weight score
2.	**Bitcoin Keyword List:** We generated a list of keywords that are used in Bitcoin. If any of the words are used, it is likely that the tweet contains useful information.	Add weight +1 if at least one or more keywords are used.
3.	**Number of Retweets, Replies and Favorites:** The number of retweets, replies and favorites are indicators of impact and reach. The larger the number, the greater the weight.	Added the round(log(no of retweets + replies + favorites)) to the weight score. As some tweets may have thousands of retweets while some have none, we took log to normalize the data.
4.	**Time:** As our price data is based on the closing price, the tweets made nearer to closing price are more reflective of the current public opinion during the closing price. Hence, we used a naive method of splitting the time period into half. Tweets made in the later half carry more weight.	For daily, if the tweet is made from 13:00 to 23:59 and for hourly, if the tweet is made in the last 30 mins, +1 is added to the weights.
5.	**Advertisement Score:** As mentioned previously, we needed a way to pre-processing. filter out the ad tweets. In the data pre-processing stage, based on the LDA ad cluster's results, we are able to tag each tweet with an ad_score.	The Advertisement Score weight is the ad_score obtained during data pre-processing.

Figure 19: *Weight Rules we have established*

The final weights equation is as follows:

Weight = [weight(prominent user) + weight(keyword) + weight(# of retweets) + weight(# of replies) + weight(# of favorites) + weight(time)] * ad_score

Weighted Sentiment per tweet = sentiment score of tweet * Weight

Weighted Sentiment per day = sum(Weighted Sentiment per tweet)/ # of tweets in one day

To address the issue with VADER assigning ads high polarity scores, we multiply the ad_score to the other weights. As ad_score for ads are assigned as 0, resultant weight is 0, which in turn, Weighted Sentiment will be 0. As the weighted sentiment of ads are now neutral, we have effectively filtered out the ads as it will not affect our model prediction.

Hence this newly generated Weighted Sentiment Score is then passed onto the RL model as a feature.

4.4 Implementation of Reinforcement Learning

4.4.1 Environment

Positions:
Positions of the market describe the amount of Bitcoin a particular user holds at that moment. In our environment, we have considered long and short positions. These suggest the two potential directions of the price required to maximize profit. Going Short is a trading technique in which

a trader borrows an asset in order to sell it, with the expectation that the price will continue to decline. In the event that the price does decline, the short seller will then buy the asset at this lower price in order to return it to the lender of the asset, making the difference in profit. Going Long is when an investor gains exposure to cryptocurrency with the expectation that prices will rise at a later date, meaning that the asset can be sold for a profit.

With regards to our project, the Long position wants to buy shares when prices are low and profit by sticking with them while their value is going up, and the Short position wants to sell shares with high value and use this value to buy shares at a lower value, keeping the difference as profit.

The environment assigns a value of 0 to the agent if the position is discovered to be Short and 1 if the position is discovered to be Long. When starting a new environment, the position of the user is considered short as default.

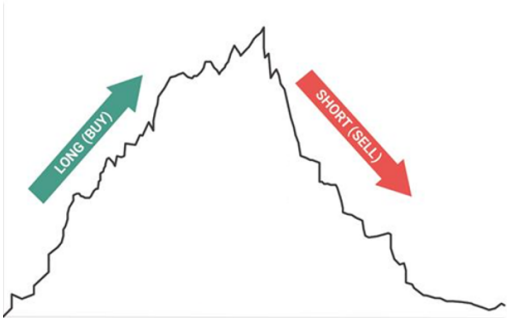


Figure 20: *Possible Long/Buy or Short/Sell situations*

Actions:
The reinforcement agent aims at maximizing profit by recommending the best course of action to the user. There are numerous actions that a user can perform in the trading market like Buy, Sell, Hold, Enter, Exit, etc. However, these actions just make things complicated with no real positive impact. In fact, they increase the learning time. Therefore there is no need to have numerous such actions, and only Sell and Buy actions are adequate to train an agent just as well. That is to say that our agent will always be recommending the user to either Buy or Sell to make sure it does not miss a single penny. The environment assigns a value of 0 to the agent if the recommended action is Sell and 1 if recommended action is Buy.

However, performing a trade on every interval does not produce reliable results for a real-life situation. To bridge the gap between simulation and reality and maximize long-term profits, the algorithm accepts the recommendation from the model and the user's current position in the market to provide the final verdict. The algorithm first decides between trading or holding. To do that, it refers to the position of the environment.

```
if ((action == 1 and env._position.value == 0) or (action == 0 and env._position.value == 1)):
            trade = True
print(trade)
```

Figure 21: *Code snippet showcasing trade condition*

When you make a short trade, you are selling a borrowed asset hoping that its price will go down and you can buy it back later for a profit. So our algorithm will recommend the user to Buy after the short trade is satisfied and Sell after the long trade is satisfied. If the above conditions are not satisfied, then no trade occurs (Holding) because our conditional logic suggests that profit can be maximized just by holding in such a case.

In the end, our algorithm can now recommend three independent actions depending on the situation of the market, namely, Buy, Hold or Sell.

4.4.2 Learning Algorithm

Model Inputs:
The reinforcement learning model predicts an action by taking the current state of the environment as input. We provide all the hand crafted features developed till now for the model to optimize its learning. Refer to Fig 22

Feature	Definition
Closing Prices	The closing price is the value of the last transacted price of Bitcoin for the day.
Volume	The Bitcoin trading volume indicates how many Bitcoins are being bought and sold on specific exchanges.
Simple Moving Average	The simple moving average (SMA) is a way to calculate the average price of a cryptocurrency for a predetermined set of data and help to track the trend direction
Exponential Moving Average	EMAs are similar to SMAs. EMA involves a more complex calculation to place more weight on the most recent price changes. EMA are more sensitive than SMA
RSI	The RSI index measures momentum and oscillates on a scale between 0 and 100 and gives a gist of the market value of Bitcoin.
Bull Market Support Band	Our custom index which indicates whether Bitcoin is currently undervalued or overvalued in the current market. It ranges between -100 to 100.
Weighted Sentiment Scores	The weighted sentiment score indicates how negative or positive the overall sentiment for trading of Bitcoin has been for that day

Figure 22: *Features used, along with definitions*

Model Specifications:
The Asynchronous Advantage Actor-Critic method (A3C) has been very influential since the paper [9] was published. The algorithm combines a few key ideas:

- An updating scheme that operates on fixed-length segments of experience (say, 20 timesteps) and uses these segments to compute estimators of the returns and advantage function.
- Architectures that share layers between the policy and value function.
- Asynchronous updates.

AI researchers questioned whether the asynchrony of A3C led to improved performance or if it was just an implementation detail that allowed for faster training with a CPU-based implementation. Synchronous A2C performs better than asynchronous implementations, and we have not seen any evidence that the noise introduced by asynchrony provides any performance benefit. This A2C implementation is more cost-effective than A3C when using single-GPU machines and is faster than a CPU-only A3C implementation when using larger policies. The host of this algorithm was the 'stable baselines 3' library on Python. The library offers multiple learning policies depending on the type of input data.

We have used the MLP (Multi-Layer Perceptron) policy which acts as a base policy class for actor-critic networks allowing both policy and value prediction. It provides the learning algorithm

with all the base parameters like the learning rate scheduler, activation functions, normalization, feature extraction class, and much more. This model is learned with a sliding window approach.

4.5 Evaluation Criteria

Entropy Loss:
Reinforcement learning agents are notoriously unstable to train compared to other types of machine learning algorithms. One of the ways that a reinforcement learning algorithm can under-perform is by becoming stuck during training on a strategy that is neither a good solution nor the absolute worst solution. We generally refer to this phenomenon as reaching a "local minimum" in the space of game strategies.

In reinforcement learning, a similar situation can occur if the agent discovers a strategy that results in a reward that is better than it was receiving when it first started but very far from a strategy that would result in an optimal reward. This might often manifest as the agent deciding to take a single move, over and over. Entropy loss, we believe first discussed in a 1991 paper, is an additional loss parameter that can be added to a model to help with local minima in the same way that momentum might and to provide encouragement for the agent to take a variety of moves and explore the set of strategies more.

To solve this, we will encourage the agent to vary its moves by adding a new loss parameter based on the entropy of its predicted moves. The equation for entropy here is:

$$H(x) = -\sum_{i=1}^{n} \mathrm{P}(x_i) \log_e \mathrm{P}(x_i)$$

Figure 23

This encourages the network to only make strong predictions if it is highly confident in them, which means that the actor-critic network will have to have a low enough loss from the policy and value loss parameters to offset the more positive entropy loss sufficiently so that the global loss remains low. Entropy loss is a clever and simple mechanism to encourage the agent to explore by providing a loss parameter that teaches the network to avoid very confident predictions. As the distribution of the predictions becomes more spread out, the network will sample those moves more often and learn that they can lead to greater rewards.

Policy Loss:
The letter π will symbolize a policy. Let's call $\pi\theta(a \mid s)$ the probability of taking action a in state s. θ represents the parameters of our policy (the weights of our neural network). Our goal is to update θ to values that make $\pi\theta$ the optimal policy. Because θ will change, we will use the notation θ_t to denote θ at iteration t. We want to find out the update rule that takes us from θ_t to θ_{t+1} in a way that we eventually reach the optimal policy.

Typically, for a discrete action space, $\pi\theta$ would be a neural network with a softmax output unit, so that the output can be thought of as the probability of taking each action.

Clearly, if action a* is the optimal action, we want $\pi\theta(a* \mid s)$ to be as close to 1 as possible.

Value Loss:
Value loss is the difference (or an average of many such differences) between the learning algorithm's expectation of a state's value and the empirically observed value of that state.

A state's value is how much reward you can expect, given that you start in that state. Immediate reward contributes completely to this amount. Reward that can possibly occur but not

immediately contribute less, with more distant occurrences contributing less and less. We call this reduction in the contribution to value a "discount", or we say that these rewards are "discounted". Expected value is how much the critic part of the algorithm predicts the value to be. In the case of a critic implemented as a neural network, it is the output of the neural network with the state as its input.

Empirically observed value is the amount you get when you add up the rewards that you actually got when you left that state, plus any rewards (discounted by some amount) you got immediately after that for some number of steps (we will say after these steps you ended up on state X), and (perhaps, depending on implementation) plus some discounted amount based on the value of state X.

4.6 Training of proposed model

The training of the RL agent was done on the Google Colab GPU, with *NVIDIA-SMI 460.67 Driver Version: 460.32.03 CUDA Version: 11.2.* The model was tested for multiple scenarios, and the below gave the most consistent results. The time steps were set to be 50000, which is equivalent to the number of iterations of training. The 1918 data instances were present, and 1233 were used for training the model with a window size of 30. The remaining 685 instances were used for testing.

4.7 User Interface

A web app is deployed to allow the RL agent to recommend the optimal action to the user. The agent can recommend one of the following actions depending on the market situation: Buy, Hold, Sell.

Figure 24

The website also provides visualizations of various important indicators for the user. This would further assist the user to reinforce the agent's recommendation.

Figure 25

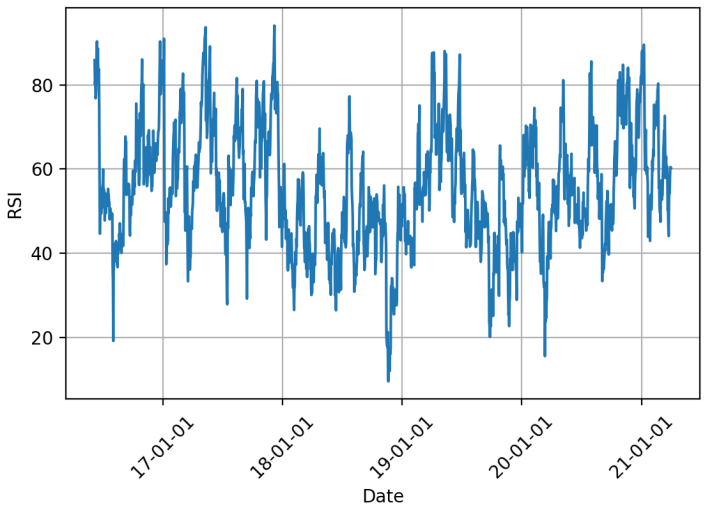


Figure 26

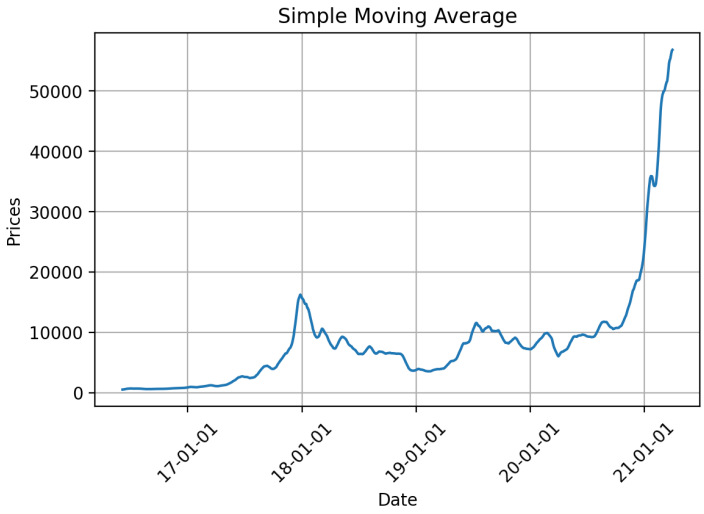


Figure 27

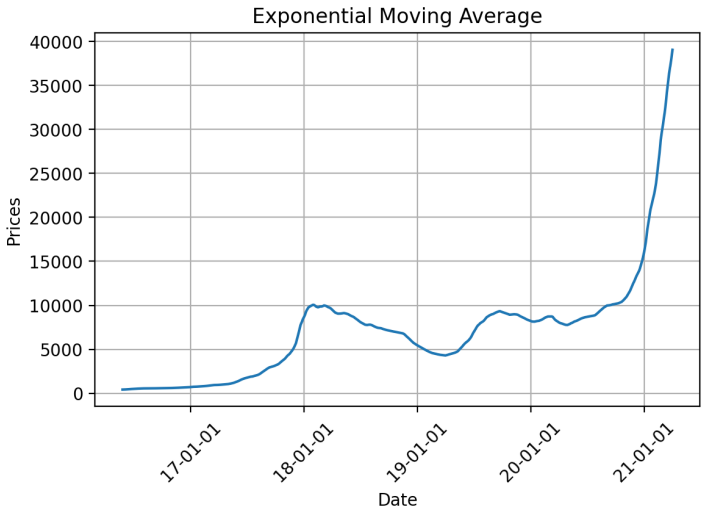


Figure 28

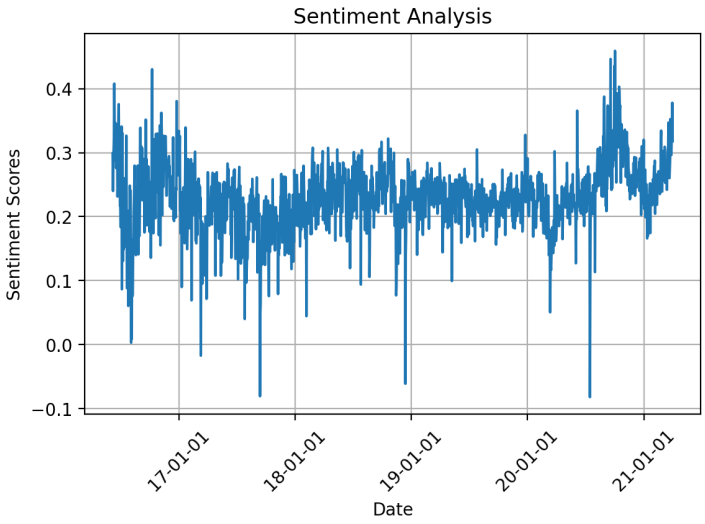


Figure 29

Figure 30

4.8 Results and discussion

After training for 50000 time steps, the following loss results were obtained.

time/	
fps	645
iterations	10000
time_elapsed	77
total_timesteps	50000
train/	
entropy_loss	-0.0289
explained_variance	0.1
learning_rate	0.0007
n_updates	9999
policy_loss	0.0172
value_loss	29.3

Figure 31

Finally, the testing was performed on the latest 685 days in the data. Specifically, the testing period started from 17th May 2019 and ended on 1st April 2021. This period is highly significant as it captures the peak points of Covid-19 and its effects on the market. Given these extreme scenarios, the model was able to generate a total reward of 39,412 and a profit value of 1.69486, which is equivalent to a 69.486% increase over 685 days.

Figure 32

The green dots in the graph indicate a long position, whereas the red dots indicate a short position. As discussed before, the agent itself recommends only Buy or Sell with the aim to capture every penny. Our algorithm then reviews the current situation/position of the user in the market and either recommend the agent action or asks them to hold their assets.

5 Conclusions and scope for further work

5.1 Conclusion

Investing in the crypto market is a tedious task and requires a lot of effort to make good trading strategies. Moreover, analyzing real-time data is a more difficult task. Traders make several trading decisions and keep on learning from them. During this process, they improve their decision-making skills. This is the major idea behind implementing a Reinforcement Learning agent, which trains against the environment to adapt and make better decisions. Thus, we can say that trading can be approached as a Reinforcement Learning Problem.

In the trading scene, experts use sophisticated tools to analyze the trend of prices better. Our plan was to allow the learning agent to learn in such a feature-rich learning environment. Specifically, we have used some of the most popular and tested long-term trend indicators like SMA, RSI, and EMA. We have also crafted our technical indicator, which is used by the agent to find opportunities for trade action. Another huge factor that impacts the flow of prices is the overall sentiment of the market. By including this factor in the environment, the agent was better able to understand the market situation.

The value of the added features was demonstrated as without these, the average profit of the model was not always positive. In addition to the features, we were able to get consistently positive profits over a long period of testing. With these handcrafted features, our model was able to analyze the situation of the user and market and recommend smart decisions to the user.

We were able to provide an interface to the user for their better understanding of the current market situation through visualizations of the important indicators and sentiment scores. The interface was able to host the agent to provide its final recommendation to the user. The results show great potential for the approach, but the bitcoin markets are quite large, complex and volatile, so the modelling of this environment still presents a lot of challenges.

5.2 Scope for further work

Many experts in the field of cryptocurrency and stock trading utilize trend analysis by identifying the popular patterns in the price action. Each of these patterns helps in the analysis of the price changes to occur in the future. The ascending and descending triangle pattern, as shown in the figure, leads to a continuity in the trend of prices. Another popular pattern experts use is the head and shoulders pattern shown in figure 2. This pattern is a strong sign of a reversal of the trend of the prices. Because of their impact, the recognition of these patterns becomes of most significance. A reliable recognition system will be sure to aid the reinforcement agent in better understanding the trading space and making smarter decisions. However, current pattern matching algorithms fail to work for different pattern spans. This problem is highly significant as even though we have an idea of what pattern we are looking for, most patterns occur at significantly different intervals.

Nevertheless, there is still a promise of research in this department to take the project to the next level. In recent times, communities on Reddit have had a significant impact on the prices of cryptocurrencies. So another step forward for the project would be to include sentiments from specific subReddits to our sentiment scores. This will involve assigning weights to communities and comparing them with the scores from Twitter.

Bibliography

References

[1] Gabriel Borrageiro, Nick Firoozye, and Paolo Barucca. "The Recurrent Reinforcement Learning Crypto Agent". In: *arXiv preprint arXiv:2201.04699* (2022).

[2] Raymond J Dolan. "Emotion, cognition, and behavior". In: *science* 298.5596 (2002), pp. 1191–1194.

[3] Zhengyao Jiang, Dixing Xu, and Jinjun Liang. "A deep reinforcement learning framework for the financial portfolio management problem". In: *arXiv preprint arXiv:1706.10059* (2017).

[4] Pieter de Jong, Sherif Elfayoumy, and Oliver Schnusenberg. "From returns to tweets and back: an investigation of the stocks in the Dow Jones Industrial Average". In: *Journal of Behavioral Finance* 18.1 (2017), pp. 54–64.

[5] Daniel Kahneman and Amos Tversky. "Prospect theory: An analysis of decision under risk". In: *Handbook of the fundamentals of financial decision making: Part I*. World Scientific, 2013, pp. 99–127.

[6] JoonBum Leem and Ha Young Kim. "Action-specialized expert ensemble trading system with extended discrete action space using deep reinforcement learning". In: *PloS one* 15.7 (2020), e0236178.

[7] Fengrui Liu et al. "Bitcoin transaction strategy construction based on deep reinforcement learning". In: *Applied Soft Computing* 113 (2021), p. 107952.

[8] Isaac Madan, Shaurya Saluja, and Aojia Zhao. "Automated bitcoin trading via machine learning algorithms". In: *URL: http://cs229. stanford. edu/proj2014/Isaac% 20Madan* 20 (2015).

[9] Volodymyr Mnih et al. "Asynchronous methods for deep reinforcement learning". In: *International conference on machine learning*. PMLR. 2016, pp. 1928–1937.

[10] Alexander Pak and Patrick Paroubek. "Twitter as a corpus for sentiment analysis and opinion mining". In: *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*. 2010.

[11] Galen Thomas Panger. *Emotion in social media*. University of California, Berkeley, 2017.

[12] Otabek Sattarov et al. "Recommending cryptocurrency trading points with deep reinforcement learning approach". In: *Applied Sciences* 10.4 (2020), p. 1506.

[13] Evita Stenqvist and Jacob Lönnö. *Predicting Bitcoin price fluctuation with Twitter sentiment analysis*. 2017.

[14] Hongkee Sul, Alan R Dennis, and Lingyao Ivy Yuan. "Trading on twitter: The financial information content of emotion in social media". In: *2014 47th Hawaii International Conference on System Sciences*. IEEE. 2014, pp. 806–815.

Nomenclature

D_n: Document Number n

W_n: Word Number n

a: Action

s: State

π: Policy Function

R_{t+1}: Reward

Σ: Discount Factor

Q: Q-value of action

J: Score Function

G_n: Time step n

V_n: Value at state n

E: Expected Value

τ: Expected future reward

A_n: Price of bitcoin at period n

RS: Relative Strength

p: Price of Bitcoin

μ: Combined Average of EMA and SMA

K_p: Price Coefficient

K_s: Scaling Coefficient

x: BMSB Value

H: Entropy Loss

$P(x_i)$: Probability Function for x_i

Appendix

Coding

The implementation of the elements presented respects the conventions for OpenAI Gym's Env class API. All sections are coded in Python.

The environment

Actions for agent are encoded in the following manner: (the final agent also adds a hold action)

```
class Actions(Enum):
    Sell = 0
    Buy = 1
```

Positions of the environment can be changed through the opposite function:

```
class Positions(Enum):
    Short = 0
    Long = 1

    def opposite(self):
        return Positions.Short if self == Positions.Long else Positions.
```

To create the instance of our environment, we need the following values:

```
def __init__(self, df, window_size):
    assert df.ndim == 2

    self.seed()
    self.df = df
    self.window_size = window_size
    self.prices, self.signal_features = self._process_data()
    self.shape = (window_size, self.signal_features.shape[1])

    # spaces
    self.action_space = spaces.Discrete(len(Actions))
    self.observation_space = spaces.Box(low=-np.inf, high=np.inf, shape=self.shape, dtype=

    # episode
    self._start_tick = self.window_size
    self._end_tick = len(self.prices) - 1
    self._done = None
    self._current_tick = None
    self._last_trade_tick = None
    self._position = None
    self._position_history = None
    self._total_reward = None
    self._total_profit = None
    self._first_rendering = None
    self.history = None
```

- *df* - Dataframe containing all the features
- *window_size* - Span of sliding window

The environment can take a step, that is to perform the recommended action with this function. It outputs the reward and information back.

```
def step(self, action):
    self._done = False
    self._current_tick += 1

    if self._current_tick == self._end_tick:
        self._done = True

    step_reward = self._calculate_reward(action)
    self._total_reward += step_reward

    self._update_profit(action)

    trade = False
    if ((action == Actions.Buy.value and self._position == Positions.Short) or
        (action == Actions.Sell.value and self._position == Positions.Long)):
        trade = True

    if trade:
        self._position = self._position.opposite()
        self._last_trade_tick = self._current_tick

    self._position_history.append(self._position)
    observation = self._get_observation()
    info = dict(
        total_reward = self._total_reward,
        total_profit = self._total_profit,
        position = self._position.value
    )
    self._update_history(info)

    return observation, step_reward, self._done, info
```

The model

The model uses a MLP policy as the base, which is defined below. Both the actor and critic models are Sequential neural networks.

```
class MLPBase(NNBase):
    def __init__(self, num_inputs, recurrent=False, hidden_size=64):
        super(MLPBase, self).__init__(recurrent, num_inputs, hidden_size)

        if recurrent:
            num_inputs = hidden_size

        init_ = lambda m: init(m, nn.init.orthogonal_, lambda x: nn.init.
                               constant_(x, 0), np.sqrt(2))

        self.actor = nn.Sequential(
            init_(nn.Linear(num_inputs, hidden_size)), nn.Tanh(),
            init_(nn.Linear(hidden_size, hidden_size)), nn.Tanh())

        self.critic = nn.Sequential(
```

```
            init_(nn.Linear(num_inputs, hidden_size)), nn.Tanh(),
            init_(nn.Linear(hidden_size, hidden_size)), nn.Tanh())

        self.critic_linear = init_(nn.Linear(hidden_size, 1))

        self.train()

    def forward(self, inputs, rnn_hxs, masks):
        x = inputs

        if self.is_recurrent:
            x, rnn_hxs = self._forward_gru(x, rnn_hxs, masks)

        hidden_critic = self.critic(x)
        hidden_actor = self.actor(x)

        return self.critic_linear(hidden_critic), hidden_actor, rnn_hxs


class A2C():
    def __init__(self,
                 actor_critic,
                 value_loss_coef,
                 entropy_coef,
                 lr=None,
                 eps=None,
                 alpha=None,
                 max_grad_norm=None,
                 acktr=False)
```

The final code demonstrates the initialization of the A2C model.

Acknowledgements

We would like to express my very great appreciation to Ankit Khivasara Sir and Chaitali Kulkarni Ma'am for their valuable and constructive suggestions during the planning and development of this project. Their willingness to give their time so generously has been very much appreciated.

We would also like to extend our gratitude to the entire team of OpenAI as well as that of Stable Baselines3 for implementing and maintaining their framework, without which this project would not be possible.