

COMPARISON OF LSTM AND XGBOOST FOR ETHER PRICE PREDICTION FROM SPAM FILTERED TWEETS

KUMARESH S	(19Z327)
AJITH NARAYANA S	(19Z327)
MRIDULA M	(19Z332)
SRAVYA VANKADARA	(19Z348)
CHANDRA PRAKASH J	(20Z461)

19Z720 PROJECT WORK-1

Dissertation submitted in partial fulfilment of the requirements for the degree of

BACHELOR OF ENGINEERING

Branch: COMPUTER SCIENCE AND ENGINEERING

of Anna University



SEPTEMBER 2022

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PSG COLLEGE OF TECHNOLOGY

(Autonomous Institution)

COIMBATORE – 641 004

PSG COLLEGE OF TECHNOLOGY

(Autonomous Institution)

COIMBATORE - 641 004

COMPARISON OF LSTM AND XGBOOST FOR ETHER PRICE PREDICTION FROM SPAM FILTERED TWEETS

Bona fide record of work done by

KUMARESH S	(19Z327)
AJITH NARAYANA S	(19Z327)
MRIDULA M	(19Z332)
SRAVYA VANKADARA	(19Z348)
CHANDRA PRAKASH J	(20Z461)

Dissertation submitted in partial fulfilment of the requirements for the degree of

BACHELOR OF ENGINEERING

Branch: COMPUTER SCIENCE AND ENGINEERING

of Anna University

SEPTEMBER 2022

.....

Dr. K. Sathiyapriya

Faculty guide

.....

Dr. G. Sudha Sadasivam

Head of the Department

Certified that the candidate was examined in the viva-voce examination held on

.....

(Internal Examiner)

.....

(External Examiner)

CERTIFICATE

Certified that this report titled '**Comparison of LSTM and XGBoost for Ether price prediction from spam filtered tweets**' for the Project Work 1(19Z720) is a bonafide work of **Kumaresh S (19Z327), Ajith Narayana M S (19Z331), Mridula M (19Z332), Sravya Vankadara (19Z348), and Chandraprakash J (20Z461)** who have carried out the work under my supervision for the partial fulfilment of the requirements for the award of the degree of Bachelor of Engineering in Computer Science and Engineering. Certified further that to the best of my knowledge and belief, the work reported herein does not form a part of any other thesis or dissertation based on which a degree or an award was conferred on an earlier occasion.

Place : Coimbatore

Date :

Dr. K. Sathiyapriya
Assistant Professor
(Selection Grade)
Department of Computer Science
PSG College of Technology

COUNTERSIGNED

HEAD

Department of Computer Science and Engineering

PSG College of Technology

COIMBATORE – 641 004

SYNOPSIS

Ether is the native cryptocurrency of the Ethereum platform. In terms of market capitalization, it ranks right below Bitcoin. Information that impacts the prices of Ether, and other cryptocurrencies, can be from a multitude of sources. One such significant information source is Twitter, a popular microblogging platform that is used robustly by active Ethereum users. Its high volume of influential users has made Twitter an important platform that impacts the purchasing decisions of cryptocurrency buyers and sellers. Analysing all the Ether information from tweets individually or manually is tremendously difficult. Understanding the relationship between tweets and the price of Ether, while taking into consideration the tweet's influence, can help predict the volatile price trends of Ether. Moreover, Twitter has a lot of spam tweets that do not add value to the topic but simply attempt to take advantage of the buzzword. Hence these tweets are filtered to exclude spam tweets, like advertising bots, impersonating bots, etc. to give more significant tweets.

The principle behind predicting price change is determining the importance of the author as well as the tweet itself that significantly impacts price: LSTM and XGBoost models are used to analyse the filtered Twitter information and predict Ether price changes. The performances of these models are then compared using metrics such as accuracy, confusion matrix, area under the receiver operating characteristics (AUROC), precision, recall, F1 score, and Matthew Correlation Coefficient (MCC) to determine which model might be more effective overall. This could help a potential buyer or seller make informed decisions while interacting with cryptocurrency, specifically Ether.

Table of Contents

Chapter	Page No.
SYNOPSIS.....	i
1. INTRODUCTION	1
1.1 Predicting price of Ether.....	1
1.2 Machine Learning Models.....	2
1.2.1 C5.0 Classification Algorithm	2
1.2.2 Valence Aware Dictionary Sentiment Reasoner (VADER).....	2
1.2.3 Extreme Gradient Boosting (XGBoost)	2
1.3 Deep Learning Models.....	3
1.3.1 Long Short-Term Memory (LSTM)	3
1.4 Motivation	4
1.5 Problem Statement	5
1.6 Objective	5
1.7 Scope	5
2. LITERATURE STUDY.....	7
3. SYSTEM ANALYSIS.....	10
3.1 Hardware Requirements	10
3.2 Software Requirements.....	10
3.3 Datasets	11
3.4 Functional Requirements	11
3.5 Non-Functional Requirements	12
3.6 Feasibility.....	12
4. SYSTEM DESIGN AND IMPLEMENTATION	13
4.1 High Level Architecture	13
4.2 System Implementation	14
4.2.1 Tweet Scraping with snsrape	14
4.2.2 Tweet Pre-processing.....	14
4.2.3 C5.0 Classification Algorithm	16
4.2.4 Sentiment Analysis with VADER	20
4.2.5 Data Condensation.....	21
4.2.6 Prediction model using LSTM	22

4.2.7 Prediction model using XGBoost	25
5. TESTING.....	28
5.1 Hyperparameter and Feature Optimization	28
5.1.1 C5.0.....	28
5.1.2 LSTM.....	29
5.1.3 XGBoost.....	30
6. EXPERIMENTAL RESULTS.....	31
6.1. Metric Definitions.....	31
6.1.1 Root Mean Squared Error	31
6.1.2 Mean Squared Error	31
6.1.3 R Squared Error	31
6.1.4 Mean Absolute Percentage Error	32
6.2. Results.....	32
6.2.1 Root Mean Squared Error	32
6.2.2 Mean Squared Error	33
6.2.3 R Squared Error	34
6.2.4 Mean Absolute Percentage Error	35
6.2.5 Comparison of LSTM and XGBoost	36
7. CONCLUSION AND FUTURE ENHANCEMENTS.....	39
BIBLIOGRAPHY	40
APPENDICES.....	44
PLAGIARISM REPORT	59

LIST OF FIGURES

Figure No.	Figure Name	Page No.
1.1	XGBoost Flowchart	3
1.2	Basic LSTM cell	4
4.1	High level view of proposed solution	13
4.2	Using TwitterSearchScraper module of snsrape tweet and user data	14
4.3	Tweet Pre-processing	16
4.4	Using IsItPhish API to check links present in tweets	17
4.5	Using Google SafeBrowsing API to check links present in tweets	17
4.6	Using predict_botometer() to access Botometer API	18
4.7	Using predict_bot() to access TweetBotornot2 predictions	18
4.8	Overview of C5.0 implementation	19
4.9	C5.0 model definition and model fitting	20
4.10	Overview of VADER implementation	20
4.11	Using VADER to assign polarity scores to tweets	21
4.12	Tweet Condensation	22
4.13	Working of LSTM Model	23
4.14	LSTM Sequence Building	24
4.15	LSTM Model Definition	24
4.16	XGBoost Model Working	25
4.17	XGBoost Model Definition	26
4.18	XGBoost Sequence Building	27
6.1	LSTM Model RMSE	33

6.2	XGBoost Model RMSE	33
6.3	LSTM Model MSE	34
6.4	XGBoost Model MSE	34
6.5	LSTM Model Training R Squared Error	35
6.6	LSTM Model Testing R Squared Error	35
6.7	XGBoost Model R Squared Error	35
6.8	LSTM Model MAPE	36
6.9	XGBoost Model MAPE	36
6.10	LSTM Model Performance Metrics	36
6.11	XGBoost Model Performance Metrics	37

LIST OF TABLES

Table No.	Table Name	Page No.
5.1	Testing feature selection for C5.0 Decision Tree	28
5.2	Hyperparameter Tuning for LSTM	29
5.3	Hyperparameter Tuning for XGBoost	30
6.1	Performance Metrics for LSTM and XGBoost Model	37

CHAPTER 1

INTRODUCTION

Traditional currency used in modern monetary systems has various benefits due to its scarcity, durability, and capacity to be divided and transferred. However, it has a few issues, such as the fact that governments control the money which can cause income inequality and hyperinflation. The second flaw with traditional currency is present due to the nature of the modern financial system which mainly works on ledgers where these ledgers are subject to manipulation and breach. This vulnerable nature of a ledger is a big drawback of traditional currency. Another issue is present during transferring traditional money. Money can be exchanged through checks, wire transfers, credit cards, or online services. However, due to the presence of a third party between the receiver and sender, like a financial institution, money transfers can cost between 6% and 10% and might take up to a week to complete depending on the destination country.

To tackle the issues with traditional currency, a new form of currency was introduced that can completely remove the middleman from a financial transaction between participants, bringing a transformation in how the economy operates. Bitcoin: A Peer-to-Peer (P2P) Electronic Cash System, a paper by a researcher named Satoshi Nakamoto [1] published in 2008, introduced the idea of peer-to-peer cash transfers of online payments without the involvement of any intermediary financial organisations. Satoshi gave an example of the concept of a decentralised chain of legitimate transactions (chain of blocks), which is dispersed among all network peers. Due to these properties, cryptocurrency solves a few disadvantages of the traditional financial system by making it global, immutable, encrypted, and decentralized.

1.1 Predicting Price of Ether

Bitcoin and other cryptocurrencies rapidly rose in popularity due to the aforementioned advantages. After Bitcoin, many other cryptocurrencies entered the market. For instance, Ether, introduced in 2015, is the second-largest cryptocurrency with a market worth of \$ 410 billion. Ether's price first began to skyrocket in spring 2017. At the time, each Ether token was trading for around \$10, but by the summer of 2017, it reached triple digits. It was not until recently that Ether reached new all-time highs. In February 2021, Ether's price hit \$1,770 before retracing back to around \$1,600, indicating an increase in popularity.

The rise in popularity of these coins has made cryptocurrency a means of investment for a lot of people. However, like any market, it is risky and highly speculative to invest without any knowledge. The price of Ether is governed by the market supply and demand - a rise in demand for Ether raises the price. Therefore, the market is closely studied by many to get general public opinion which is a factor that affects Ether price. One such major platform used is Twitter. Twitter is one of the most popular social media platforms, with over 300 million active users logging in at least once a month. It enables users to follow people or businesses whose content they enjoy reading, find articles about the biggest news and events of the day, or just connect with friends. It is very commonly studied in multiple test cases to gauge the opinions of the public in various domains. Hence, with a few modifications made to the process

of filtering tweets, the aim is to predict the general trend in the change of Ether's price. Although multiple factors affect the price and the price is very vulnerable and dynamic, machine and deep learning models have made the task a bit easier. Sentiment analysis can be performed on tweets to determine the general opinion of a tweet. Along with the polarity of a tweet, looking at how the tweet has affected the readers can potentially give even more accurate predictions. Hence, the project will apply two different machine learning algorithms and compare them to determine the method with better prediction accuracy.

1.2 Machine Learning Models

This section briefly describes some of the machine learning models implemented in the system.

1.2.1 C5.0 Classification Algorithm

The C5.0 algorithm is a decision tree-based classification algorithm. C5.0 is the industry standard decision tree-based algorithm for any type of classification problem or dataset. It is simpler and easily understandable compared to its counterparts in deep learning. The major challenge in implementing a C5.0 algorithm is the identification of the feature to split upon. C5.0 uses entropy to calculate the best choice of splitting of a dataset based on the chosen feature

Using the Entropy of a particular class level, information gain, which represents the importance of the feature, can be extracted. The higher the information gain of a feature, the better it is at creating homogenous groups after the dataset is split on that feature. Since decision trees can grow indefinitely long as the tree moves from general to specific features, overfitting might occur. Therefore, pruning can be introduced in a decision tree for size reduction and to avoid overfitting. C5.0 algorithm utilizes pruning by post-pruning the decision tree. This is done by overfitting the training dataset initially. Afterward, nodes and branches that have minimal effect on the classification results are removed resulting in a much better generalization of the feature set.

1.2.2 Valence Aware Dictionary Sentiment Reasoner (VADER)

Valence Aware Dictionary Sentiment Reasoner (VADER) is a model used for text sentiment analysis. VADER is sensitive to both the polarity and intensity of emotions. VADER can be applied directly to text data and is extremely accurate in handling social media content, such as sarcasm, slang, different languages, and emoticons. VADER is a lexicon-based and rule-based sentiment analysis approach that relies on a dictionary mapping lexical features to their corresponding emotion intensities, which are represented as sentiment scores. VADER algorithm does not classify texts as subjective or objective but rather only accounts for the polarity of the text itself. The lexicon dictionary of VADER was using Amazon's, Mechanical Turk. VADER requires lesser preprocessing of text data and processing capabilities when compared with deep learning models due to its dictionary structure.

1.2.3 Extreme Gradient Boosting (XGBoost)

Extreme Gradient Boosting (XGBoost) is a highly scalable, distributed gradient-boosted decision tree (GBDT) library. XGBoost is commonly used for regression, classification, and in some cases, ranking problems. It evolved from the concepts of decision trees, bagging, random forests, boosting mechanisms, and gradient boosting. It is a supervised machine

learning capable of ensemble learning, tree-pruning, and parallelization through GPU acceleration. Fig 1.1 [2] depicts the overall working of the XGBoost algorithm.

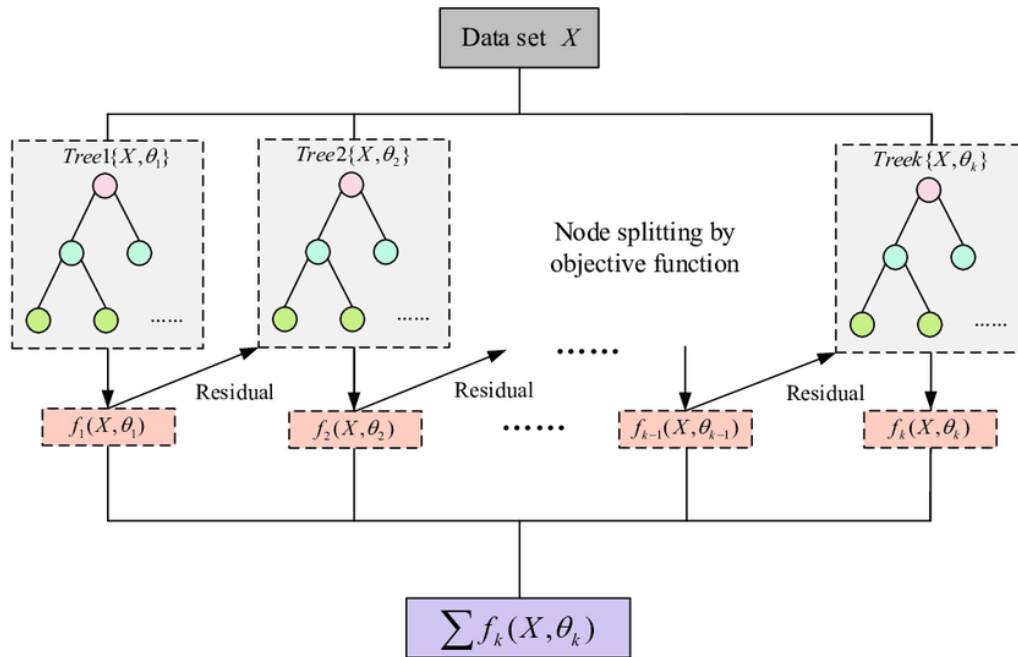


Figure 1.1 XGBoost flowchart

Generally, the numerical features are scaled and the categorical features are encoded before being used with the XGBoost algorithm. Instead of holding a single decision at the “leaf” nodes of a tree, XGBoost assigns scores to them. When the max depth of a tree is reached, XGBoost categorizes them based on certain thresholds using the scores assigned to the “leaf” nodes.

1.3 Deep Learning Models

This section describes briefly some of the deep learning models implemented in the system.

1.3.1 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) is based on recurrent neural networks (RNN) solving the issue of short-term memory that the RNN models suffer from due to the vanishing gradient problem. The vanishing gradient problem is when the weights of the node in the neural network do not get updated as the value of the gradient becomes extremely small and vanishes, resulting in short-term memory. This is solved by the LSTM using the gates mechanism by regulating the flow of information. The gates allow the LSTM model to learn the important data in the sequence and ignore the rest. This allows only relevant information to be passed to the next states of the LSTM cell maintaining better long-term memory compared to RNNs. Fig 1.2 [3] consists of a pictorial representation of the basic LSTM cell.

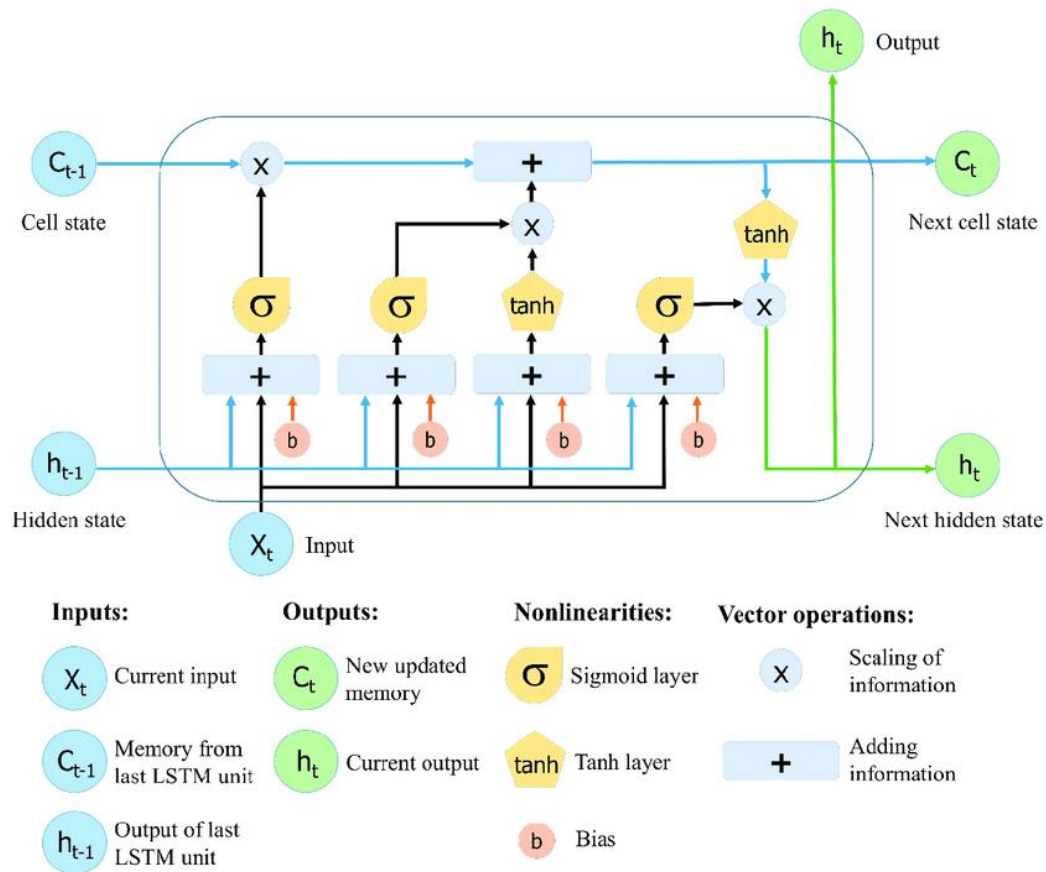


Figure 1.2 Basic LSTM cell

The basic LSTM cell represented in Fig 1.2 consists of three gates to regulate the flow of information: Forget gate, Input gate, and Output gate.

- I. **Forget Gate:** The forget gate decides the information passed on to the next cell state based on the previous hidden state and current input by applying a sigmoid function on the two values.
- II. **Input Gate:** The input gate captures the previous hidden state and current input which will be transformed by the sigmoid function and tanh function separately. The sigmoid output is responsible for deciding the importance of the tanh output
- III. **Output Gate:** The output gate performs pointwise multiplication between the forget vector and the cell state. The result is then pointwise added with the output of the input gate to update the new cell state.

Training requires a large dataset and it is computationally more intensive compared to RNNs and GRUs. But LSTM models present high capabilities in terms of capturing long-term and short-term trends.

1.4 Motivation

The news regarding the merge of Ethereum's Proof-of-Work (PoW) with Proof-of-Stake (PoS) is speculated to bring a new era to cryptocurrency. With 'The Merge' having the potential to reduce the energy consumption of Ethereum by 99%, many even speculate Ether to take the number one spot in cryptocurrency popularity soon. Concerning this information, it is a very ideal time to look at the investment opportunities related to Ether. Twitter plays a key role in predicting the demand for Ether and in accordance, the price of the crypto coin. However,

making predictions based on Twitter sentiment as it has been done so far may not give the most accurate results; due to the hype created with the new development in Ethereum, it seems many people have taken to Twitter to use “Ether” and “Ethereum” keywords to gain attention and spread spam. This kind of Twitter data might interfere with legitimate opinions and discussions regarding Ether. These spam tweets add no value to the discussion and only work as clickbait on Twitter users. Hence, additional steps to filter the dataset with spam filtering are implemented and the performance is measured.

1.5 Problem Statement

Cryptocurrencies are primarily used as a means of money exchange. However, Cryptocurrencies are viewed as an attractive investment opportunity by many, so people try to predict the price of cryptocurrencies like Ether to make smart investment choices. The price of Ether can be determined by studying the market demand and supply, similar to how the stock market functions. One of the best ways to determine user demand is by considering public opinion. To grasp the general opinions of Ether, Twitter can prove to be an effective platform since it is a famous microblogging platform that is very commonly used by the cryptocurrency investors community; many influential people use Twitter which can affect public opinion significantly. However, the high volume of tweets present on Twitter makes the task of analysing the sentiments of tweets and their impact on Ether prices very arduous to perform manually. Moreover, the presence of Twitter bots that post multiple spam tweets, along with the task of determining which tweets have a bigger significance in affecting public opinion makes the task of predicting even more complicated. Hence a solution must be implemented so that these factors are considered for predicting while accounting for a large volume of tweets.

1.6 Objective

The main objectives of the system are to:

- Help a potential buyer or seller make informed decisions while interacting with cryptocurrency, specifically Ether.
- Filter out spam and identify the tweets that impacts prices to help implement predictions for Ether prices.
- Analyze the filtered Twitter data using LSTM and XGBoost models, and predict Ether price.
- Compare LSTM and XGBoost models for Ether price prediction

1.7 Scope

Ether price prediction has the potential to be used in multiple contexts. Through the comparisons between LSTM and XGBoost for Ether price prediction, a system tailored for users to check the price predictions and calculated accuracies of the same could be modelled. The price predictions for Ether could be made for a greater range of dates and numerous datasets of different sizes. The precision of price fluctuation could also be improved using larger datasets. The application would also allow the analysis of the effects of opinions on social media of various platforms might have on Ether price fluctuations. Based on the impacts that major social media influencers cause, inferences can be made on the influence hierarchy

Ether. The predicted price fluctuations of Ether can also aid in predicting the prices of the tokens correlated to the Ethereum platform, a few of them being Basic Attention Token (BAT), Bancor (BNT), and the cryptocurrencies dependent on the Ethereum platform, notable ones being Polygon (MATIC), Chainlink (LINK) and Decentraland (MANA). Finally, the social media sentiments and price fluctuations allow the user to make informed investment decisions on Ethereum-based decentralized applications (dApps).

CHAPTER 2

LITERATURE STUDY

Cryptocurrencies are considered to be the future of online transactions. Since COVID-19, online trading has become a major domain, in particular cryptocurrency trading. Prediction of the prices of various cryptocurrencies has become a pivotal aspect of online trading. Many researchers and scholars have performed extensive and exhaustive research on predicting the variations in the prices of cryptocurrencies using numerous methods throughout the years. Currently, due to the increase in processing capabilities and the emergence of high-accuracy algorithms & models, machine learning and deep learning models are being preferred over statistical predictions.

P. Jay et al. [4] proposed a stochastic neural network model based on introducing randomness into the activation functions of Multi-Layer Perceptron (MLP) and Long Short-Term Memory (LSTM). This was done to simulate the cryptocurrency market's volatility by using deterministic models. The study concluded that the use of the stochastic version of the neural net models had performed better when compared with the deterministic versions in accuracy and validity.

Mohil Maheshkumar Patel et al. [5] improvised the basic LSTM model by implementing a hybrid model based on Gated Recurrent Unit (GRU) and LSTM by passing common input to the concatenated models and passing the models through a dense layer for the final output. The authors concluded that such model-based predictions would be applicable for short-term predictions and would not be ideal for long-term predictions due to the volatility of the cryptocurrency market.

Bidirectional LSTM (bi-LSTM) was proposed as one of the three models tested by Hamayel, Mohammad J. et al. [6] in predicting Ether (ETH), Bitcoin (BTC), and Litecoin (LTC) prices. After comparison with LSTM and GRU, bi-LSTM exhibited lower accuracy and correctness. GRU demonstrated the best accuracy in the price prediction of all three coins.

A hybrid artificial neural network of LSTM and Bayesian Optimization was considered a viable option in predicting BTC prices by Pour, E. S. et al. [7] The model was tested using different batch sizes and differences in the number of training epochs. The proposed model displayed considerably lower accuracy scores when compared with some of the other deep learning models mentioned previously in other research papers.

Andi et al [8] recommended implementing LSTM with logistic regression (LR) in predicting the price of BTC. Leveraging the forecast of BTC through normalization of the training dataset was the primary objective. The proposed model outperformed LR and Lasso algorithm by overcoming the issues of overfitting and errors when trained with a large dataset.

Prediction of cryptocurrency prices based on the seasonality of the datasets was studied by Y. Indulkar [9]. Different seasons were considered, and numerous changes in parameters were evaluated by considering Daily & Yearly periods, along with Monthly & Daily periods because of the insufficiencies in the dataset.

M. Shin et al. [10] implemented an ensemble-enabled LSTM with different time interval models for predicting BTC prices. This approach, which consisted of three different LSTM layers with independent and individual datasets based on different time intervals was advantageous in capturing long-term trends. Even though the proposed model aimed at analysing long-term trends, only short-duration predictions were possible, acknowledging instantaneous price fluctuations.

Since most of the research done previously suggested that deep learning models were better in predicting price fluctuations of cryptocurrency, and in particular LSTM and its variations, the parameters to be considered were studied in detail.

Initially, the relationship between the cryptocurrency to be predicted and the other cryptocurrencies were explored. S. Tanwar et al. [11] aimed at predicting the price of LTC and Zcash with inter-dependent relations between the chosen cryptocurrencies and their corresponding parent cryptocurrencies using the LSTM and GRU-based hybrid approach. Different window lengths were tested and the results showed definitive improvements in terms of accuracy for windows of smaller lengths.

Cross-Cryptocurrency Relationship Mining module (C²RM) was proposed to represent the synchronous and asynchronous factors among BTC and BTC's related altcoins that impact the price of BTC by Li, Panpan et al. [12] The authors utilized Dynamic Time Warping (DTW) algorithm and Lead-lag Variance Kernel (LVK) for deriving information about the influential factors resulting in more robust price predictions.

Another factor analysed was news and social media information, in particular, Twitter forums. Abraham, Jethin et al. [13], suggested the usage of tweet volume and sentiment analysis for better results. Sentiment analysis was performed on the tweets collected to gauge the direction of price change, reinforced by the tweet data volume and Google Trends data.

Another ensemble deep learning model was recommended by Ye, Zi et al. [14] for predicting BTC price fluctuations. Along with a hybrid model of LSTM and GRU, sentiment indexes of tweets were also considered in price prediction to improve the accuracy of decisions. Valence Aware Dictionary for Sentiment Reasoning (VADER) was used for the initial assignment of sentiment scores which was further used for small granularity sentiment indicators calculation.

VADER was analysed to a greater extent concerning the analysis of tweets by Pano, Toni et al. [15] The authors intended on applying VADER for sentiment analysis of tweets during COVID-19 for estimating the price direction of BTC. It was concluded that VADER scores obtained from text processing tweets have a significant short-term correlation with BTC prices but were incapable of maintaining the correlation over long periods.

Abayomi-Alli A et al. [16] studied Yahoo-Yahoo Hash-Tag Tweets by implementing sentiment analysis and opinion mining algorithms. VADER, Multidimensional Scaling (MDS) graphs, Liu Hu method, Latent Semantic Indexing (LSI), and Latent Dirichlet Allocation (LDA) were applied for selecting the better-suited algorithm. VADER performed the best among the other algorithms as it is better attuned to analysing social media content.

Other sentiment analysis models, such as lexicon-based, machine learning-based, and hybrid approaches were compared against each other and studied in detail by Maroune Birjali et al. [17] After considering many possible parameters, dataset pre-processing and optimization

techniques were inspected. Spam filtering of tweets was assessed for better dataset pre-processing.

Tida, Vijay Srinivas et al. [18] detected spam emails using transfer learning of the Bidirectional Encoder Representations from Transformers (BERT) model trained on four different datasets. Even though the model portrayed high accuracy due to training with multiple datasets, BERT requires large datasets for training and enormous processing capabilities.

The effects of spam and bot activity on stock microblogs on Twitter have been explained by Cresci, Stefano et al. [19] Due to inflations and price bubbles caused by such perpetrators, the elimination of spam tweets would be advantageous in training the price prediction models and analysing cryptocurrency price trends.

Real-Time and near real-time spam detection has been implemented through numerous methods such as decision tree-based algorithms, K nearest neighbours (kNN) based algorithms, Boosting algorithms, Bayesian algorithms, and deep neural network architectures by Rodrigues, Anisha [20] and Sun, Nan [21]. The study concluded that decision tree-based algorithms, namely C5.0 and Random Forest (RF), and the deep neural network model, namely LSTM, performed the best among all others.

A generalized approach toward spam filtering comprising work emails and comments was introduced by P. Garg et al. [22]. The different features of the various types of spam were discussed and several detection techniques for spam emails and spam comments were examined.

Lexicon and learn-based techniques are suggested by Anum Waheed et al. [23] Fake news websites recorded by Kaggle were used as the dataset for training. Three different hybrid models namely, the Naive Bayes (NB) model, RF model, and recurrent convolutional neural network (RCNN), each combined with the lexicon-based approach, were tested against the web scrapped dataset, with the RCNN hybrid model exhibiting the highest training score and validation score.

The study served the purpose of understanding the various approaches to predicting prices for different cryptocurrencies, filtering spam especially in the context of social media and communication contexts, and the metrics used to compare the performance of multiple models. Results of the surveyed research reveals that LSTM based approaches and Decision Tree based approaches perform better for time series analysis and prediction, in the case of Deep Learning and Machine Learning methods respectively. It is also understood that when medium sized datasets are used, decision tree-based algorithms, like C5.0, show good performance for supervised classification tasks like filtering spam from social network data, and VADER shows better performance and accuracy for sentiment analysis.

CHAPTER 3

SYSTEM ANALYSIS

This chapter focuses on the hardware and software requirements essential to develop, train, test, implement the system and its modules. It also discusses the datasets used, along with the feasibility of the system.

3.1 Hardware Requirements

The hardware requirements mentioned are the minimum hardware settings required to deploy the project in a computer system.

- I. **OS:** Windows, Mac, or Ubuntu
- II. **Minimum Cores:** 6
- III. **Graphical Processor:** 2 GB RAM
- IV. **Minimum System RAM:** 16 GB
- V. **Hard Disk Space:** 30 GB SSD

3.2 Software Requirements

The software requirements mentioned are the minimum software setting necessary to run the project in a computer system.

- I. R 4.0 or above
- II. R Libraries:
 - i. **Tweetbotornot2:**
Tweetbotornot2 is a package that uses over 100 features derived from user attributes, tweet patterns, and text-based patterns to provide an estimated probability of an account being a bot.
 - ii. **Data.table:**
Data.table is a package is used for working with tabular data in R. It provides the data.table object, an version of the default data.frame object. It provides a means of processing large datasets much faster.
 - iii. **Rtweet:**
rtweet provides users a range of functions designed to extract data from Twitter's REST and streaming APIs.
- III. Python 3.8 or above
- IV. Google Colaboratory
- V. Python Libraries:
 - i. **NumPy:**
NumPy is a library for the Python language. It is a general-purpose array processing package that provides support and high-level mathematical operations for working on large, multi-dimensional arrays and matrices.

ii. **Snsrape v.0.4.3+:**

Snsrape is a Social Network Service (SNS) Scraper that scrapes data from popular social networking sites like Facebook, Instagram, Twitter, etc.

iii. **pandas v.1.4.2+:**

pandas is a Python library that is used to analyze and manipulate data. It provides fast and flexible data structures to ease working with tabular data.

iv. **PyTorch v.1.7.0+:**

PyTorch is a Python library based on the Torch library. It provides Tensor computing capabilities and deep neural networks.

v. **Sci-kit learn:**

Scikit-learn is a machine learning library for Python that provides various algorithms. It works alongside other popular Python libraries like NumPy, Pandas, SciPy, Matplotlib, etc.

vi. **Google Colaboratory:**

Google Colaboratory is a cloud-run Jupyter notebook environment that allows collaboration on code without requiring additional setup. It supports many popular machine learning libraries and even provides facilities to switch runtime environments.

vii. **Natural Language ToolKit (NLTK):**

NLTK is a suite of open-source Python modules, data sets, and tutorials supporting research and development in Natural Language Processing

3.3 Datasets

The dataset comprises tweets on the topic 'Ethereum' and 'Ether' only. Tweets are collected using a scraper for social networking services called snsrape. [24] The query is set to search for tweets containing the keywords 'Ether' or 'Ethereum'. The tweets are collected across different dates to ensure better learning. The dataset for training spam and ham tweets is obtained by considering the tweets that contain malicious links in them and the tweets posted by bot accounts. Along with this, a dataset of Ether prices from September 9, 2017, to March 25, 2022, has also been obtained from Kaggle [25], and is used to test the prediction.

3.4 Functional Requirements

The functional requirements of the system are:

- Take in input as a dataset of tweets, and their attributes
- Filter the spam tweets out of the dataset and produce only ham tweets
- Perform sentiment analysis on the ham tweets and provide them with a polarity
- Take the ham tweets along with their polarity and provide a prediction in the direction of fluctuation of Ether prices

3.5 Non-Functional Requirements

Non-functional requirements are requirements that specify the criteria that can be used to judge the operation of a system rather than the behavior of a system.

I. Usability:

The system must be able to identify spam tweets, and predict the direction of fluctuation of price change seamlessly and without errors.

II. Efficiency:

The system must be able to predict the direction of fluctuation of Ether prices more accurately, in lesser time.

III. Correctness:

The output of the system matches the expectations outlined in the requirements, and the system operates without failure.

3.6 Feasibility:

I. Economic Feasibility:

Training of modules and collection of the dataset requires hardware facilities like faster RAM, processors, GPUs, etc. The libraries and tools used for the development of the system are open-source and under a general-purpose license. The training, testing, and deploying of these modules do not require special additional peripherals.

II. Technical Feasibility:

The required resources and tools to develop and run the system are available on hand. They have sufficient documentation and support to perform maintenance and upgradation of the system if necessary.

III. Operational Feasibility:

The system satisfies operational feasibility in setups that satisfy the aforementioned hardware and software requirements. Additional or improved resources will improve the operational functioning of the system.

CHAPTER 4

SYSTEM DESIGN AND IMPLEMENTATION

This chapter describes about the system architecture and the models trained using different machine learning algorithms implemented.

4.1 High Level System Architecture

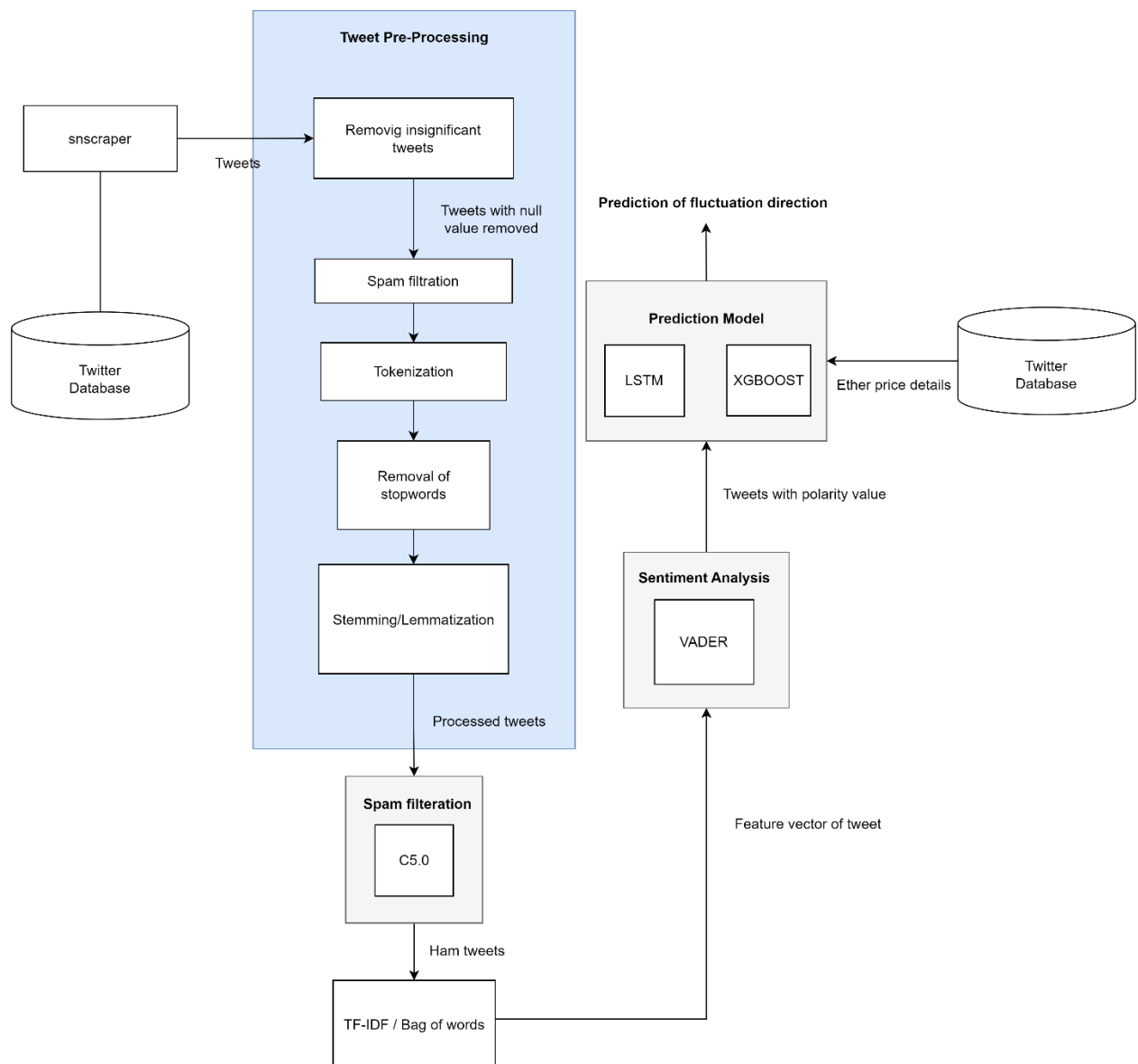


Figure 4.1 High level view of proposed solution

Fig 4.1 depicts the high-level overview of the proposed model. The system takes the preprocessed tweets as input and feeds into the C5.0 model that classifies the tweets as spam or ham using decision trees that perform splits based on the feature that provides maximum

information gain. The ham tweets are then converted to feature vectors and undergo sentiment analysis. The VADER algorithm provides the tweets with a polarity value based on the sentiment detected. These tweets are provided as input to both LSTM and XGBoost models respectively, to perform prediction of price fluctuation direction.

4.2 System Implementation

4.2.1 Tweet Scraping with snsrape

Web scraping is the process of extracting content and data from a website using bots. Web scraping extracts underlying HTML code and, with it, data stored in a database. Social media scraping is a process of automatically extracting data from social media platforms such as Twitter, Facebook, and Instagram. The resources provided by the platforms themselves for accessing information are in the form of authenticated APIs that have heavy request caps or long request refresh rates imposed on them. To circumvent this, a social network scraper named snsrape [24] was used. snsrape allows the collection of data from social networks like Facebook, Instagram, Mastodon, Reddit, Telegram, Twitter, VKontakte and Weibo (Sina Weibo).

```
for i, tweet in enumerate(snmodules.twitter.TwitterSearchScraper('(Ethereum OR
Ether) lang:en since:2021-10-11 until:2021-10-21').get_items()):
    print(i)
    username = tweet.user.username
    user_id = tweet.user.id
    # user_profile = twitter_scraper.Profile('username').to_dict()
    user_verified = tweet.user.verified
    user_followers = tweet.user.followersCount
    user_friends = tweet.user.friendsCount
    tweetlinks = tweet.links[0] if tweet.links !=None else None
    tweet_links = tweetlinks.url if tweetlinks !=None else None # links in the
tweet
    tweet_id = tweet.id
    tweet_timestamp = tweet.date
    tweet_text = tweet.renderedContent
    tweet_likes = tweet.likeCount
    tweet_replies = tweet.replyCount
    tweet_url = tweet.url # link of the tweet

    tweet_rts = tweet.retweetCount
    tweet_quotes = tweet.quoteCount

    tweet_row = [username, user_id, user_verified, user_followers, user_friends,
tweet_id, tweet_timestamp, tweet_text, tweet_links ,tweet_url , tweet_likes,
tweet_rts, tweet_replies, tweet_quotes]
    df_rows.append(tweet_row)

tweets_df = pd.DataFrame(df_rows, columns=columns)
```

Figure 4.2 Using TwitterSearchScraper module of snsrape to scrape tweet and user data

Fig 4.2 depicts the usage of TwitterSearchScraper module to extract tweet and user data. The TwitterSearchScraper function of the snsrape Twitter module accepts a query defined by the Twitter API documentation, and returns the result of the query as a Tweet object. User-based and tweet-based attributes can be obtained from this tweet object.

4.2.2 Tweet Preprocessing

After the tweets are spam filtered, a portion of tweets that are deemed to be insignificant are removed. The tweets are considered to be insignificant if they have no likes, no comments, no retweets, and are not mentioned in any tweets as a quote. This is done to reduce the noise of irrelevant data in the training dataset. The tweets which are obtained after spam filtration

are to be pre-processed using some basic techniques. This pre-processing step is vital since the tweets that are obtained are from a database scraping tool and not from a benchmark dataset.

1. Tokenization of Tweets:

Tokenization helps to focus on each individual word and its characteristic for processing. The tweets procured are initially tokenized, such that each word in the tweet is broken down into a token for further pre-processing.

2. Removal of @ mentions:

Tweets have the option of including user ids in the tweet body as a form of tagging that particular user in the tweet. Since the mentions have less significance regarding the emotion and weight of the tweet, these mentions are removed from the tweet body.

3. Removal of stop words:

Stop words are functional words that are useful syntactically and grammatically for the sentence to be meaningful. They are topic-neutral and do not add any value to the topic of the sentence or convey any emotion. These stop words are to be removed for more accurate sentiment analysis.

4. Lemmatization:

Lemmatization can be defined as the process of grouping different inflected forms of a particular word so that the word can be analyzed once as a single item rather than as multiple different entities. Lemmatization is done so that the accuracy and speed of the sentimental analysis algorithm can be improved.

5. Sentence Forming:

After the pre-processing steps are performed, the words are combined back together into a sentential form for sentimental analysis. The input for the sentimental analysis model is of sentential form since the context of each word contributes towards the entire polarity of the sentence.


```

#extract tweets
x = csvFile['Tweet Text']
for lines in (x):
    #remove mentions
    char = '@'
    lines = " ".join(
        filter(lambda word: not word.startswith(char), lines.split()))
    word_tokens = word_tokenize(lines)

    #removal of stop words
    filtered_sentence = [w for w in word_tokens if not w.lower() in stop_words]
    filtered_sentence = []
    flag = 0
    for w in word_tokens:
        if w not in stop_words:
            filtered_sentence.append(w)
    lines = ''

    #lemmatization
    lemmatizer = WordNetLemmatizer()
    for w in filtered_sentence:
        w = lemmatizer.lemmatize(w)

    for w in filtered_sentence:
        lines = lines + (w) + ' '

```

Figure 4.3 Tweet Pre-processing

A snippet of the tweet pre-processing steps has been shown in Fig 4.3.

4.2.3 C5.0 Classification Algorithm

C5.0 is a supervised decision tree algorithm that is currently the industry standard for decision trees. C5.0 was built on the concept of information gain and entropy of each attribute in the dataset. Entropy measures the uncertainty within a group of observations and is measured between 0 and 1 (0 being completely impure and 1 being completely pure).

$$\text{Entropy}(S) = \sum_{i=1}^c -p_i \log_2(p_i) \quad (1)$$

For a given dataset (S), the term c signifies the different classes available, and p_i signifies the proportions of values within the class level i.

Calculating the entropy of a particular class level as defined in formula (1), information gain, which represents the importance of the feature, can be extracted. Information gain of a particular feature F is defined as

$$\text{InfoGain}(F) = \text{Entropy}(S_1) - \text{Entropy}(S_2) \quad (2)$$

Information gain, as shown in formula (2), is a measure of how relevant the feature is in the classification of the object's class. The tree is built and the nodes are split based on the information gain from the features in the dataset which sorts the tree's features hierarchically.

The C5.0 machine learning algorithm is opinionated about pruning, such that it prunes the decision tree based on rules automatically. C5.0 decomposes the tree structure into a set of mutually exclusive rules and the rules are pruned and modified into a smaller set of overlapping rules.

A tweet is considered to be spam if it contains a malicious URL or is made by a bot account. The maliciousness of a link can be in the form of malware, social engineering, unwanted software, potentially harmful applications, or phishing links. The tweets made by bot accounts

on Twitter usually contain such links. Bots on twitter can significantly affect user behaviour, and even elicit engagement from non-bot twitter accounts.

The creation of a labelled dataset for C5.0 was done using various APIs. The IsItPhish API and Google Safebrowsing API accepted payloads containing URLs found in the tweets that were collected. These links were then processed by the APIs and the result of the analysis returned in the json response body. The labels of the links were then extracted from the json response list and appended to the tweet dataset.

```
url = "https://api.exerra.xyz/scam?url="

payload={}
headers = {
    'Accept': 'application/json'
}

outputfile1 = "Output" + str(random()) + ".txt"
i=1
for link in urllist:
    print(i)
    if i%1000==0:
        sleep(60)
    geturl = url + link

    response = requests.request("GET", geturl, headers=headers,
data=payload)
    print(response.text)
    with open(outputfile1, "a") as text_file:
        text_file.write(str(i) + " " + str(response.text) + ",")
    i+=1
```

Figure 4.4 Using IsItPhish API to check links present in tweets

Fig 4.4 displays a snippet of the usage of IsItPhish API for verifying the links present in each tweet. The links were verified and stored in a text file.

```
for i in range(0, q, 500):
    if limit == 3:
        time.sleep(60)
        limit = 0

    threatEntries = []
    j=1
    while j+500 < n and j<i+500:
        threatEntries.append({"url": urllist[j]})
        j+=1

    print(len(threatEntries), threatEntries[0])
    post_body = {
        "client": {
            "clientId": "SEM7projectwork",
            "clientVersion": "0.0.1"
        },
        "threatInfo": {
            "threatTypes": ["MALWARE", "SOCIAL_ENGINEERING", "UNWANTED_SOFTWARE",
"THREAT_TYPE_UNSPECIFIED", "POTENTIALLY_HARMFUL_APPLICATION"],
            "platformTypes": ["WINDOWS"],
            "threatEntryTypes": ["URL"],
            "threatEntries": threatEntries
        }
    }

    response = requests.post(
        url=post_url,
        headers=post_headers,
        json=post_body
    )
    print(response.text)
```

Figure 4.5 Using GoogleSafeBrowsing API to check links present in tweets

The GoogleSafeBrowsing API was utilized to verify the links present in the tweets as shown in Fig 4.5.

Tweetbotornot2 considers over a hundred different features derived from user-level attributes (names, profile information such as URL, description, and location, account creation date, number and rate of statuses, favoured tweets, lists, friends, and followers, and account

creation date), top-level tweeting patterns (frequency, proportion, and timing of pure/original tweets, quoted statuses, retweets, number of favourites, retweets) and text-based patterns in user tweets (number of hashtags, mentions, and links, length of tweets, punctuation, word complexity, etc.). Tweetbotornot also provides an interface to access the Botometer API [26], which takes in user data from the Twitter API result in its request. Tweetbotornot, however, performs predictions and provides a probability score for a user account being a bot by taking in usernames or user ids of Twitter accounts as input as shown in Fig 4.6 and Fig 4.7.

```
library(tweetbotornot2)
library(rtweet)

library(data.table)

packageVersion('rtweet')

screen_names <- usernamelist[[1]][1:10]
p2 <- predict_botometer(users = screen_names, key =
"67490ae44amsh2bddb3621208d42p1c61b9jsn266682f0500f", set_key = TRUE, parse = TRUE,
verbose = TRUE)
fwrite(p1, "botometerwithtweetbot.csv")
```

Figure 4.6 Using predict_botometer() to access Botometer API

```
library(tweetbotornot2)
library(rtweet)
packageVersion('rtweet')

screen_names <- usernamelist[[1]][1:1000]

screen_names
p1 <- predict_bot(screen_names)
fwrite(p1, "1 to 1000 Unique Usernames Spam.csv")
library(data.table)
```

Figure 4.7 Using predict_bot() to access Tweetbotornot2 predictions

The C5.0 package under a General-Purpose License is provided through the R programming language. After package initialization and loading of training data, the dataset is randomized to better train the model. The spam label is then label encoded and set as the dependent variable.

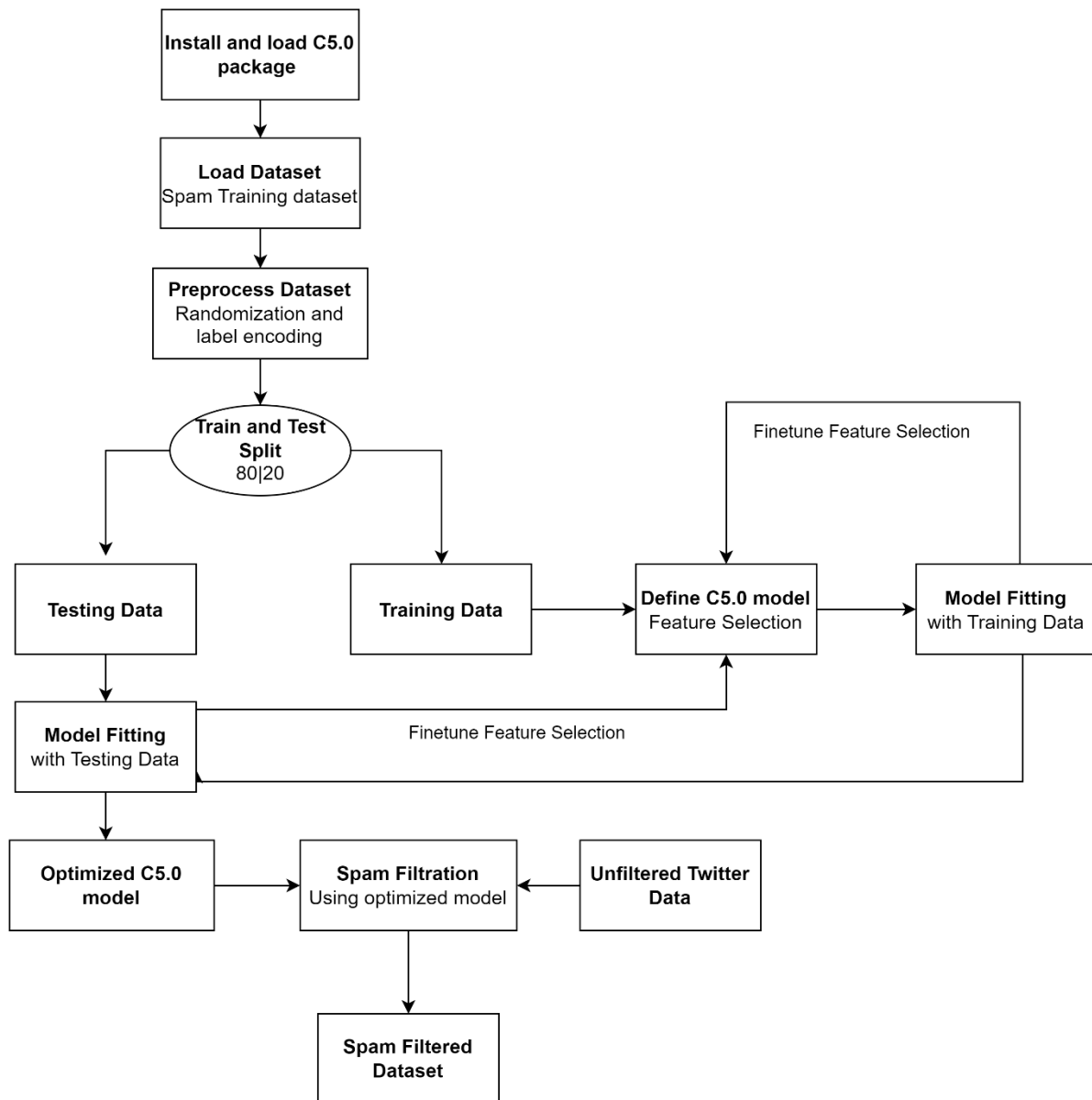


Figure 4.8 Overview of C5.0 implementation

The model is defined based on the input parameters of tweet characteristics and user characteristics which is used by the C5.0 algorithm to describe the tree structure as depicted in the snippet of code in Fig 4.9. The tree structure is based on rules formulated by the model using the input parameters. The tree is pruned by ignoring repetitive and specific decisions to decrease the length of the tree, thereby avoiding overfitting. The dataset is used by the C5.0 algorithm to derive rules from the input parameters to classify the data into spam and ham.

```

datar[1,c("User.Id", "User.Verified", "User.Followers", "User.Following",
"Tweet.Text", "Tweet.Links", "Tweet.Likes", "Tweet.Retweet.Count",
"Tweet.Reply.Count", "Tweet.Quote.Count")]

m2 <- C5.0(datar[1:60000, c( "User.Verified", "User.Followers", "User.Following",
"Tweet.Likes", "Tweet.Reply.Count")], datar[1:60000, 15])

summary(m2)|

```

Figure 4.9 C5.0 model definition and model fitting

After the model is trained, the model is implemented on the dataset to filter out spam data. The final dataset obtained is used for training the LSTM and XGBoost model. The overview of the entire C5.0 implementation is represented in Fig 4.8.

4.2.4 Sentiment Analysis with VADER

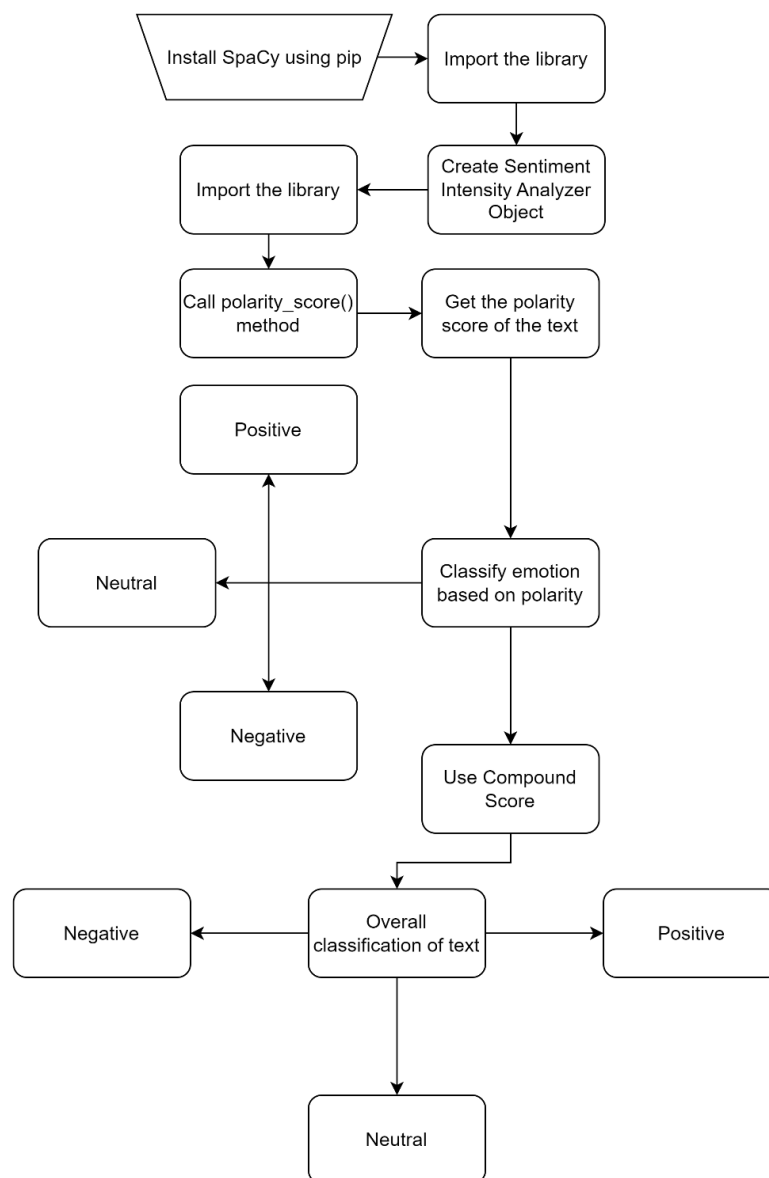


Figure 4.10 Overview of VADER implementation

Sentence-level sentimental analysis is the extraction of the opinion (sentiment) described by the sentence. The sentiment of the overall sentence could be positive, negative, or neutral and this is captured using Valence Aware Dictionary and Sentiment Reasoner (VADER) algorithm. VADER is a lexicon and rule-based sentiment analysis tool which is specifically designed to capture the sentiments expressed in social media. VADER works by assigning a valence score to each word under consideration by observation and from previous experience.

The valence score ranges from -4 to +4 with -4 being the most negative sentiment, +4 being the most positive sentiment, and 0 being the neutral midpoint. These valence scores are pretrained on Amazon Mechanical Turk (MTurk) dataset. This algorithm is based on a dictionary that maps words and numerous other lexical features common to social media such as emoticons, acronyms, slang, and many other rules called a heuristic. The primary heuristics of VADER are punctuation, capitalization, degree modifiers, conjunctions, and negations.

The compound score is computed by summing the valence score of each word in the lexicon and normalizing it between -1 for most negative and +1 for most positive. The overall process of sentiment analysis using VADER has been described in Fig 4.10. These compound scores depict the sentiment of the tweets which are added to the dataset for the prediction of Ether price fluctuations by the deep learning and machine learning models.

```
def sentiment_scores(sentence):
    sid_obj = SentimentIntensityAnalyzer()

    sentiment_dict = sid_obj.polarity_scores(sentence)
    # print("Overall sentiment dictionary is : ", sentiment_dict)
    # print("sentence was rated as ", sentiment_dict['neg']*100, "% Negative")
    negative.append(sentiment_dict['neg']*100)
    # print("sentence was rated as ", sentiment_dict['neu']*100, "% Neutral")
    neutral.append(sentiment_dict['neu']*100)
    # print("sentence was rated as ", sentiment_dict['pos']*100, "% Positive")
    positive.append(sentiment_dict['pos']*100)

    if sentiment_dict['compound'] >= 0.05:
        # print("Positive")
        polarity.append("Positive")

    elif sentiment_dict['compound'] <= - 0.05:
        # print("Negative")
        polarity.append("Negative")

    else:
        # print("Neutral")
        polarity.append("Neutral")
```

Figure 4.11 Using VADER to assign polarity scores to tweets

The compound scores are obtained for the spam filtered tweets and the opinion of the tweet along with its intensity is calculated based on the positive and negative sentiment of the tweet as shown in Fig 4.11.

4.2.5 Data Condensation

Before the tweets are fed into the models, the tweets are condensed into a lower dimension for easier processing and better mapping of input and output features. The tweets are bunched together based on dates, such that, if there are 5000 tweets per day, all their attributes are scaled and organized as a single row representing a particular day's opinion. Then the numeric values to be considered are typecasted as floats; categorical data like whether a user is verified or not is encoded using label encoding; polarity is taken as a numeric value by taking a difference between the positive and negative values. Further the columns are then condensed together based on the correlation that each column has with the dependent variable close price. An aggregate score is calculated for each tweet which takes into

consideration the polarity, user verified, user followers, user following, tweet likes, tweet reply and tweet quote count and summarizes all this information into a singular value as depicted in the code snippet in Fig 4.12. This aggregate value assigns a numeric value to indicates how strongly a sentiment for each tweet has spread.

```
df = df.groupby('Tweet Date').mean()

df["Aggregate"] = df["Polarity"] + (0.31*df["User Verified"] + 0.30*df["User Followers"] -
0.06*df["User Following"] + 0.5*df["Tweet Likes"] + 0.22*df["Tweet Retweet Count"] + 0.21*df
["Tweet Reply Count"] + 0.05*df["Tweet Quote Count"])

df2=df[["Aggregate", "Open", "Close"]]
```

Figure 4.12 Tweet Condensation

The minimized version of the twitter data which consists of the date, the aggregate of the tweet sentiment, the open price of Ether on that particular date, and the close price, is then used for prediction.

4.2.6 Prediction model using LSTM

The LSTM model is used to predict the closing prices of a day given the input dataset. This input dataset contains condensed information about the tweet (likes count, reply count, retweets count), the publisher of the tweet (followers count, user following count, verified), polarity score calculated for the tweet and Ether price for the respective day the tweet was posted. The dataframe is further pre-processed by scaling of data columns using Standard Scaler. Normalization is done using a standard scalar that is taken from the sklearn module. This standard scalar is applied on the entire dataframe so that the model works with smaller manageable numbers and will not require a very high computational power. LSTM requires information to be passed as time intervals.

Considering h hidden unit dimension, n batch size, and d inputs, the input to the LSTM cell is $X_t \in R^{n*d}$ and the hidden state of the previous time step is $H_{t-1} \in R^{n*h}$. The output of the cell at time step t is $O_t \in R^{n*o}$. Accordingly, the gates at the time step t are defined as:

$$I_t = \sigma(X_t W_{xi} + H_{t-1} W_{hi} + b_i) \quad (3)$$

$$F_t = \sigma(X_t W_{xf} + H_{t-1} W_{hf} + b_f) \quad (4)$$

$$O_t = \sigma(X_t W_{xo} + H_{t-1} W_{ho} + b_o) \quad (5)$$

where $W_{xi}, W_{xf}, W_{xo} \in R^{n*h}$, are the weights of the input to the LSTM cell at different gates and $W_{hi}, W_{hf}, W_{ho} \in R^{h*h}$, are the weights of the hidden states with bias parameters $b_i, b_f, b_o \in R^{1*h}$ for each gate. The input gate, forget gate, and output gate is designed as shown in formulae (3), (4), and (5).

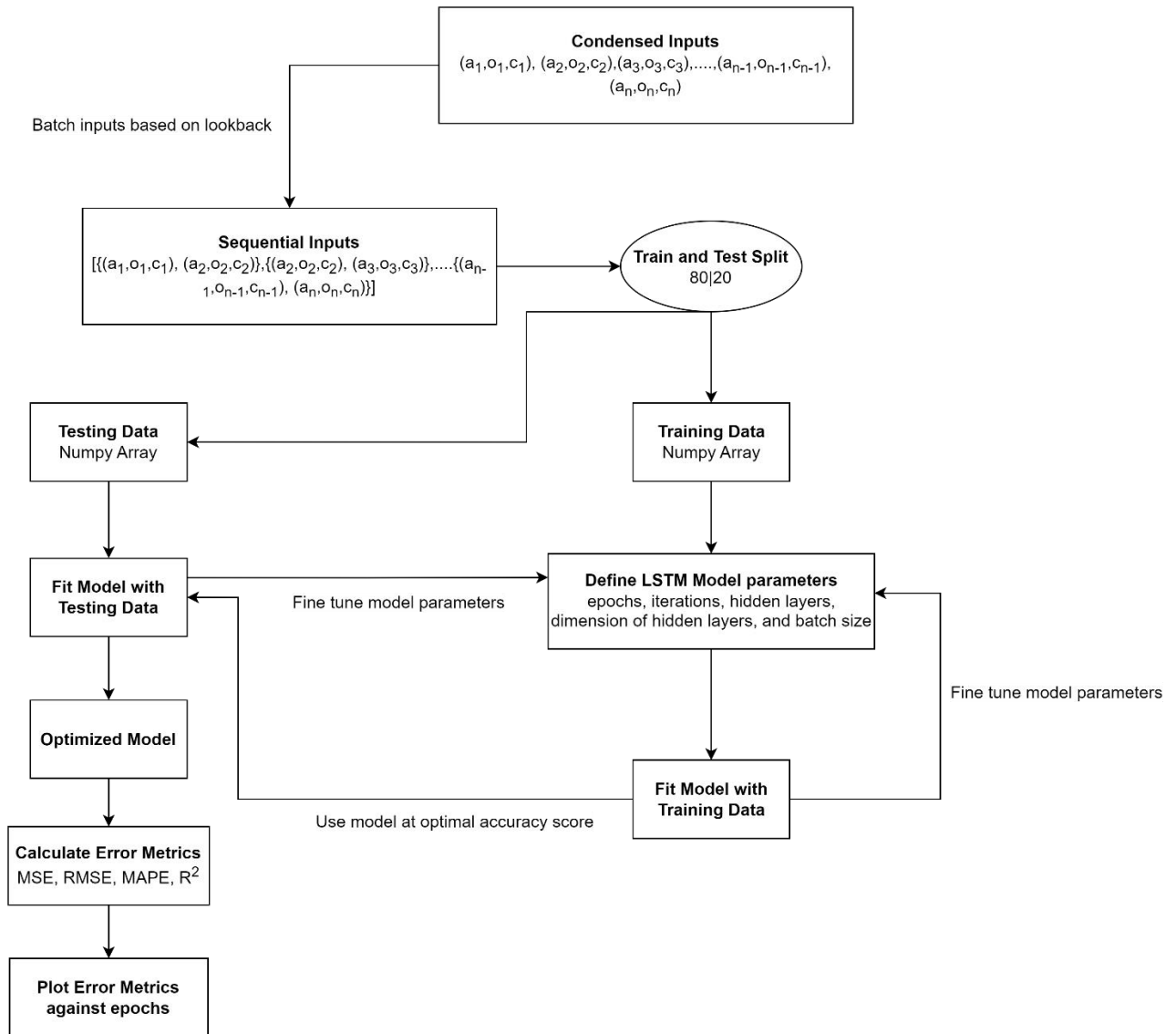


Figure 4.13 Working of LSTM model

The overall implementation of the LSTM model has been portrayed in Fig 4.13. The LSTM model is implemented using the torch package. In the torch package, the nn module is used for the definition of the LSTM network. Within the LSTM class, the number of inputs, hidden dimension, number of layers, and the output dimensions are chosen to be the hyperparameters. The forward pass of the LSTM model is implemented to reinitialize the gradients of the hidden state and the cell state at every iteration to disallow the errors from backpropagating till the end of the network.

The lookback hyperparameter is manually set by the user based on results obtained by trial and error as shown in Fig 4.14. The batch size, number of epochs, and iterations per epoch are assigned to overcome underfitting of the model and overfitting based on training data. Data loaders are utilized for batching data based on lookback and is useful for faster iteration through the dataset. The input dimension (X_i) in our model is [3,2] due to the 3 parameters and setting lookback as 2. We consider the output dimension (O_i) to be 1 which is the predicted close prices.


```

lookback = 2
data_independent=[]
data_dependent=[]
for index in range(len(df)-lookback):
    temp1 = []
    temp1 = (df.iloc[index:index+lookback,:-1])
    data_independent.append(temp1)
    data_dependent.append(df.iloc[index:index+lookback,-1:])

data_independent=np.array(data_independent)
data_dependent=np.array(data_dependent)
test_set_size = int(np.round(0.2 * data_independent.shape[0]))
train_set_size = data_independent.shape[0] - (test_set_size)

x_train = data_independent[:train_set_size]
y_train = data_dependent[:train_set_size,-1]

x_test  = data_independent [train_set_size:]
y_test  = data_dependent [train_set_size:-1]

```

Figure 4.14 LSTM Sequence Building

The input dimension is set to 3 to include the aggregate, open price, and close price of a particular day. The output dimension is set to 1 to predict the next day's price. The LSTM model defined uses Adam optimizer as the optimization algorithm with a low learning rate of 0.01 to avoid oscillation around local minima. The error function used for optimization is Mean Square Error (MSE) which described the training of the model accurately. Fig 4.15 shows the hyperparameter setting of the LSTM model.

```

input_dim = 3
hidden_dim = 60
num_layers = 2
output_dim = 1

class LSTM(nn.Module):
    def __init__(self, input_dim, hidden_dim, num_layers, output_dim):
        super(LSTM, self).__init__()
        # Hidden dimensions

        self.input_dim = input_dim
        self.hidden_dim = hidden_dim

        # Number of hidden layers
        self.num_layers = num_layers

        # Building your LSTM
        # batch_first=True causes input/output tensors to be of shape
        # (batch_dim, seq_dim, feature_dim)
        self.lstm = nn.LSTM(input_dim, hidden_dim, num_layers, batch_first=True)

        # Readout layer
        self.fc = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        # Initialize hidden state with zeros
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_dim).requires_grad_()

        # Initialize cell state
        c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_dim).requires_grad_()
        # One time step
        # We need to detach as we are doing truncated backpropagation through time (BPTT)
        # If we don't, we'll backprop all the way to the start even after going through another batch
        out, (hn, cn) = self.lstm(x, (h0.detach(), c0.detach()))

        # Index hidden state of last time step
        # out.size() --> 100, 28, 100
        # out[:, -1, :] --> 100, 100 --> just want last time step hidden states!
        out = self.fc(out[:, -1, :])
        # out.size() --> 100, 10
        return out

model = LSTM(input_dim = input_dim, hidden_dim = hidden_dim, output_dim = output_dim, num_layers =
num_layers)

loss_fn = torch.nn.MSELoss(reduction='mean')

optimiser = torch.optim.Adam(model.parameters(), lr=0.001)

```

Figure 4.15 LSTM Model Definition

4.2.7 Prediction model using XGBoost

Similar to LSTM, given the input dataset, the XGBoost model is used to forecast the closing prices of a day. This input dataset includes data on the tweet (such as the number of likes, replies, and retweets), data on the tweet's publisher (such as the number of followers and verified users), a score for the message's polarity, and the price of Ether on the day the tweet was posted. The data is processed in a similar manner as that of LSTM - the data is appropriately typecasted; polarity for each tweet is calculate; an aggregate is taken taking both polarity and tweet information into consideration; tweets are grouped by date and an average polarity is taken. The data is then normalized and split into training and test data, like in LSTM. The overall structure of the XGBoost model is described in Fig 4.18.

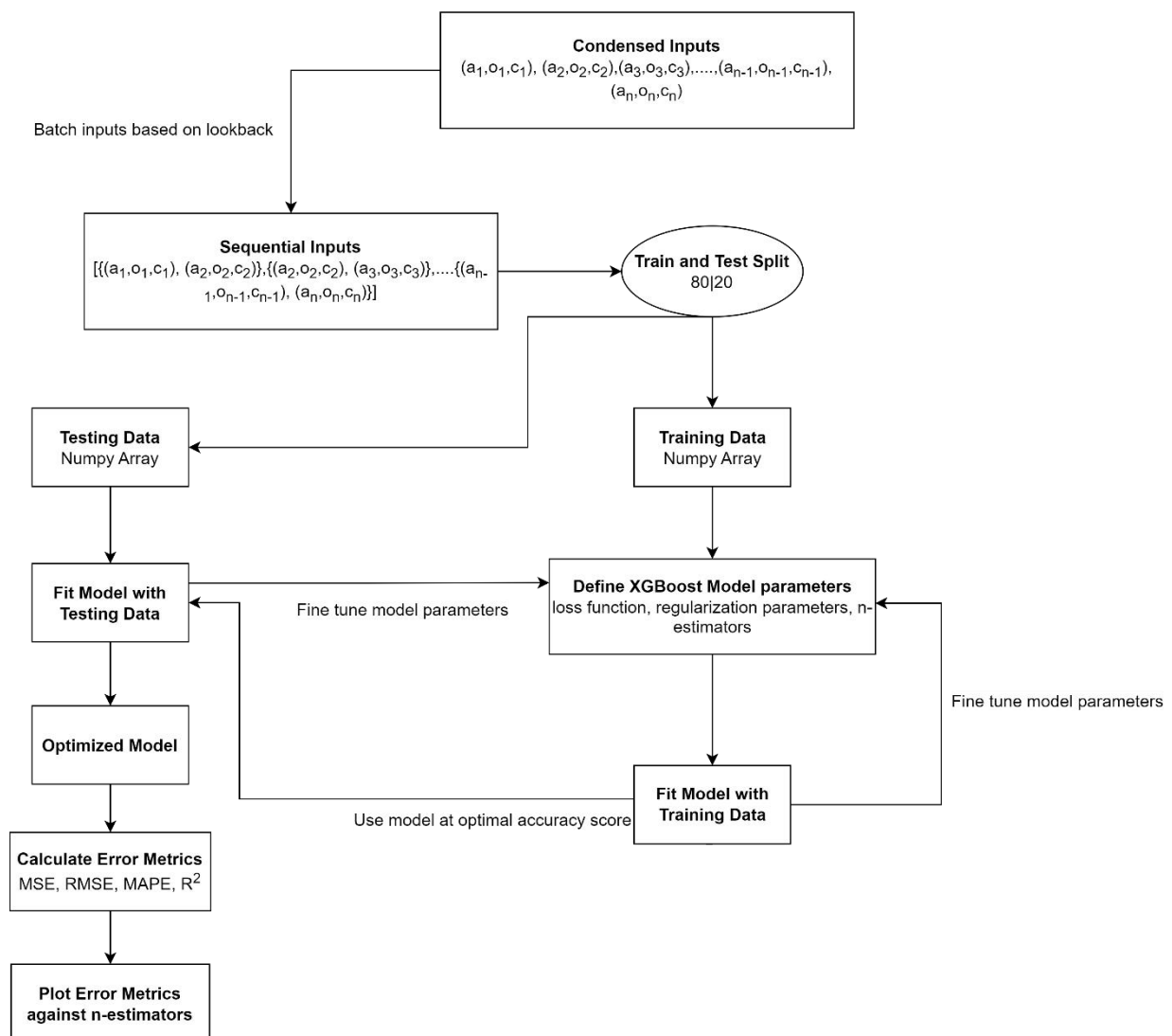


Figure 4.18 XGBoost Model Working

To implement the XGBoost model the scikit-learn module is used. Since XGBoost is a powerful version of a supervised regression model, it is necessary to define the loss function it uses in order to train. Using the objective function that contains loss function and a regularization term,

we define the basics of the model. The objective function of the XGBoost model is defined as shown in formula (6).

$$obj(\theta) = L(\theta) + \Omega(\theta) \quad (6)$$

where L is the training loss function, and Ω is the regularization term.

A common choice of the loss function is the mean squared error, which is given by formula (7).

$$L(\theta) = \sum_i (y_i - \hat{y}_i)^2 \quad (7)$$

Where y is actual value and \hat{y} is the predicted value.

XGBoost is one of the ensemble learning techniques, which entails training and integrating various independent models (sometimes referred to as base learners) to produce a single prediction. In order for bad predictions to cancel out and better ones to add up to final positive predictions, XGBoost anticipates having base learners in a manner and tries to ensure these bad learners are not given importance. By default, this model will be using decision trees as the base learners. The formula to find the prediction scores of each individual tree are summed up to get the final score as represented in formula (8).

$$\hat{y}_i = \sum_{k=1}^K f_k(x_i), f_k \in \mathcal{F} \quad (8)$$

where K is the number of trees, f_k is a function in the functional space \mathcal{F} , and \mathcal{F} is the set of all possible trees.

For this model, using the objective keyword it is specified that the model is performing regression and using the squared error as the loss function in order to train the model. The loss function combined with regularization methods make the model effective in training. The regularization used is the default values: alpha is 1, lambda is 1 and gamma is 0. A good regularization prevents overfitting.

Moreover, the estimators are set to be 25. This value indicates that 25 trees will be used in the ensemble for this particular model, meaning there are 25 boosting rounds. These parameters together help define the overall structure for the XGBoost model implemented as depicted in the code snippet in Fig 4.16.

```
model = XGBRegressor(objective='reg:squarederror', n_estimators=25)
model.fit(X_train, y_train.ravel())
```

Figure 4.16 XGBoost Model Definition

Before fitting the data into the model, the dataframe is passed into the window data function as shown in Fig 4.17 which ensures that the data for the past 3 days is being used to predict the output for one day ahead, i.e., the 4th day. To be more precise, this function takes in the features closing price, opening price and polarity to make an array structure that can be passed to the model for training.

```

def window_data(df, window, feature_col_number1, feature_col_number2, feature_col_number3, target_col_number):
    # Create empty lists "X_close", "X_polarity", "X_volume" and y
    X_polarity = []
    X_open = []
    X_close = []
    y = []
    for i in range(len(df) - window):

        # Get close, ts_polarity, tw_vol, and target in the loop
        ts_polarity = df.iloc[i:(i + window), feature_col_number1]
        tw_open = df.iloc[i:(i + window), feature_col_number2]
        close = df.iloc[i:(i + window), feature_col_number3]
        target = df.iloc[(i + window), target_col_number]

        # Append values in the lists
        X_polarity.append(ts_polarity)
        X_open.append(tw_open)
        X_close.append(close)
        y.append(target)

    return np.hstack((X_polarity,X_open,X_close)), np.array(y).reshape(-1, 1)

window_size = 3

feature_col_number1 = 0
feature_col_number2 = 1
feature_col_number3 = 2
target_col_number = 2
x, y = window_data(df2, window_size, feature_col_number1, feature_col_number2, feature_col_number3, target_col_number)

```

Figure 4.17 XGBoost Sequence Building

CHAPTER 5

TESTING

This chapter presents the results of testing and hyperparameter finetuning the models with regards to spam filtration and Ether price prediction

5.1 Hyperparameter and Feature Optimization

5.1.1 C5.0

The C5.0 model was tested with various user parameters and tweet parameters as conditions for rule formation and pruning of the decision tree.

Features Used	Accuracy
"Tweet.Likes", "Tweet.Retweet.Count", "Tweet.Reply.Count", "Tweet.Quote.Count"	0.0579
"Tweet.User Verified", "Tweet.Retweet.Count", "Tweet.Reply.Count", "Tweet.Quote.Count"	0.0631
"Tweet.User Verified", "Tweet.Reply.Count", "Tweet.Quote.Count"	0.0629
"Tweet.Retweet.Count", "Tweet.Reply.Count", "Tweet.Quote.Count"	0.0631
"User.Followers", "User.Following", "User.Verified"	0.0269
"User.Verified", "User.Followers", "User.Following", "Tweet.Likes", "Tweet.Reply.Count"	0.0163

Table 5.1 Testing feature selection for C5.0 Decision Tree

The C5.0 model's accuracy is calculated with regards to False Positive (Ham tweets being marked as Spam), since a major criterion is to not filter out Ham tweets which could be useful for prediction. From Table 5.1, it is observed that the model performs better when the tweet's parameters are considered for filtration rather than the user's parameters. Even though the accuracy, when user parameters are considered, are commendable, tweet parameters such as "Tweet Likes" and "Tweet Retweet Count" hold a major significance on the structure of the decision tree and accuracy of the model.

5.1.2 LSTM

The LSTM model was tested by finetuning the hyperparameters such as lookback, batch size, epochs, number of iterations, hidden dimension, and number of hidden layers.

Lookback	Batch Size	Epochs	Number of Iterations	Hidden Dimensions	Layers	Training RMSE	Testing RMSE
1	60	400	800	60	3	0.09098	0.11592
2	60	400	800	60	3	0.08474	0.11651
3	60	400	800	60	3	0.08997	0.13775
4	60	400	800	60	3	0.08491	0.12032
1	30	400	800	60	3	0.09302	0.11964
1	30	600	1000	60	3	0.09330	0.12051
1	30	400	1000	120	3	0.08802	0.11050
2	30	400	1000	120	3	0.08438	0.11233
2	50	400	1000	120	3	0.08473	0.11140
2	50	500	900	120	3	0.08376	0.11597
2	45	500	600	120	3	0.08377	0.11643
2	30	400	600	60	3	0.08853	0.13172
2	50	300	600	60	3	0.10269	0.14828

Table 5.2 Hyperparameter tuning for LSTM

The scores calculated are for the normalized values of the price changes of Ether. It is observed from Table 5.2, increase in lookback causes an increase in Root Mean Square Error (RMSE) scores. This phenomenon is due to the volatility of Ether's price. Having a larger lookback leads to confusing the model. A large batch size is optimal for training with datasets of low size. It is noticed that greater the number of iterations, better the training RMSE results. But this leads to lowers test results since the model overfits the training data at higher number of iterations. As the size of hidden dimension increases, the model is able to capture more features of the dataset, allowing better RMSE scores.

5.1.3 XGBoost

The XGBoost model was tested by finetuning the hyperparameters such as lookback and number of estimators.

Lookback	Number of estimators	Training RMSE	Testing RMSE
3	10	0.13357	0.22010
3	20	0.08328	0.19262
3	30	0.07450	0.18913
3	40	0.06952	0.18893
3	50	0.05978	0.18942
1	30	0.06982	0.14505
2	30	0.06047	0.16663
3	30	0.05763	0.16456
4	30	0.06061	0.18913
10	30	0.04230	0.17489
20	30	0.08376	0.22400

Table 5.3 Hyperparameter tuning for XGBoost

Similar to the observations with the LSTM model, it is observed from Table 5.3 that the general trend is that increase in lookback causes an increase in RMSE scores which is caused due to the volatility of Ether's price. On the other hand, increasing the `n_estimators` can cause overfitting to occur quite easily. Hence it is observed that, up to about 30 trees, the RMSE generally decreases and then it does not change much. Hence 30 boosting trees are used in our model.

CHAPTER 6

EXPERIMENTAL RESULTS

6.1 Metric Definitions

6.1.1 Root Mean Square Error:

The standard deviation of the residuals is defined as the Root Mean Square Error (RMSE) (prediction errors) as shown in formula (9). Residuals are a measure of how far away data points are from the regression line. RMSE is a measure of how to spread out the residuals. In other words, it indicates how concentrated the data is around the best-fit line. To validate experimental data, root mean square error is often used in forecasting and regression analysis.

$$RMSE = \sqrt{\frac{\sum_{i=1}^N (Predicted_i - Actual_i)^2}{N}} \quad (9)$$

where RMSE stands for Root Mean Square Error, $Predicted_i$ stands for predicted i^{th} value, $Actual_i$ stands for actual i^{th} value, and N stands for total number of samples.

6.1.2 Mean Squared Error:

The Mean Squared Error (MSE) is a metric that quantifies how near a regression line is to a set of data points. It represents the expected value of the squared error loss as a risk function as represented in formula (10). The average, especially the mean, of errors squared from data as it pertains to a function is used to determine mean square error.

$$MSE = \frac{1}{N} \sum_{i=1}^N (Predicted_i - Actual_i)^2 \quad (10)$$

where RMSE stands for Root Mean Square Error, $Predicted_i$ stands for predicted i^{th} value, $Actual_i$ stands for actual i^{th} value, and N stands for total number of samples.

6.1.3 R Squared Error

R squared is a statistical metric that measures a regression model's quality of fit and is calculated using formula (11). The optimal R square value is 1. The closer the R^2 number is to one, the better the model fits. R square is a ratio that compares the residual sum of squares to the total sum of squares.

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} \quad (11)$$

where R^2 stands for R squared error, SS_{res} stands for residual sum of squares, and SS_{tot} stands for total sum of squares.

The total sum of squares is computed by adding the squares of the perpendicular distances between data points and the average line as depicted in formula (12).

$$SS_{tot} = \sum (Predicted_i - Actual_{avg})^2 \quad (12)$$

where SS_{tot} stands for total sum of squares, $Predicted_i$ stands for predicted i^{th} value, and $Actual_{avg}$ stands for average of actual values.

The residual sum of squares is obtained by adding the squares of the perpendicular distance between the data points and the best-fitted line as portrayed in formula (13).

$$SS_{res} = \sum (Predicted_i - Actual_i)^2 \quad (13)$$

where SS_{res} stands for residual sum of squares, $Predicted_i$ stands for predicted i^{th} value, and $Actual_i$ stands for actual i^{th} value.

6.1.4 Mean Absolute Percentage Error

The mean absolute percentage error (MAPE) is the average or mean of anticipated absolute percentage errors and is calculated as shown in formula (14). Error is defined as the difference between the actual or observed value and the projected value. MAPE is computed by adding percentage mistakes without respect to sign. This metric is easy to grasp because it displays the inaccuracy in terms of percentages. Furthermore, when absolute percentage mistakes are employed, the problem of positive and negative errors cancelling out is avoided.

$$MAPE = \frac{1}{N} \sum_{t=1}^N \left| \frac{Actual_t - Predicted_t}{Actual_t} \right| \quad (14)$$

Where MAPE stands for mean absolute percentage error, N stands for the number of iterations, $Predicted_t$ stands for predicted t^{th} value, and $Actual_t$ stands for actual t^{th} value.

6.2 Results

6.2.1 Root Mean Square Error

In order to determine how many epochs and estimators are ideal for training the model, Fig 6.1 and Fig 6.2 is plotted. It can be noticed that the RMSE value is generally decreasing as the number of epochs increases. However, at a certain point it starts to converge to a particular value as indicated by the line flattening. Beyond the 300-epoch limit, the train and test RMSE scores of the LSTM model begin to converge indicating it to be an ideal number of epochs. Similarly, for the XGBoost model, the number of estimators were chosen as the parameter and convergence begins at 25 for the testing data and at 150 for the training data. This difference is due to the fact that size of the training data is larger, and the model learns its features better as the number of estimators increases. The LSTM model shows a better RMSE score at convergence but requires greater epochs when compared to the XGBoost model.

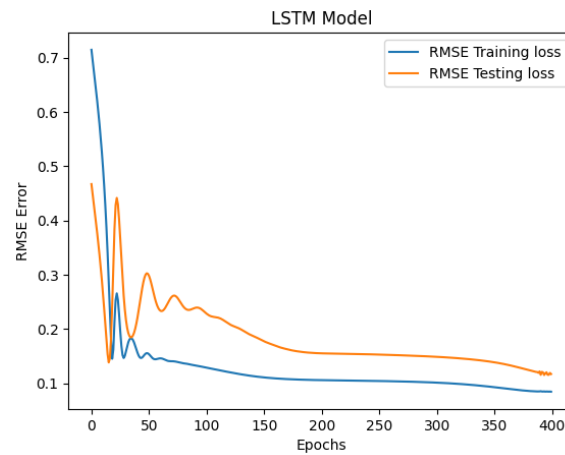


Figure 6.1 LSTM Model RMSE

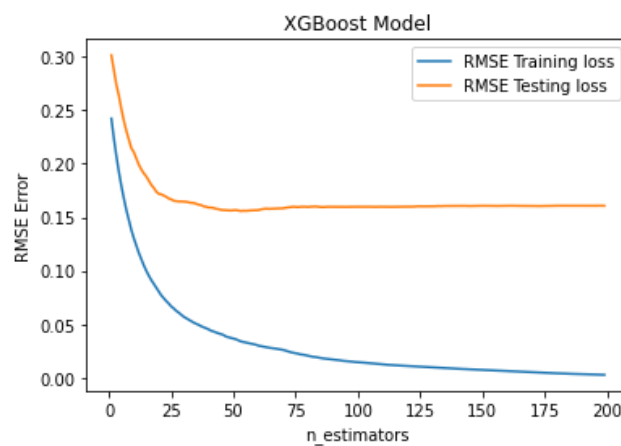


Figure 6.2 XGBoost Model RMSE

6.2.2 Mean Square Error

The mean squared error is used as another metric to compare the LSTM model and the XGBoost model. From Fig 6.3 and Fig 6.4, it is observed that the LSTM model shows high variations for the testing data at the beginning of the graph. It is because of the low number of epochs, the LSTM model is not able to capture the pattern of the graph. Even though the error might dip low suddenly for a random set of weights, the model overall will not be able to learn the trend of the graph in very low epochs. The LSTM model converges at 200 epochs from which the change in error is miniscule. The XGBoost model shows a better convergence in MSE scores when compared with the RMSE scores since the training and testing data begin convergence together. The XGBoost model converges at 25 epochs but is still slightly higher than the LSTM model.

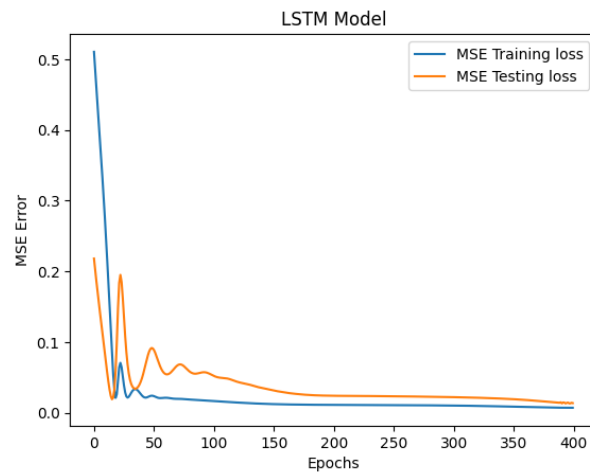


Figure 6.3 LSTM Model MSE

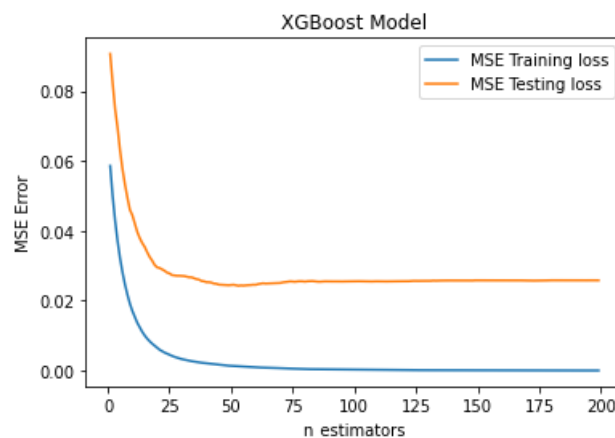


Figure 6.4 XGBoost Model MSE

6.2.3 R Squared Error

The R squared error for LSTM varies significantly when comparing training and testing data as observed in Fig 6.5 and Fig 6.6. The R squared value for testing starts to fluctuate often in the beginning epochs while in training, it is not as prevalent. This issue might have risen due to the testing data being relatively smaller, causing the model to not evaluate as accurately and hence requiring more epochs for a steady R squared value. However, in both graphs, for training and testing it is evident that at approximately 250 the fluctuation becomes more stable. Hence the epochs chosen is valid for the LSTM model. Similarly, it is inferred from Fig 6.7, in the XGBoost model, the values for both testing and training follow a similar trend and testing gives lower R-squared values overall. This can indicate that the model is well fitted. Here the `n_estimators` start to stabilise at around 25 which justifies the choice made in implementing the XGBoost model.

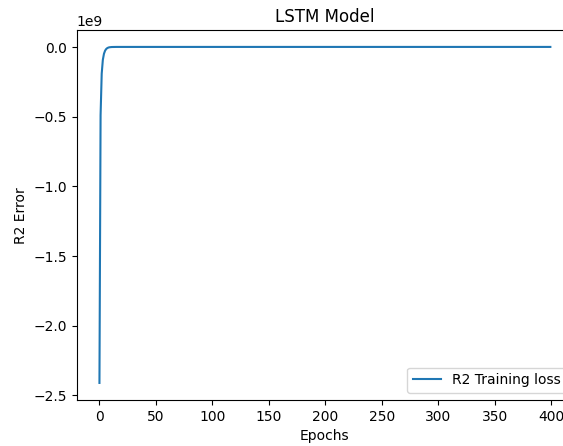


Figure 6.5 LSTM Model Training R Squared Error

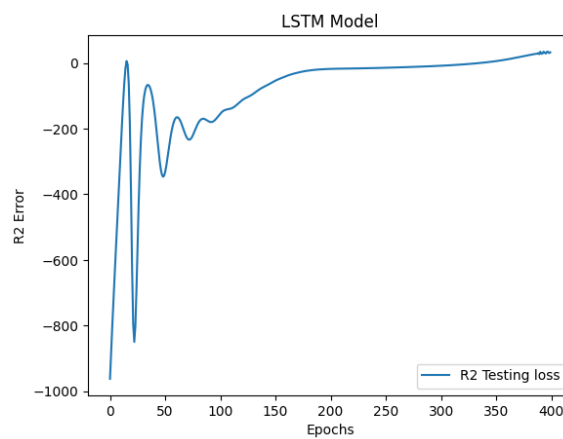


Figure 6.6 LSTM Model Testing R Squared Error

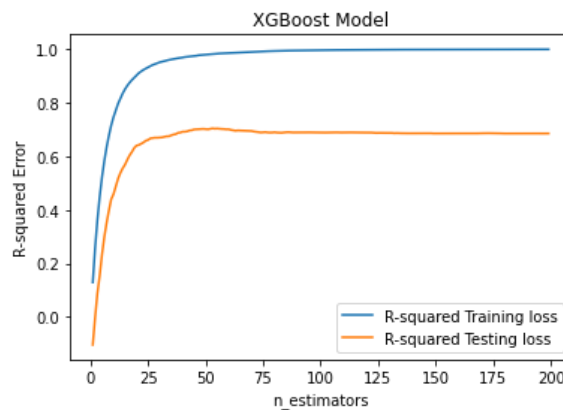


Figure 6.7 XGBoost Model R Squared Error

6.2.4 Mean Absolute Percentage Error

The MAPE fluctuates a lot in the starting epochs for LSTM as observed in Fig 6.8. This is due to the fact that Ether prices fluctuate a lot. Hence in the starting where the model has not trained as much, the predictions can be more erratic. It can be observed that at 200 epochs the fluctuations reduce and it shows the model has been fitted well. In XGBoost, it is deduced from Fig 6.9 that the MAPE value for training is way lesser than testing which can once again attribute to the fact that the size of testing data was smaller compared to training. LSTM has an overall smaller MAPE value but it would require more epochs.

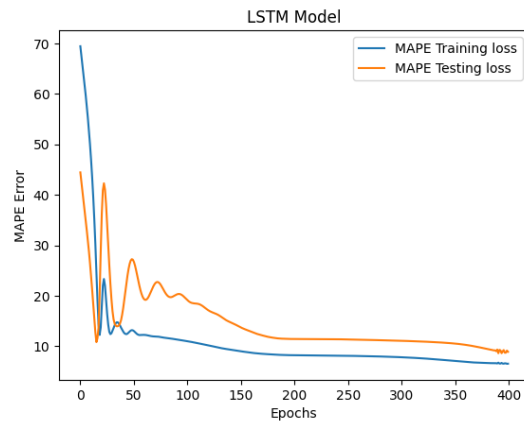


Figure 6.8 LSTM Model MAPE

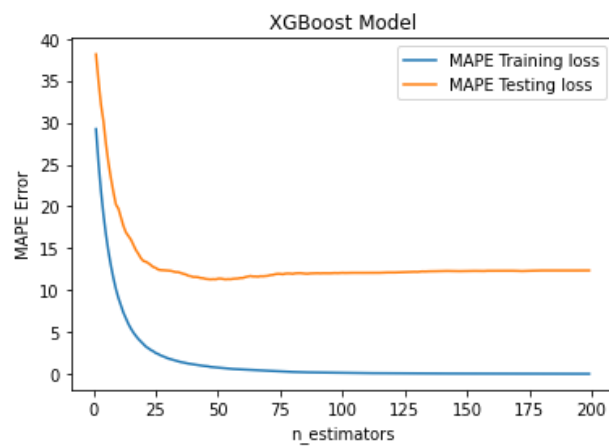


Figure 6.9 XGBoost Model MAPE

6.2.5 Comparison of LSTM and XGBoost

The figures 6.10 and 6.11 show how the real values of Ether differ from the predicted prices. The normalized values have been plotted for both training and testing data.

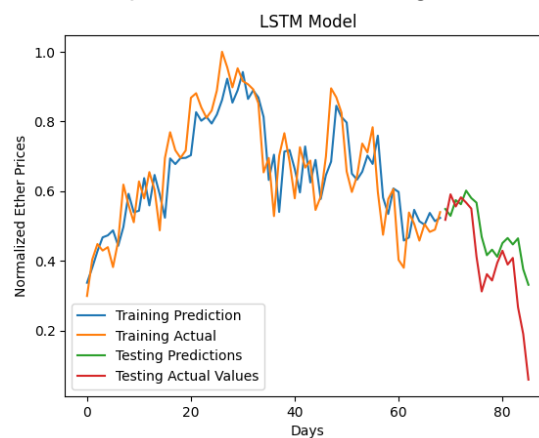


Figure 6.10 LSTM Model Performance Metrics

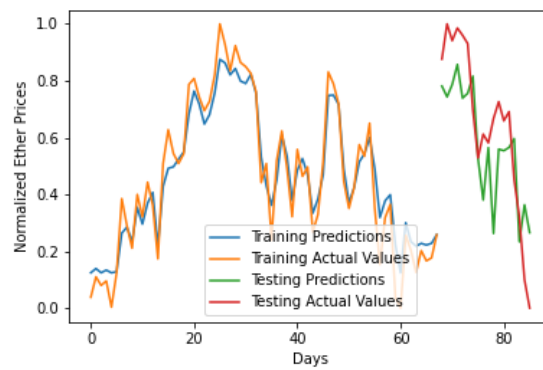


Figure 6.11 XGBoost Model Performance Metrics

Figure 6.10 is the graph depicting the resulting values for the LSTM model. It can be noticed that the predicted prices line closely follows that of real prices. Though not very exact, in comparison it is able to predict the overall trend of whether the prices increase or decrease decently. On the other hand, the XGBoost has the real prices and predicted prices lines almost overlapping for the training dataset; but the prediction for testing is not as great. This implies to a certain extent the model might be slightly overfitted during the training. However even then the testing parameters gave good values that shows the predictions were good. Table 6.1 shows the performance metrics for both the models on the test data.

Metric	LSTM	XGBoost
Root Mean Square Error	0.1144	0.1646
Mean Squared Error	0.0137	0.0270
R-squared Error	36.1707	0.6702
Mean Absolute Percentage Error	8.5883	12.321

Table 6.1 Performance Metrics for LSTM and XGBoost Model

In terms of RMSE the general consensus is that the lower the value, the better the model is predicting. Hence in comparison, LSTM has a better performance in terms of RMSE. Since RMSE is the measure of how well a regression line fits the data points, implying that LSTM can predict a line of best fit that fits the data points better than XGBoost.

The MSE value works on a similar principle to that of rms since it is just the squared value of RMSE. So again, we see that LSTM will have a better performance than XGBoost.

However, unlike RMSE and MSE, it is ideal to have a higher r-squared error. When the R-squared value is greater, the regression model is considered to be a good model since it accounts for a majority of the variance in the values of the response variable. A combination of both R-squared and RMSE is to be used for evaluating the models better. Thus, from Table 6.1, it can be observed that LSTM again has a better score than XGBoost.

MAPE basically tells the average percentage difference between the predicted and actual prices hence this should also ideally be low. In comparison, the XGBoost model has a higher value than for LSTM. However, for our use case it is more important to consider the values for RMSE, MSE and R-squared as in this use case it is more important to see if the general trend can be predicted rather than the actual prices. Hence, the other parameters are a better indicator of whether the line follows the trend or not.

From these values it can be concluded that the deep learning model, LSTM, has a better performance when predicting Ether prices than XGBoost. Although both models have a good performance, LSTM model is more accurate.

CHAPTER 7

CONCLUSION AND FUTURE ENHANCEMENTS

Non-conventional forms of asset and money transactions continue to evolve and grow, testing the limits of human innovation. Cryptocurrency and its implementation have cemented the truth into place that observing and adopting such transaction forms is highly beneficial to those looking to venture into expanding their assets in new domains. Ether (ETH) is a form of digital currency that functions as the primary asset of the Ethereum blockchain platform. In recent times, Ether has gained popularity due to its changing technologies and business activities.

Currently, many people use various methods for price prediction of Ether. Economic studies, statistical evaluations, machine learning models, and deep learning models are few of the major techniques used for price prediction. One of the major factors considered by most techniques is the influence of social media.

The effect of social media on the activities of cryptocurrency buyers and sellers is undeniable. One such social media platform that affects its users through its constant stream of algorithmically generated timelines and topics is the popular microblogging site Twitter. Examining the influence of tweets, the posts made on Twitter, can help us gain an understanding of how the sentiments of legitimate users on the platform can affect the prices of cryptocurrency, like Ether.

A holistic pipelined model which derives the effects of the Twitter platform and perform Ether price prediction using two models, namely LSTM and XGBoost has been developed. A spam filtration algorithm with C5.0 has been implemented to reduce the effect of noise and improve accuracy. A comparison of both the models has been performed to identify which model would perform better under different circumstances. After finetuning of both the models and comparison based on different metrics, such as RMSE, MSE, MAPE, AND R2, the LSTM model performs marginally better than the XGBoost model when predicting the price of Ether.

As further enhancements, a larger dataset can be used to better capture the trend of the price variations of Ether. Further, real time data along with statistical methods can be combined for performing better prediction on different time scales. The current model is trained to remove spam data from tweets whereas the future work can include multiple different media forums having high correlation with Ether prices to better capture the Ether graph trend.

BIBLIOGRAPHY

- [1] Nakamoto, Satoshi. (2009). Bitcoin: A Peer-to-Peer Electronic Cash System. Cryptography Mailing list at <https://metzdowd.com>
- [2] Degradation state recognition of piston pump based on ICEEMDAN and XGBoost - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Flow-chart-of-XGBoost_fig3_345327934 [accessed 17 Sep, 2022]

[Flow chart of XGBoost. | Download Scientific Diagram \(researchgate.net\)](#)
- [3] Application of Long Short-Term Memory (LSTM) Neural Network for Flood Forecasting - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/The-structure-of-the-Long-Short-Term-Memory-LSTM-neural-network-Reproduced-from-Yan_fig8_334268507 [accessed 17 Sep, 2022]

[The structure of the Long Short-Term Memory \(LSTM\) neural network.... | Download Scientific Diagram \(researchgate.net\)](#)
- [4] P. Jay, V. Kalariya, P. Parmar, S. Tanwar, N. Kumar and M. Alazab, "Stochastic Neural Networks for Cryptocurrency Price Prediction," in IEEE Access, vol. 8, pp. 82804-82818, 2020, doi: 10.1109/ACCESS.2020.2990659.

[Stochastic Neural Networks for Cryptocurrency Price Prediction | IEEE Journals & Magazine | IEEE Xplore](#)
- [5] Mohil Maheshkumar Patel, Sudeep Tanwar, Rajesh Gupta, Neeraj Kumar, A Deep Learning-based Cryptocurrency Price Prediction Scheme for Financial Institutions, Journal of Information Security and Applications, Volume 55, 2020, 102583, ISSN 2214-2126, <https://doi.org/10.1016/j.jisa.2020.102583>.

[A Deep Learning-based Cryptocurrency Price Prediction Scheme for Financial Institutions - ScienceDirect](#)
- [6] Hamayel, Mohammad J., and Amani Yousef Owda. 2021. "A Novel Cryptocurrency Price Prediction Model Using GRU, LSTM and bi-LSTM Machine Learning Algorithms" AI 2, no. 4: 477-496. <https://doi.org/10.3390/ai2040030>

[AI | Free Full-Text | A Novel Cryptocurrency Price Prediction Model Using GRU, LSTM and bi-LSTM Machine Learning Algorithms \(mdpi.com\)](#)
- [7] Pour, E. S., Jafari, H., Lashgari, A., Rabiee, E., & Ahmadisharaf, A. (2022). Cryptocurrency Price Prediction with Neural Networks of LSTM and Bayesian Optimization. European Journal of Business and Management Research, 7(2), 20–27. <https://doi.org/10.24018/ejbmr.2022.7.2.1307>

[Cryptocurrency Price Prediction with Neural Networks of LSTM and Bayesian Optimization | European Journal of Business and Management Research \(ejbmr.org\)](#)
- [8] Andi, Hari Krishnan. "An accurate bitcoin price prediction using logistic regression with LSTM machine learning model." Journal of Soft Computing Paradigm 3, no. 3 (2021): 205-217.

- [9] Y. Indulkar, "Time Series Analysis of Cryptocurrencies Using Deep Learning & Fbprophet," 2021 International Conference on Emerging Smart Computing and Informatics (ESCI), 2021, pp. 306-311, doi: 10.1109/ESCI50559.2021.9397004.

[Time Series Analysis of Cryptocurrencies Using Deep Learning & Fbprophet | IEEE Conference Publication | IEEE Xplore](#)

- [10] M. Shin, D. Mohaisen and J. Kim, "Bitcoin Price Forecasting via Ensemble-based LSTM Deep Learning Networks," 2021 International Conference on Information Networking (ICOIN), 2021, pp. 603-608, doi: 10.1109/ICOIN50884.2021.9333853.

[Bitcoin Price Forecasting via Ensemble-based LSTM Deep Learning Networks | IEEE Conference Publication | IEEE Xplore](#)

- [11] S. Tanwar, N. P. Patel, S. N. Patel, J. R. Patel, G. Sharma and I. E. Davidson, "Deep Learning-Based Cryptocurrency Price Prediction Scheme With Inter-Dependent Relations," in IEEE Access, vol. 9, pp. 138633-138646, 2021, doi: 10.1109/ACCESS.2021.3117848.

[Deep Learning-Based Cryptocurrency Price Prediction Scheme With Inter-Dependent Relations | IEEE Journals & Magazine | IEEE Xplore](#)

- [12] Li, Panpan, Shengbo Gong, Shaocong Xu, Jiajun Zhou, Yu Shanqing, and Qi Xuan. "Cross Cryptocurrency Relationship Mining for Bitcoin Price Prediction." arXiv preprint arXiv:2205.00974 (2022).

[\[2205.00974\] Cross Cryptocurrency Relationship Mining for Bitcoin Price Prediction \(arxiv.org\)](#)

- [13] Abraham, Jethin, Danny W. Higdon, Johnny Nelson and Juan Ibarra. "Cryptocurrency Price Prediction Using Tweet Volumes and Sentiment Analysis." (2018).

[Cryptocurrency Price Prediction Using Tweet Volumes and Sentiment Analysis \(smu.edu\)](#)

- [14] Ye, Zi, Yinxu Wu, Hui Chen, Yi Pan, and Qingshan Jiang. 2022. "A Stacking Ensemble Deep Learning Model for Bitcoin Price Prediction Using Twitter Comments on Bitcoin" Mathematics 10, no. 8: 1307. <https://doi.org/10.3390/math10081307>

[Mathematics | Free Full-Text | A Stacking Ensemble Deep Learning Model for Bitcoin Price Prediction Using Twitter Comments on Bitcoin | HTML \(mdpi.com\)](#)

- [15] Pano, Toni, and Rasha Kashef. 2020. "A Complete VADER-Based Sentiment Analysis of Bitcoin (BTC) Tweets during the Era of COVID-19" Big Data and Cognitive Computing 4, no. 4: 33. <https://doi.org/10.3390/bdcc4040033>

[BDCC | Free Full-Text | A Complete VADER-Based Sentiment Analysis of Bitcoin \(BTC\) Tweets during the Era of COVID-19 | HTML \(mdpi.com\)](#)

- [16] Abayomi-Alli A, Abayomi-Alli O, Misra S, Fernandez-Sanz L. Study of the Yahoo-Yahoo Hash-Tag Tweets Using Sentiment Analysis and Opinion Mining Algorithms. Information. 2022; 13(3):152. <https://doi.org/10.3390/info13030152>

[Information | Free Full-Text | Study of the Yahoo-Yahoo Hash-Tag Tweets Using Sentiment Analysis and Opinion Mining Algorithms | HTML \(mdpi.com\)](#)

- [17] Marouane Birjali, Mohammed Kasri, Abderrahim Beni-Hssane, A comprehensive survey on sentiment analysis: Approaches, challenges and trends, Knowledge-Based Systems, Volume 226, 2021, 107134, ISSN 0950-7051, <https://doi.org/10.1016/j.knosys.2021.107134>.
[A comprehensive survey on sentiment analysis: Approaches, challenges and trends - ScienceDirect](#)
- [18] Tida, Vijay Srinivas, and Sonya Hsu. "Universal Spam Detection using Transfer Learning of BERT Model." arXiv preprint arXiv:2202.03480 (2022).
[\[2202.03480\] Universal Spam Detection using Transfer Learning of BERT Model \(arxiv.org\)](#)
- [19] Cresci, Stefano & Lillo, Fabrizio & Regoli, Daniele & Tardelli, Serena & Tesconi, Maurizio. (2018). Cashtag Piggybacking: Uncovering Spam and Bot Activity in Stock Microblogs on Twitter. ACM Transactions on the Web. 13. 10.1145/3313184.
[\(PDF\) Cashtag Piggybacking: Uncovering Spam and Bot Activity in Stock Microblogs on Twitter \(researchgate.net\)](#)
- [20] Rodrigues, Anisha & Fernandes, Roshan & A, Aakash & B, Abhishek & Shetty, Adarsh & K, Atul & Lakshman, Kuruva & Rajasaheb, Mahammad Shafi. (2022). Real-Time Twitter Spam Detection and Sentiment Analysis using Machine Learning and Deep Learning Techniques. Computational Intelligence and Neuroscience. 2022. 1-14. 10.1155/2022/5211949.
[\(PDF\) Real-Time Twitter Spam Detection and Sentiment Analysis using Machine Learning and Deep Learning Techniques \(researchgate.net\)](#)
- [21] Sun, Nan & Lin, Guanjun & Qiu, Junyang & Rimba, Paul. (2020). Near real-time twitter spam detection with machine learning techniques. International Journal of Computers and Applications. 1-11. 10.1080/1206212X.2020.1751387.
[\(PDF\) Near real-time twitter spam detection with machine learning techniques \(researchgate.net\)](#)
- [22] P. Garg and N. Girdhar, "A Systematic Review on Spam Filtering Techniques based on Natural Language Processing Framework," 2021 11th International Conference on Cloud Computing, Data Science & Engineering (Confluence), 2021, pp. 30-35, doi: 10.1109/Confluence51648.2021.9377042.
[A Systematic Review on Spam Filtering Techniques based on Natural Language Processing Framework | IEEE Conference Publication | IEEE Xplore](#)
- [23] Arif, Muhammad & Li, Jianxin & Iqbal, Muhammad & Liu, Kaixu. (2018). Sentiment analysis and spam detection in short informal text using learning classifier systems. Soft Computing. 22. 10.1007/s00500-017-2729-x.
[\(PDF\) Sentiment analysis and spam detection in short informal text using learning classifier systems \(researchgate.net\)](#)
- [24] Snscape Library

- <https://github.com/JustAnotherArchivist/snsrape>
- [25] Ethereum Data, Arpit Verma, Kaggle :
<https://www.kaggle.com/datasets/varpit94/Ethereum-data>
- [26] Yang, KC., Ferrara, E. & Menczer, F. Botometer 101: social bot practicum for computational social scientists. *J Comput Soc Sc* (2022). <https://doi.org/10.1007/s42001-022-00177-5>
<https://link.springer.com/article/10.1007/s42001-022-00177-5#citeas>
- [27] Package 'C50'
<https://cran.r-project.org/web/packages/C50/C50.pdf>
- [28] LSTM layer
https://keras.io/api/layers/recurrent_layers/lstm/&sa=D&source=docs&ust=1668578517206723&usq=AOvVaw1ebaJBhztFOvI2SSuE8Vlo
- [29] Stacked Long Short-Term Memory Networks
<https://machinelearningmastery.com/stacked-long-short-term-memory-networks/>
- [30] A Complete Understanding of Dense Layers in Neural Networks
<https://analyticsindiamag.com/a-complete-understanding-of-dense-layers-in-neural-networks/#:~:text=Like%20we%20use%20LSTM%20layers%20mostly%20in%20the,in%20the%20final%20stages%20of%20the%20neural%20network>
- [31] XGBoost Parameters
<https://xgboost.readthedocs.io/en/stable/parameter.html>

APPENDICES

```

import snsrape.modules as snmodules
import pandas as pd

columns = ["Username", "User Id", "User Verified", "User Followers", "User Following",
"Tweet Id", "Tweet Timestamp", "Tweet Text", "Tweet Links", "Tweet URL", "Tweet Likes",
"Tweet Retweet Count", "Tweet Reply Count", "Tweet Quote Count"]
df_rows = []

tweets_list = []
for i, tweet in enumerate(snmodules.twitter.TwitterSearchScraper('(Ethereum OR Ether)
lang:en since:2021-10-11 until:2021-10-21').get_items()):
    print(i)
    username = tweet.user.username
    user_id = tweet.user.id
    # user_profile = twitter_scraper.Profile('username').to_dict()
    user_verified = tweet.user.verified
    user_followers = tweet.user.followersCount
    user_friends = tweet.user.friendsCount
    tweetlinks = tweet.links[0] if tweet.links !=None else None
    tweet_links = tweetlinks.url if tweetlinks !=None else None # links in the tweet
    tweet_id = tweet.id
    tweet_timestamp = tweet.date
    tweet_text = tweet.renderedContent
    tweet_likes = tweet.likeCount
    tweet_replies = tweet.replyCount
    tweet_url = tweet.url # link of the tweet

    tweet_rts = tweet.retweetCount
    tweet_quotes = tweet.quoteCount

    tweet_row = [username, user_id, user_verified, user_followers, user_friends, tweet_id,
tweet_timestamp, tweet_text, tweet_links ,tweet_url , tweet_likes, tweet_rts,
tweet_replies, tweet_quotes]
    df_rows.append(tweet_row)

tweets_df = pd.DataFrame(df_rows, columns=columns)

tweets_df.to_csv('C:/Ether Price
Prediction/1.1/venv/TweetsCollectedFromTwitterScraper1.csv')

import snsrape.modules as snmodules
import pandas as pd

columns = ["Username", "User Id", "User Verified", "User Followers", "User Following",
"Tweet Id", "Tweet Timestamp", "Tweet Text", "Tweet Links", "Tweet URL", "Tweet Likes",
"Tweet Retweet Count", "Tweet Reply Count", "Tweet Quote Count"]
df_rows = []

tweets_list = []
for i, tweet in enumerate(snmodules.twitter.TwitterSearchScraper('(Ethereum OR Ether)
lang:en since:2021-10-11 until:2021-10-21').get_items()):
    print(i)
    username = tweet.user.username
    user_id = tweet.user.id
    # user_profile = twitter_scraper.Profile('username').to_dict()
    user_verified = tweet.user.verified
    user_followers = tweet.user.followersCount
    user_friends = tweet.user.friendsCount
    tweetlinks = tweet.links[0] if tweet.links !=None else None
    tweet_links = tweetlinks.url if tweetlinks !=None else None # links in the tweet
    tweet_id = tweet.id
    tweet_timestamp = tweet.date

```

```

tweet_text = tweet.renderedContent
tweet_likes = tweet.likeCount
tweet_replies = tweet.replyCount
tweet_url = tweet.url # link of the tweet

tweet_rts = tweet.retweetCount
tweet_quotes = tweet.quoteCount

tweet_row = [username, user_id, user_verified, user_followers, user_friends, tweet_id,
tweet_timestamp, tweet_text, tweet_links ,tweet_url , tweet_likes, tweet_rts,
tweet_replies, tweet_quotes]
df_rows.append(tweet_row)

tweets_df = pd.DataFrame(df_rows, columns=columns)

tweets_df.to_csv('C:/Ether Price
Prediction/1.1/venv/TweetsCollectedFromTwitterScrapers1.csv')

import threading
import time
import pandas as pd
import requests
import datetime
import json

urls = pd.read_csv("E:\Sem 7\pr1\TweepyTest\Book1.csv")
# urls = urls.drop(columns=urls.columns[0],
# axis=1)
print(urls.head)
urlslst = [list(row)[0] for row in urls.values]
# print(urlslst)

post_url =
'https://safebrowsing.googleapis.com/v4/threatMatches:find?key=AIzaSyCp3eif2n0TlGTuMxEax_U7
G1lDaeYD9yg'
post_headers = {
    'Content-Type' : 'application/json'
}

threatEntries = []
n = len(urlslst)
rem = n % 500
q = n - rem + 1
responses = []
for i in range(0, q, 500):

    threatEntries = []
    j=i
    while j+500 < n and j<i+500:
        threatEntries.append({"url": urlslst[j]})
        j+=1

    print(len(threatEntries), threatEntries[0])
    post_body = {
        "client": {
            "clientId": "SEM7projectwork",
            "clientVersion": "0.0.1"
        },
        "threatInfo": {
            "threatTypes": ["MALWARE", "SOCIAL_ENGINEERING", "UNWANTED_SOFTWARE",
"THREAT_TYPE_UNSPECIFIED", "POTENTIALLY_HARMFUL_APPLICATION"],
            "platformTypes": ["WINDOWS"],
            "threatEntryTypes": ["URL"],
            "threatEntries": threatEntries
        }
    }

```

```

    response = requests.post(
        url=post_url,
        headers=post_headers,
        json=post_body
    )
    print(response.text)
    filename = "E:\Sem 7\pr1\TweepyTest\safebrowsing
scripts\outputs\Outputlinksfrom0to79298" + str(i) + ".txt"
    with open(filename, "w") as text_file:
        text_file.write(str(responses))

with open("E:\Sem 7\pr1\TweepyTest\safebrowsing scripts\Output.txt", "w") as text_file:
    text_file.write(str(responses))

df = pd.DataFrame(responses)
df.to_csv('E:\Sem 7\pr1\TweepyTest\safebrowsing scripts\safebrowsingresponses1.csv')

usernames <- read.csv("Unique Usernames Spam with index.csv", header = TRUE, sep =",")
#usernames

usernamelist <- list(usernames$username)

library(tweetbotornot2)
library(rtweet)
packageVersion('rtweet')

screen_names <- usernamelist[[1]][1:1000]

screen_names
p1 <- predict_bot(screen_names)
fwrite(p1, "1 to 1000 Unique Usernames Spam.csv")
library(data.table)

usernames <- read.csv("uniqueusername.csv", header = TRUE, sep =",")
usernames

usernamelist <- list(usernames$X0)
usernamelist
length(usernamelist)

usernamelist[[1]][1:50]

chunk_length <- 500

splituname <- split(usernamelist, ceiling(seq_along(usernamelist) / chunk_length))
length(splituname)

splituname$nof_values <- lengths(splituname$summary)
splituname$nof_values
length(splituname$summary)

lapply(splituname, '[', 1)

library(tweetbotornot2)
library(rtweet)

library(data.table)

packageVersion('rtweet')

```

```

screen_names <- usernamelist[[1]][1:10]
p2 <- predict_botometer(users = screen_names, key =
"67490ae44amsh2bddb3621208d42p1c61b9jsn266682f0500f", set_key = TRUE, parse = TRUE, verbose
= TRUE)
fwrite(p1, "botometerwithtweetbot.csv")

screen_names_df <- data.frame(screen_names = screen_names)
p1

# Install packages
"""

install.packages("C50")

""""# Load package""""

require(C50)

""""Load dataset""""

data <- read.csv("Spam_Training.csv")

head(data)

str(data)

""""# To find levels of categorical features""""

table(data$spamlabel)

""""# Randomization""""

set.seed(69420)
g <- runif(nrow(data))
datar <- data[order(g), ]
str(datar)

datar$spamlabel[datar$spam == 'TRUE'] <- 'SPAM'
datar$spamlabel[datar$spam == 'FALSE'] <- 'HAM'
datar$spamlabel <- as.factor(datar$spamlabel)
datar$User.Verified[datar$User.Verified == 'TRUE'] <- 1
datar$User.Verified[datar$User.Verified == 'FALSE'] <- 0

str(datar)

datar[1,c("User.Id", "User.Verified", "User.Followers", "User.Following", "Tweet.Text",
"Tweet.Links", "Tweet.Likes", "Tweet.Retweet.Count", "Tweet.Reply.Count",
"Tweet.Quote.Count")]

""""# Implementation of model""""

m2 <- C5.0(datar[1:60000, c( "User.Verified", "User.Followers", "User.Following",
"Tweet.Likes", "Tweet.Reply.Count")], datar[1:60000, 15])
m2

summary(m2)

C5imp(m2, metric="splits")

plot(m2)

p1 <- predict(m2,datar[60001:80998,])

```



```

plot(p1)

p1 <- predict(m2, datar[60001:80998,])

""""# Prediction on full dataset""""

finaldf <- read.csv("final_unlabeled.csv")
finaldf

str(finaldf)

finaldf$classlabel <- NA
p2 <- predict(m2, finaldf)

finaldf$classlabel <- p2

str(finaldf)

head(finaldf)

library(data.table)

fwrite(finaldf, "labelled_final.csv")

labelled <- read.csv("labelled_final.csv")
str(labelled)

plot(p2)

install.packages("rio")

library(data.table)
df <- read.csv("final_unlabeled.csv", sep=",", stringsAsFactors=FALSE, fill=TRUE, header=
TRUE)
nrow(df)

df[1,]

df$class <- NA
p <- predict(m2, df)
df$class <- p

df = subset(df, select = -c(classlabel) )

summary(df)

fwrite(df, "final_labeled.csv")

df2 <- read.csv("final_labeled.csv", sep=",", stringsAsFactors=FALSE, fill=TRUE, header=
TRUE)
nrow(df2)

!pip install hvplot
import numpy as np
import pandas as pd
import hvplot.pandas
from xgboost import XGBRegressor
# %matplotlib inline
from sklearn import metrics
from sklearn import preprocessing
from google.colab import files
import matplotlib.pyplot as plt

from google.colab import drive
drive.mount('/content/drive')

```

```

df = pd.read_csv('/content/drive/MyDrive/ether price prediction/final_preprocessed.csv')

df = df.drop(columns={'Unnamed: 0', 'Username', 'User Id', 'Tweet Text', 'Tweet Links',
'apiresult', 'spamlabel', 'Tweet URL', 'Tweet Id'})

label_encoder = preprocessing.LabelEncoder()
df['User Verified'] = label_encoder.fit_transform(df['User Verified'])
df

df['Tweet Date'] = pd.to_datetime(df['Tweet Date'], dayfirst=True, format='%d-%m-%y',
infer_datetime_format=True)
df = df.groupby('Tweet Date').mean()
df

convert_dict = {
    'User Verified': float,
    'User Followers': float,
    'User Following': float,
    'Tweet Likes': float,
    'Tweet Retweet Count': float,
    'Tweet Reply Count': float,
    'Open': float,
    'Close': float,
    'High': float,
    'Low': float,
    'Positive': float,
    'Neutral': float,
    'Negative': float,
}

df = df.astype(convert_dict)
print(df.dtypes)

#df.loc[df["Neutral"] == 100.00, "Negative"] = 18
df["Polarity"] = df["Positive"] - df["Negative"]
print(df.dtypes)

df["Negative"] = df["Negative"].astype('float64')
df["Positive"] = df["Positive"].astype('float64')
df["User Verified"] = df["User Verified"].astype('int')
df["Aggregate"] = df["Polarity"] + (0.31*df["User Verified"] + 0.30*df["User Followers"] -
0.06*df["User Following"] + 0.5*df["Tweet Likes"] + 0.22*df["Tweet Retweet Count"] +
0.21*df["Tweet Reply Count"] + 0.05*df["Tweet Quote Count"])
df2=df[["Aggregate", "Open", "Close"]]
df2

def window_data(df, window, feature_col_number1, feature_col_number2, feature_col_number3,
target_col_number):
    # Create empty lists "X_close", "X_polarity", "X_volume" and y
    X_polarity = []
    X_open = []
    X_close = []
    y = []
    for i in range(len(df) - window):

        # Get close, ts_polarity, tw_vol, and target in the loop
        ts_polarity = df.iloc[i:(i + window), feature_col_number1]
        tw_open = df.iloc[i:(i + window), feature_col_number2]
        close = df.iloc[i:(i + window), feature_col_number3]
        target = df.iloc[(i + window), target_col_number]

        # Append values in the lists
        X_polarity.append(ts_polarity)
        X_open.append(tw_open)
        X_close.append(close)

```

```

        y.append(target)

    return np.hstack((X_polarity,X_open,X_close)), np.array(y).reshape(-1, 1)

window_size = 4

# Column index 0 is the `Adj Close` column
# Column index 1 is the `ts_polarity` column
# Column index 2 is the `twitter_volume` column
feature_col_number1 = 0
feature_col_number2 = 1
feature_col_number3 = 2
target_col_number = 2
X, y = window_data(df2, window_size, feature_col_number1, feature_col_number2,
feature_col_number3, target_col_number)

X_split = int(0.8 * len(X))
y_split = int(0.8 * len(y))

# Set X_train, X_test, y_train, t_test
X_train = X[: X_split]
X_test = X[X_split:]
y_train = y[: y_split]
y_test = y[y_split:]

from sklearn.preprocessing import MinMaxScaler

x_train_scaler = MinMaxScaler()
x_test_scaler = MinMaxScaler()
y_train_scaler = MinMaxScaler()
y_test_scaler = MinMaxScaler()

# Fit the scaler for the Training Data
x_train_scaler.fit(X_train)
y_train_scaler.fit(y_train)

# Scale the training data
X_train = x_train_scaler.transform(X_train)
y_train = y_train_scaler.transform(y_train)

# Fit the scaler for the Testing Data
x_test_scaler.fit(X_test)
y_test_scaler.fit(y_test)

# Scale the y_test data
X_test = x_test_scaler.transform(X_test)
y_test = y_test_scaler.transform(y_test)

rmlss=[]
mplss=[]
rlss=[]
mlss=[]
trmlss=[]
tmplss=[]
trlss=[]
tmlss=[]
eps=[]
for n in range (1,200):
    model = XGBRegressor(objective='reg:squarederror', n_estimators=n)
    model.fit(X_train, y_train.ravel())
    training_predicted = model.predict(X_train)
    predicted = model.predict(X_test)
    trmlss.append(np.sqrt(metrics.mean_squared_error(y_train, training_predicted)))
    rmlss.append(np.sqrt(metrics.mean_squared_error(y_test, predicted)))
    tmplss.append(metrics.mean_squared_log_error(y_train, training_predicted)*1000)
    mplss.append(metrics.mean_squared_log_error(y_test, predicted)*1000)

```

```

    trlss.append(metrics.r2_score(y_train, training_predicted))
    rlss.append(metrics.r2_score(y_test, predicted))
    tmlss.append(metrics.mean_squared_error(y_train, training_predicted))
    mlss.append(metrics.mean_squared_error(y_test, predicted))
    eps.append(n)

#plt.plot(mse_error_train, label="MSE Training loss")
#plt.plot(mse_error_test, label="MSE Testing loss")
plt.title("XGBoost Model")
plt.plot(eps, trmlss, label="RMSE Training loss" )
plt.plot(eps, rmlss, label="RMSE Testing loss" )
plt.legend()
plt.xlabel("n_estimators")
plt.ylabel("RMSE Error")
plt.savefig('xgboost_RMSE.png')
files.download('xgboost_RMSE.png')

plt.title("XGBoost Model")
plt.plot(eps, tmlss, label="MAPE Training loss" )
plt.plot(eps, mplss, label="MAPE Testing loss" )
#plt.plot(eps, mplss)
plt.legend()
plt.xlabel("n_estimators")
plt.ylabel("MAPE Error")
plt.savefig('xgboost_MAPE.png')
files.download('xgboost_MAPE.png')

plt.title("XGBoost Model")
plt.plot(eps, trlss, label="R-squared Training loss" )
plt.plot(eps, rlss, label="R-squared Testing loss" )
#plt.plot(eps, rlss)
plt.legend()
plt.xlabel("n_estimators")
plt.ylabel("R-squared Error")
plt.savefig('xgboost_r2.png')
files.download('xgboost_r2.png')

plt.title("XGBoost Model")
plt.plot(eps, tmlss, label="MSE Training loss" )
plt.plot(eps, mlss, label="MSE Testing loss" )
#plt.plot(eps, mlss)
plt.legend()
plt.xlabel("n_estimators")
plt.ylabel("MSE Error")
plt.show()
#plt.savefig('xgboost_MSE.png')
#files.download('xgboost_MSE.png')

model = XGBRegressor(objective='reg:squarederror', n_estimators=30)

model.fit(X_train, y_train.ravel())

training_predicted = model.predict(X_train)

predicted = model.predict(X_test)
predicted

print("MSE: ", metrics.mean_squared_error(y_test, predicted))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, predicted)))
print("MAPE: ", metrics.mean_squared_log_error(y_test, predicted)*1000)
print('R-squared :', metrics.r2_score(y_test, predicted))

# Recover the original prices instead of the scaled version
predicted_prices = y_test_scaler.inverse_transform(predicted.reshape(-1, 1))
real_prices = y_test_scaler.inverse_transform(y_test.reshape(-1, 1))

```

```

stocks = pd.DataFrame({
    "Real": real_prices.ravel(),
    "Predicted": predicted_prices.ravel()
}, index = df.index[-len(real_prices): ])
stocks.head()

import matplotlib.pyplot as plt

ls = []
ls2 = []
for i in range(0,68):
    ls.append(i)
for i in range(68,86):
    ls2.append(i)

len(y_test)

len(y_train)

len(predicted)

plt.plot(ls,training_predicted, label = "Training Predictions")
plt.plot(ls,y_train, label = "Training Actual Values")
plt.plot(ls2,predicted, label = "Testing Predictions")
plt.plot(ls2,y_test, label = "Testing Actual Values")
plt.legend()
plt.xlabel("Days")
plt.ylabel("Normalized Ether Prices")
plt.savefig('xgboost.png')
files.download('xgboost.png')

from xgboost import plot_tree

plot_tree(model)
plt.savefig('xgboost_dt.jpeg')
files.download('xgboost_dt.jpeg')

import pandas as pd
import numpy as np
from sklearn import preprocessing

from google.colab import drive
drive.mount('/content/drive')

df = pd.read_csv('/content/drive/MyDrive/ether price prediction/final_preprocessed.csv')

df = df.drop(columns={'Unnamed: 0', 'Username', 'User Id', 'Tweet Text', 'Tweet Links',
'apiresult', 'spamlabel', 'Tweet URL', 'Tweet Id'})
df

label_encoder = preprocessing.LabelEncoder()
# Encode labels in column 'species'.
df['User Verified']= label_encoder.fit_transform(df['User Verified'])
df

df['Tweet Date']=pd.to_datetime(df['Tweet Date'],dayfirst=True,format='%d-%m-%y',
infer_datetime_format=True)

convert_dict = {
    'User Verified': float,
    'User Followers':float,
    'User Following':float,
    'Tweet Likes':float,
    'Tweet Retweet Count':float,
    'Tweet Reply Count':float,
    'Open':float,

```

```

        'Close':float,
        'High':float,
        'Low':float,
        'Positive':float,
        'Neutral':float,
        'Negative':float,
    }

df = df.astype(convert_dict)
print(df.dtypes)

df = df.groupby('Tweet Date').mean()

df

print(df.corr()['Close'])

# print("Positive : ", df['Polarity'].value_counts()['Positive'])
# print("Negative : ", df['Polarity'].value_counts()['Negative'])
# print("Neutral : ", df['Polarity'].value_counts()['Neutral'])

df["Polarity"] = (df["Positive"]) - (df["Negative"])
print(df.dtypes)

df["Negative"] = df["Negative"].astype('float64')
df["Positive"] = df["Positive"].astype('float64')
df["User Verified"] = df["User Verified"].astype('int')
df["Aggregate"] = df["Polarity"] + (0.31*df["User Verified"] + 0.30*df["User Followers"] -
0.06*df["User Following"] + 0.5*df["Tweet Likes"] + 0.22*df["Tweet Retweet Count"] +
0.21*df["Tweet Reply Count"] + 0.05*df["Tweet Quote Count"])
# df["Aggregate"] = -0.09*df["Positive"] + 0.18*df["Neutral"] - 0.19*df["Negative"] +
(0.31*df["User Verified"] + 0.30*df["User Followers"] - 0.06*df["User Following"] +
0.5*df["Tweet Likes"] + 0.22*df["Tweet Retweet Count"] + 0.21*df["Tweet Reply Count"] +
0.05*df["Tweet Quote Count"])
df2=df[["Aggregate","Open", "Close"]]
df2

print(df2.corr()['Close'])

df3 = df2
df3

print(df3.corr()['Close'])

len(df3)

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
Close = pd.DataFrame(df3)
Close = Close.drop(columns={"Aggregate", "Open"})
# Open = df3.loc[:, 'Open']
# Polarity = df3.loc[:, 'Polarity']

c = scaler.fit(Close)
# o = scaler.fit(Open)
# p = scaler.fit(Polarity)
scaled_close = c.transform(Close)
# scaled_open = o.transform(Open)
# scaled_polarity = p.transform(Polarity)
print(scaled_close)

df4 = df3
for column in df4.columns:
    df4[column] = (df4[column] - df4[column].min()) / (df4[column].max() -
df4[column].min())

```

```

df4

df = df4[: -1]
df

temp = df4[['Close']]
temp = temp[1:]
temp

l = []
for i in range(len(temp)):
    l.append(temp.iloc[i,0])
df['Next_Day_Price'] = l
df

print(df.corr()['Next_Day_Price'])

"""## 1"""

lookback = 2
data_independent=[]
data_dependent=[]
for index in range(len(df)-lookback):
    temp1 = []
    temp1 = (df.iloc[index:index+lookback, :-1])
    data_independent.append(temp1)
    data_dependent.append(df.iloc[index:index+lookback, -1:])

data_independent=np.array(data_independent)
data_dependent=np.array(data_dependent)
test_set_size = int(np.round(0.2 * data_independent.shape[0]))
train_set_size = data_independent.shape[0] - (test_set_size)

x_train = data_independent[:train_set_size]
y_train = data_dependent[:train_set_size, -1]

x_test = data_independent [train_set_size:]
y_test = data_dependent [train_set_size:, -1]

print('x_train.shape = ',x_train.shape)
print('y_train.shape = ',y_train.shape)
print('x_test.shape = ',x_test.shape)
print('y_test.shape = ',y_test.shape)

x_train

y_train

import torch

x_train = torch.from_numpy(x_train).type(torch.Tensor)
x_test = torch.from_numpy(x_test).type(torch.Tensor)
y_train = torch.from_numpy(y_train).type(torch.Tensor)
y_test = torch.from_numpy(y_test).type(torch.Tensor)

x_train.shape

"""### 2"""

print(x_train.size(), y_train.size())

n_steps = lookback-1
batch_size = 50
num_epochs = 300
n_iters = 600

```

```

train_useful = torch.utils.data.TensorDataset(x_train,y_train)
test_useful = torch.utils.data.TensorDataset(x_test,y_test)

train_loader_useful = torch.utils.data.DataLoader(dataset=train_useful,
                                                  batch_size=batch_size,
                                                  shuffle=False)

test_loader_useful = torch.utils.data.DataLoader(dataset=test_useful,
                                                  batch_size=batch_size,
                                                  shuffle=False)

from torch import nn

"""### 3"""

input_dim = 3
hidden_dim = 60
num_layers = 2
output_dim = 1

class LSTM(nn.Module):
    def __init__(self, input_dim, hidden_dim, num_layers, output_dim):
        super(LSTM, self).__init__()
        # Hidden dimensions

        self.input_dim = input_dim
        self.hidden_dim = hidden_dim

        # Number of hidden layers
        self.num_layers = num_layers

        # Building your LSTM
        # batch_first=True causes input/output tensors to be of shape
        # (batch_dim, seq_dim, feature_dim)
        self.lstm = nn.LSTM(input_dim, hidden_dim, num_layers, batch_first=True)

        # Readout layer
        self.fc = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        # Initialize hidden state with zeros
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_dim).requires_grad_()

        # Initialize cell state
        c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_dim).requires_grad_()
        # One time step
        # We need to detach as we are doing truncated backpropagation through time (BPTT)
        # If we don't, we'll backprop all the way to the start even after going through
another batch
        out, (hn, cn) = self.lstm(x, (h0.detach(), c0.detach()))

        # Index hidden state of last time step
        # out.size() --> 100, 28, 100
        # out[:, -1, :] --> 100, 100 --> just want last time step hidden states!
        out = self.fc(out[:, -1, :])
        # out.size() --> 100, 10
        return out

model = LSTM(input_dim = input_dim, hidden_dim = hidden_dim, output_dim = output_dim,
             num_layers = num_layers)

loss_fn = torch.nn.MSELoss(reduction='mean')

```



```

optimiser = torch.optim.Adam(model.parameters(), lr=0.001)
print(model)
print(len(list(model.parameters())))
for i in range(len(list(model.parameters()))):
    print(list(model.parameters())[i].size())

import math
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error, r2_score

# Train model
#####

hist = np.zeros(num_epochs)

# Number of steps to unroll
seq_dim = lookback-1
mse_error_train = []
rmse_train = []
mape_train = []
r2_train = []
mse_error_test = []
rmse_test = []
mape_test = []
r2_test = []

for t in range(num_epochs):
    # Initialise hidden state
    # Forward pass
    y_train_pred = model(x_train)

    loss = loss_fn(y_train_pred, y_train)
    rmse_train.append(math.sqrt(mean_squared_error(y_train_pred.detach().numpy().reshape(-1, 1), y_train.detach().numpy().reshape(-1, 1))))
    mape_train.append(mean_absolute_error(y_train_pred.detach().numpy().reshape(-1, 1), y_train.detach().numpy().reshape(-1, 1))*100)
    r2_train.append(r2_score(y_train_pred.detach().numpy().reshape(-1, 1), y_train.detach().numpy().reshape(-1, 1))*100)
    mse_error_train.append(loss.item())

    x_test_pred = model(x_test)
    y_test_pred_ether = x_test_pred[:]

    mse_error_test.append(loss_fn(y_test_pred_ether, y_test).item())
    rmse_test.append(math.sqrt(mean_squared_error(y_test[:].detach().numpy().reshape(-1, 1), y_test_pred_ether.detach().numpy().reshape(-1, 1))))
    mape_test.append(mean_absolute_error((y_test[:].detach().numpy().reshape(-1, 1)), y_test_pred_ether.detach().numpy().reshape(-1, 1))*100)
    r2_test.append(r2_score((y_test[:].detach().numpy().reshape(-1, 1)), y_test_pred_ether.detach().numpy().reshape(-1, 1))*100)

    hist[t] = loss.item()

    # Zero out gradient, else they will accumulate between epochs
    optimiser.zero_grad()

    # Backward pass
    loss.backward()

    # Update parameters
    optimiser.step()

```

```

y_train_pred.size()

import matplotlib.pyplot as plt

from google.colab import files

plt.plot(mse_error_train, label="MSE Training loss")
plt.plot(mse_error_test, label="MSE Testing loss")
plt.legend()
plt.xlabel("Epochs")
plt.ylabel("MSE Error")
plt.show()
plt.savefig('lstm_MSE.png')
files.download('lstm_MSE.png')

plt.plot(mape_train, label="MAPE Training loss")
plt.plot(mape_test, label="MAPE Testing loss")
plt.legend()
plt.xlabel("Epochs")
plt.ylabel("MAPE Error")
plt.show()
plt.savefig('lstm_MAPE.png')
files.download('lstm_MAPE.png')

plt.plot(rmse_train, label="RMSE Training loss")
plt.plot(rmse_test, label="RMSE Testing loss")
plt.legend()
plt.xlabel("Epochs")
plt.ylabel("RMSE Error")
plt.show()
plt.savefig('lstm_RMSE.png')
files.download('lstm_RMSE.png')

plt.plot(r2_train, label="R2 Training loss")
plt.legend()
plt.xlabel("Epochs")
plt.ylabel("R2 Error")
plt.show()

plt.plot(r2_test, label="R2 Testing loss")
plt.legend()
plt.xlabel("Epochs")
plt.ylabel("R2 Error")
plt.show()

plt.plot(y_train_pred.detach().numpy()[0], label="Training Fit")
plt.plot(y_train.detach().numpy()[0])
plt.xlabel("Days")
plt.ylabel("Normalized Ether Price")
plt.legend()
plt.show()

x_train_pred = model(x_train)
y_train_pred_ether = x_train_pred[:]
print(x_train_pred.shape)
print(y_train_pred_ether.shape)

x_test_pred = model(x_test)
y_test_pred_ether = x_test_pred[:]
print(x_test_pred.shape)

y_train[:].detach().numpy().reshape(-1, 1).shape
y_train_pred_ether.reshape(-1, 1).shape

scaler.inverse_transform(y_train[:].detach().numpy().reshape(-1, 1))

```

```

"""### 4"""

print("After Training Completion")
print("MSE")
print('Train Score: %.5f' % (mse_error_train[-1]))
print('Test Score: %.5f' % (mse_error_test[-1]))
print("RMSE")
rmse_train = math.sqrt(mean_squared_error(y_train[:].detach().numpy().reshape(-1, 1),
y_train_pred_ether.detach().numpy().reshape(-1, 1)))
print('Train Score: %.5f' % (rmse_train))
rmse_test = math.sqrt(mean_squared_error(y_test[:].detach().numpy().reshape(-1, 1),
y_test_pred_ether.detach().numpy().reshape(-1, 1) ))
print('Test Score: %.5f' % (rmse_test))
print("MAPE")
mape_train = mean_absolute_error((y_test[:].detach().numpy().reshape(-1, 1)),
y_test_pred_ether.detach().numpy().reshape(-1, 1) )*100
print('Train Score: %.5f' % mape_train)
mape_test = mean_absolute_error((y_test[:].detach().numpy().reshape(-1, 1)),
y_test_pred_ether.detach().numpy().reshape(-1, 1) )*100
print("Test Score: %.5f" % mape_test)
print("R2 Error")
r2_train = r2_score((y_test[:].detach().numpy().reshape(-1, 1)),
y_test_pred_ether.detach().numpy().reshape(-1, 1) )*100
print('Train Score: ',r2_train)
r2_test = r2_score((y_test[:].detach().numpy().reshape(-1, 1)),
y_test_pred_ether.detach().numpy().reshape(-1, 1) )*100
print("Test Score: ", r2_test)

"""# 5"""

scaler.inverse_transform(y_test[:].numpy().reshape(-1, 1))[:5]

y_test_pred_ether.reshape(-1, 1)[:5]

len(y_train_pred_ether)-len(y_test_pred_ether)

ls = []
ls2 = []
for i in range(0,69):
    ls.append(i)
for i in range(69,86):
    ls2.append(i)


len(y_train_pred_ether)

len(y_test_pred_ether)

plt.plot(ls,y_train_pred_ether.detach().numpy().reshape(-1, 1), label = "Training
Predictiona")
plt.plot(ls,y_train, label = "Training Actual Values")
plt.plot(ls2,y_test_pred_ether.detach().numpy().reshape(-1, 1), label = "Testing
Predictions")
plt.plot(ls2,y_test, label = "Testing Actual Values")
plt.legend()
plt.xlabel("Days")
plt.ylabel("Normalized Ether Prices")
plt.show()


```

PLAGIARISM REPORT

English | 繁體中文 | 简体中文

Submissions OverviewLogout

Submissions Overview



Background Information [\[what is this?\]](#)
Batch file name: Final Report (draft).docx
Report generated on: 30/11/2022, 11:33:16 PM

Checking Parameters [\[what is this?\]](#)
Matching scope(s): Within submission, Internet
Leniency: Detailed matching with threshold 70%
Minimum sentence length: Sentences with more than or equal to 3 meaningful words were checked

Similarity Statistics

Similarity Statistics [\[what is this?\]](#)
Total number of documents: 1
Number of documents which can be processed: 1
Number of documents which cannot be processed: 0

Show 10 entries

Search:

Entry	Document	Status	Similarity	Action
1	FinalReport_draft.docx	processed	85/1157=7.30%	View details

Showing 1 to 1 of 1 entries

[First](#) | [Previous](#) | [Next](#) | [Last](#)

[Home](#) | [Services](#) | [News](#) | [Partners](#) | [About](#)

© 2005-2014 The Chinese University of Hong Kong [Terms of use](#)