Bitcoin Market Fluctuation Analysis using Textual Data

A

Project Report

submitted in partial fulfillment of the
requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

Specialization in
Business Analytics and Optimization

by

Student Name	**Roll Number**
Aswin S	R103218027
Indrakshee Roy Choudhury	R103218060
Milisha Gupta	R103218082
Sarthak Saraiya	R103218129

under the guidance of

Mr. Rahul Kumar Singh

Department of Informatics
School of Computer Science
University of Petroleum & Energy Studies
Bidholi, Via Prem Nagar, Dehradun, UK

May– 2021

DECLARATION

We hereby certify that the project work entitled **"Bitcoin Market Fluctuation Analysis using Textual Data"** in partial fulfillment of the requirements for the award of the Degree of BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE AND ENGINEERING specialization **Business Analytics and Optimization** and submitted to the Department of Informatics, School of Computer Science, University of Petroleum & Energy Studies, Dehradun, is an authentic record of my/ our work carried out during a period from **January, 2021 to May, 2021** under the supervision of **Mr. Rahul Kumar Singh.**

The matter presented in this project has not been submitted by me/ us for the award of any other degree of this or any other University.

Milisha Gupta	**Sarthak Saraiya**	**Indrakshee Roy Choudhury**	**Aswin S**
R103218082	**R103218129**	**R103218060**	**R103218027**

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date: 02-05-2021

Dr. Thipendra Pal Singh

Head**,** Department of Informatics

School of Computer Science

University of Petroleum & Energy Studies

Dehradun – 248001 (Uttarakhand)

Mr. Rahul Kumar Singh

Assistant Professor of Informatics

School of Computer Science

University of Petroleum & Energy Studies

Dehradun – 248001 (Uttarakhand)

ACKNOWLEDGEMENT

We wish to express our deep gratitude to our mentor Mr. Rahul Kumar Singh sir, for all the advice, encouragement and constant support he has given us throughout our project work. This work would not have been possible without his support and valuable suggestions.

We sincerely thank our respected Head of Department, Informatics, **Dr. Thipendra Pal Singh,** for his guidance and support as and when required.

We are also grateful to Dr. **Priyadarsan Patra, Dean SCS,** for providing the necessary facilities to carry out our project work successfully.

We would like to thank all our friends for their help and constructive criticism during our project work. Finally, we have no words to express our sincere gratitude to our **parents** who have shown us this world and for everything they have given to us.

Milisha Gupta	**Sarthak Saraiya**	**Indrakshee Roy Choudhury**	**Aswin S**
R103218082	**R103218129**	**R103218060**	**R103218027**

ABSTRACT

Bitcoin alongside other cryptocurrencies became one of the largest trends recently, due to its redefinition of the concept of money, and its price fluctuation. Especially on the social media, people keep discussing Bitcoin topics, consulting, and advising about cryptocurrency trading. As contradicting opinions increase, it is worthy to dive into discussions on social media and use methods to evaluate public's non-negligible role in crypto price fluctuation. This project explores how user sentiments on Bitcoin over various social media platforms and news headlines affect market fluctuations. Sentiment analysis, which is a notably method in text mining, can be used to extract the sentiment from people's opinion. It then provides us with valuable perception on a topic from the public's attitude, which create more opportunities for deeper analysis and prediction.

This project aims to investigate public's sentiment towards Bitcoin over various social media platforms and news headlines and assigning sentiment points on tweets that would later help in predicting market fluctuations.

TABLE OF CONTENTS

-

LIST OF FIGURES & TABLES

1. INTRODUCTION

Sentiments and emotions of people across the globe directly affects the prices of virtual currency like Bitcoin, Litecoin, Ethereum, etc. Of these cryptocurrencies, undoubtedly the most popular has been Bitcoin and it was also the first cryptocurrency in the market. What people think about Bitcoin alters the current price of the Bitcoin. If a celebrity tweets something about Bitcoin, then people start investing in the crypto-currency and the price increases as the demand is more and supply is always limited.

The rise in the value of Bitcoin has resulted in the growth of a new asset: cryptocurrencies. Considering the amount of known information with regards to traditional financial markets, we have many different pricing models as well as models which analyzes the sentiments that have been proven over time. However, when we begin to examine the sentiment formation for cryptocurrencies, we find that there has not been a comparable model yet developed. Further, attempting to use traditional asset sentiment models results in inaccuracies. Due to the unproven nature of cryptocurrencies, the value of each coin rests much more on sentiment in comparison to the impact of sentiment on traditional asset valuation.

This project aims to study the impact of human emotions on the price movements of the cryptocurrency, particularly Bitcoin, by analyzing the effect of sentiment contained in Twitter posts (tweets) and news articles. This is done by implementing data mining techniques to collate tweets that try to infer the relationship between information contained in such items and cryptocurrency price direction. The dataset for the project has been made manually and contains tweets and news headlines that are labelled according to the text. Later preprocessing steps are applied to the dataset and Machine learning models have been used to find out whether a text is related to the increase of the price of BTC or related to the decrease of the price.

2. RELATED WORK

Cryptocurrency is a decentralized virtual currency, based on blockchain technology, which uses cryptography for security. The most well-known cryptocurrency is Bitcoin, which was first introduced in 2009 by Satoshi Nakamoto [1]. Unlike fiat, currencies issued by traditional central banks, Bitcoin's supply is limited as there can only be 21 million Bitcoins in existence. The value of Bitcoin is not backed by gold or other commodities, but it's based solely on a value that people assign to it.

A variety of studies have been performed on using sentiment analysis on micro blogs to predict movements in the Bitcoin market. Bollen et al., [1] for instance showed in their study that when days contained a lot of tweets with, according to Google-Profile of Mood States (GPOMS), a "calm" sentiment, the DJIA rose over the new few days. The study of Oh and Sheng [2] shows that micro blog sentiments contain information that can be valuable to investment decision making and that people do in fact use social media to influence their investment decisions.

Another study conducted by [3] implies that the sentiment analysis of public mood can be used for forecasting Bitcoin prices. The event study conducted by Ranco et al., [4] shows that although the period of impact of Twitter sentiment on price returns is lower for only non-earnings announcements events, the impact is still statistically significant. The results of all these studies imply that sentiment analysis can indeed be used for predicting Bitcoin market movement.

Just like on stocks, studies on sentiment affecting cryptocurrency have also been carried out, however in lower volume, which have resulted in mixed conclusions. [5] conclude that social media can be used to predict future Bitcoin returns as a shock of positive postings indicate positive returns the next day and a negative shock indicates a negative return the next day. Additionally, [6] and [7] were able forecast prices with an over 75% accuracy. Whereas [8] neglected the sentiment analysis because of the lack of relationship between sentiment and price. They concluded that even when prices decreased the sentiment was still high.

3. PROBLEM STATEMENT

In the last decade, cryptocurrency has emerged in the financial area as a key factor in businesses and financial market opportunities. The crypto-currency price prediction is considered a very challenging task, due to its chaotic and very complex nature and we can see that there has not been a comparable model yet developed. Further, attempting to use traditional asset pricing models to determine the price formation of crypto-currencies results in inaccuracies because of the many differences between crypto-currencies as an asset class and traditional assets. The main reason why the traditional models cannot be applied to the prediction of cryptocurrency is due to the public/investor sentiment with regards to each asset. Due to the unproven nature of cryptocurrencies, the value of each coin rests much more on sentiment in comparison to the impact of sentiment on traditional asset valuation.

In this project we analyze Bitcoin Market fluctuation using text mining techniques like finding the target dataset and design preprocessing functions without implementing any core NLP library and by training the dataset by suitable machine learning models to check the negative and positive polarity of the tweets.

4. OBJECTIVE

Our objective is to analyze Bitcoin Market fluctuation using text mining techniques which include:

- Finding Bitcoin-related news headlines or Twitter data manually.
- Segregation of textual data into labelled classes.
- Preprocessing the textual data to make it fit for further processes.
- Mining structured opinions from the textual data using Machine learning algorithms.
- Finding a partial correlation between the price fluctuation of Bitcoin and the fluctuation of the sentiment classes using machine learning algorithms.
- Comparing the accuracies of the machine learning algorithms used.

5. DESIGN

Sentiment analysis, also referred to as opinion mining, is a sub machine learning task where we want to determine which is the general sentiment of a given document. Using machine learning techniques and natural language processing we can extract the subjective information of a document and try to classify it according to its polarity such as “positive” or “negative” sentiment by building a model based on probabilities.

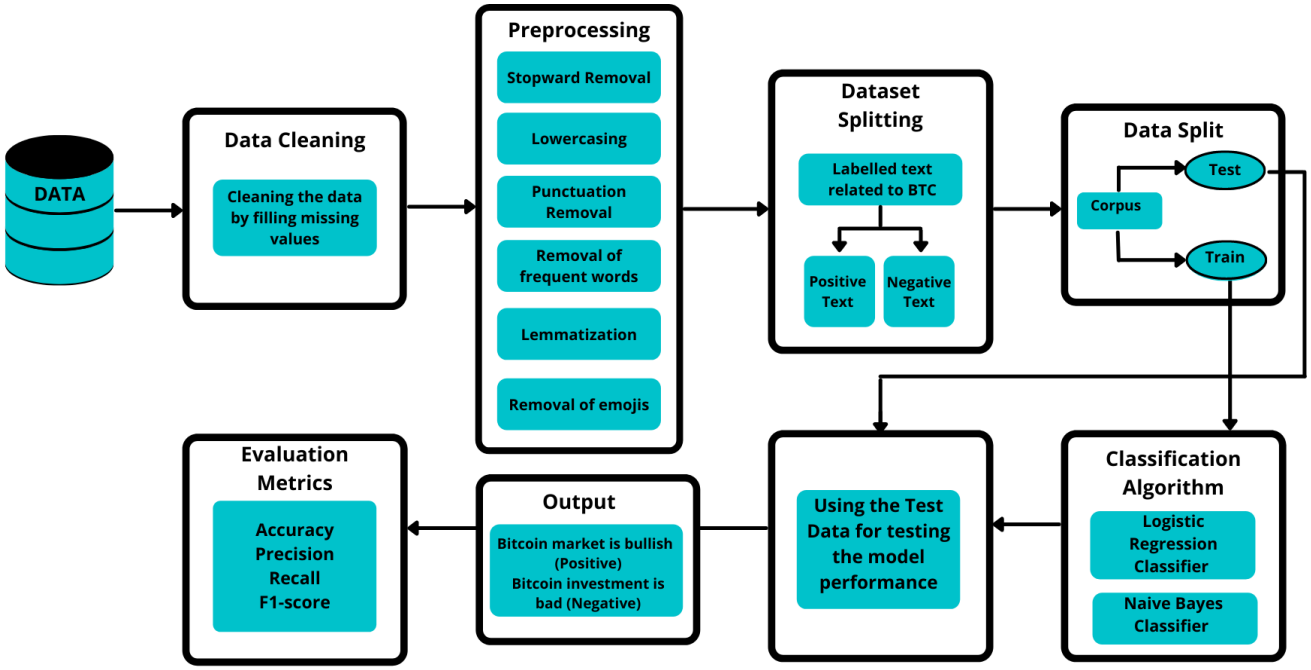

Figure 1: Workflow of the project

Data Preparation: News headlines and social media sites like Twitter are rich sources of information which makes it easier to extract the opinions of people on various topics. But usually, the data found on these platforms are unstructured and hence need to go through a data cleaning process before the sentiments can be correctly analyzed. The unlabeled data extracted is first labelled manually and then stored in the form of comma-separated values files with “tweets” related to Bitcoin along with other relevant columns.

Resources: To facilitate the pre-processing part of the data, we introduce five resources which are,

● A stop word dictionary corresponding to words which are filtered out before or after processing of natural language data because they are not useful in our case.
●A word form dictionary which helps in the building up of the lemmatization function.

Data preprocessing: Raw tweets and scraped from twitter and other platforms generally result in a noisy and obscure dataset. Tweets have certain special characteristics such as retweets, emoticons, user mentions, etc. which should be suitably extracted. Therefore, raw twitter data must be normalized to create a dataset which can be easily learned by various classifiers. We have applied an extensive number of pre-processing steps to standardize the dataset and reduce its size. The data preprocessing steps which we have performed to make the data suitable for use are:

1) Lowercase Conversion
2) Punctuation Removal
3) Retweet Removal
4) URL Removal
5) Emoji Removal
6) Stopword Removal
7) Tokenization
8) Lemmatization

After obtaining the preprocessed data, we divide the data set into two parts, the training set and the test set. To do this, we first shuffle the data set to get rid of any order applied to the data, then we form the set of positive tweets and the set of negative tweets, we take 3/4 of tweets from each set and merge them together to make the training set. The rest is used to make the test set.

Machine Learning: Once our dataset is ready, we can now focus on training our model with the help of suitable machine learning algorithms. In our project Logistic Regression and Naive Bayes machine learning classification model has been applied, the accuracy and test cases have been compared respectively.

LOGISTIC REGRESSION

Step 1.

Building Frequency dictionary

Now, we will create a function that will take tweets and their labels as input, go through every tweet, preprocess them, count the occurrence of every word in the data set and create a frequency dictionary.

Step 2.

Sigmoid Function

Logistic regression makes use of the sigmoid function which outputs a probability between 0 and 1. The sigmoid function with some weight parameter θ and some input x^{(i)}$x(i)$ is defined as follows: -

$$h(x^{(i)}, \theta) = 1/(1 + e^{(-\theta^T * x^{(i)})}.$$

The sigmoid function gives values between -1 and 1 hence we can classify the predictions depending on a particular cutoff. (say: 0.5)
Note that as $(\theta^T)x(i)$ gets closer and closer to $-\infty$ the denominator of the sigmoid function gets larger and larger and as a result, the sigmoid gets closer to 0. On the other hand, $(\theta^T)x(i)$ gets closer and closer to ∞ the denominator of the sigmoid function gets closer to 1 and as a result the sigmoid also gets closer to 1.

Step 3.

Cost Function and Gradient Descent
The logistic regression cost function is defined as

$$J(\theta)=(-1/m)*\sum i=1 \text{ to } m[y(i)\log(h(x(i),\theta)+(1-y(i))\log(1-h(x(i),\theta))]$$

We aim to reduce cost by improving the theta using the following equation:

$$\theta j:=\theta j-\alpha*\partial J(\theta)/\theta j$$

Here, α is called the learning rate. The above process of making hypothesis (h) using the sigmoid function and changing the weights (θ) using the derivative of cost function and a specific learning rate is called the Gradient Descent Algorithm.

The parameter θ is initialized, that you can use in your sigmoid, then the gradient is computed which is used to update θ, and then the cost is calculated. You keep doing so until good enough. Now, a function is created that will extract features from a tweet using the 'freqs' dictionary and above defined preprocessing function. After executing all the required functions, we can finally train our model using the training data set and test it on the test data set. On testing the model using the test data set an accuracy in the range of 60-70% was acquired.

NAIVE BAYES

Naive Bayes algorithm is based on the Bayes rule, which can be represented as follows:

$$P(X|Y) = P(Y)P(Y|X)\ P(X)$$

Here, the process up to creating a dictionary of frequencies (importing libraries, preprocessing, etc.) is the same.

The way the algorithm works is as follows:-

Step 1:

Find the log of the ratio of the number of positive tweets and negative sentiment tweets. i.e. logprior :-

$$\log(P(Dpos)) - \log(P(Dneg)) = \log(Dpos) - \log(Dneg)$$

Step 2:

Instead of keeping the frequencies of each word with the positive and negative labels we take the ratio of their frequency in that label by the total number of frequencies in that label. This will give the probability of occurrence of that word given the tweet is positive/negative.

Step 3:

Then we make another property called *loglikelihood.* It is the log of the ratio of Positive probability to that of the negative probability of a particular word.

To compute the positive probability and the negative probability for a specific word in the vocabulary, we'll use the following inputs:

- $freqpos$ and $freqneg$ are the frequencies of that specific word in the positive or negative class. In other words, the positive frequency of a word is the number of times the word is counted with the label of 1.
- Npos and Nneg are the total numbers of positive and negative words for all documents (for all tweets), respectively.
- V is the number of unique words in the entire set of documents, for all classes, whether positive or negative.

We'll use these to compute the positive and negative probability for a specific word using this formula:

$$(Wpos) = (freqpos + 1)/(Npos + V)$$

$$(Wneg) = (freqneg + 1)/(Nneg + V)$$

Notice that we add the "+1" in the numerator for additive smoothing.

And the loglikelihood can be represented as: $\text{loglikelihood} = \log(P(Wpos)/P(Wneg))$

.

PREDICTING USING NAIVE BAYES

In order to predict the sentiment of a tweet we simply have to sum up the loglikelihood of the words in the tweet along with the logprior. If the value is positive, then the tweet shows positive sentiment but if the value is negative then the tweet shows negative sentiment.

So let's write the predicting (takes in a tweet, loglikelihood, and logprior and returns the prediction) and a testing function (to test the model using the test data set).

On testing the model on the test data set an accuracy of **65%-70%** was achieved.

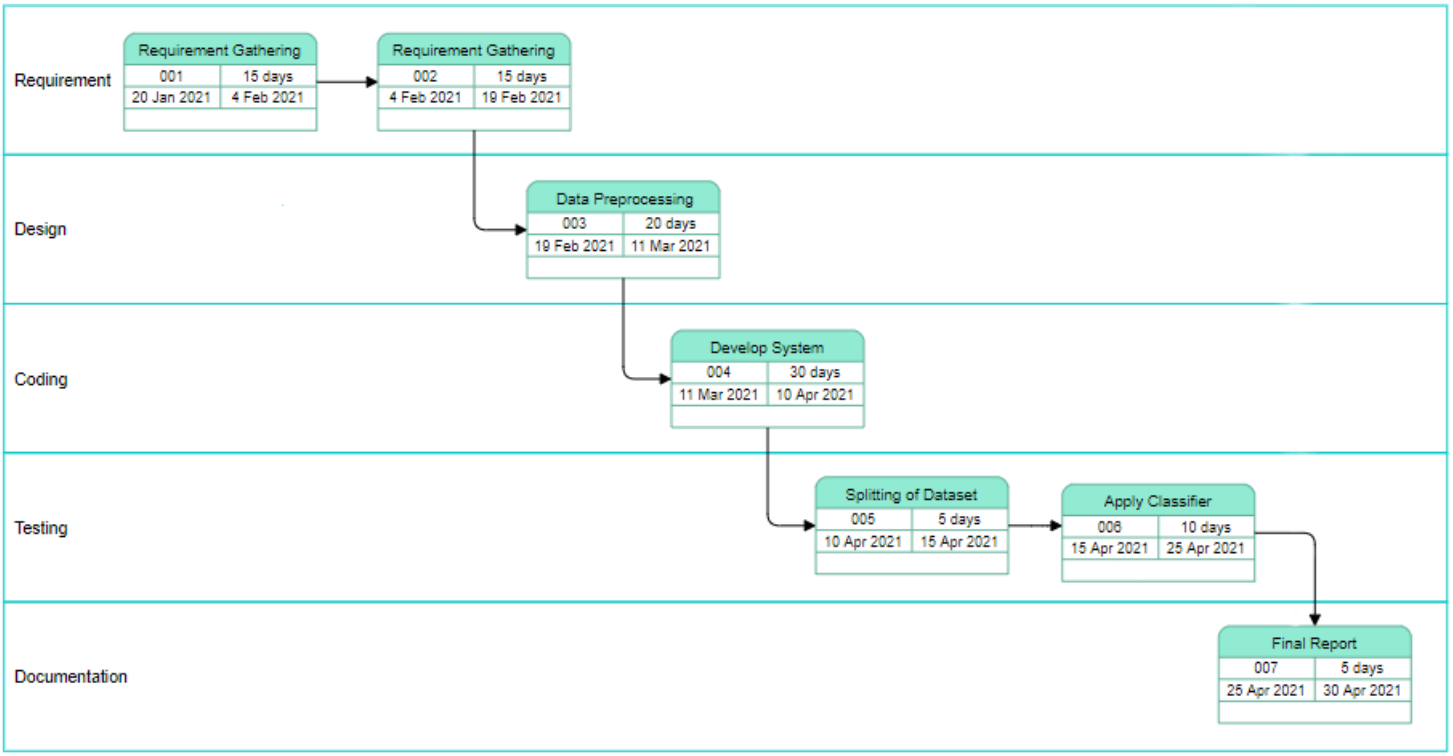

Figure 2: Timeline

6. IMPLEMENTATION

- **Data Preprocessing**

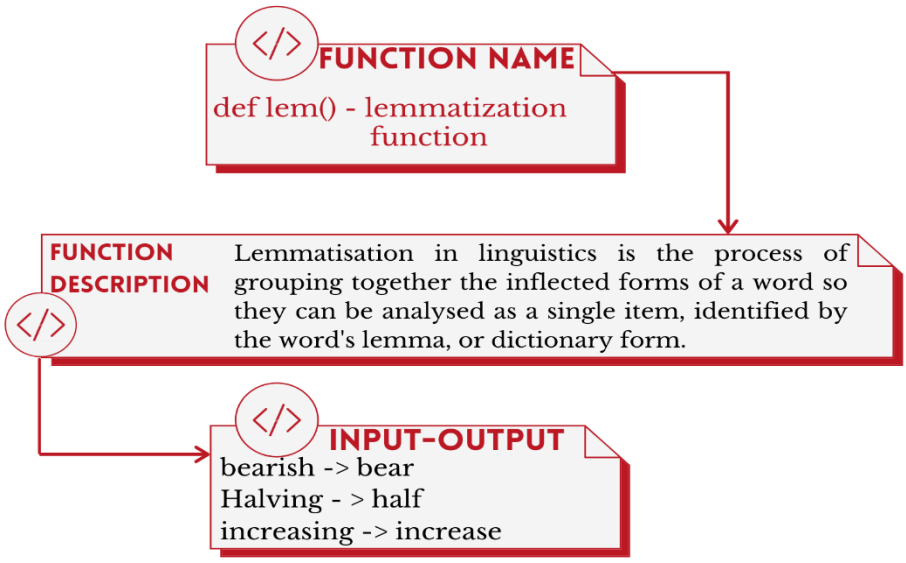

Figure 3: Lemmatization function

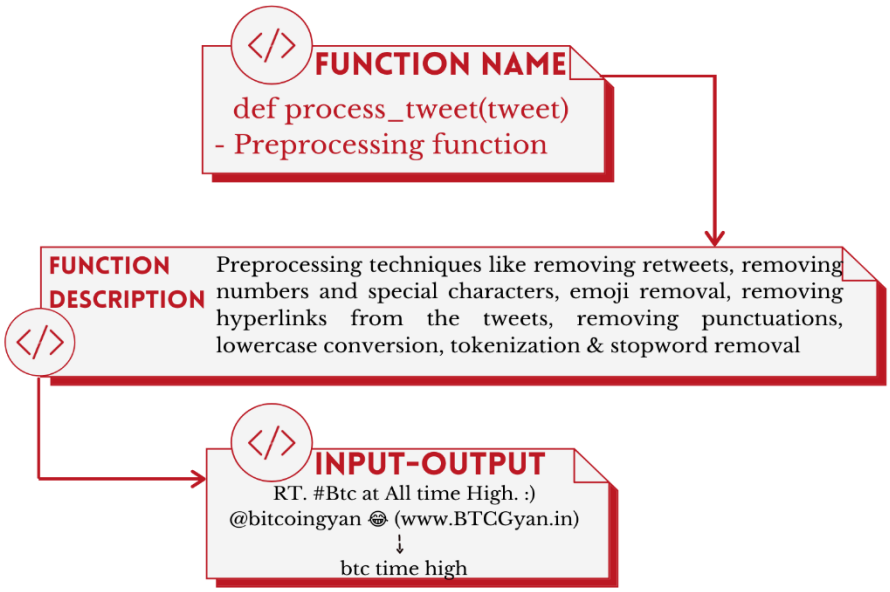

Figure 4: Process tweet function

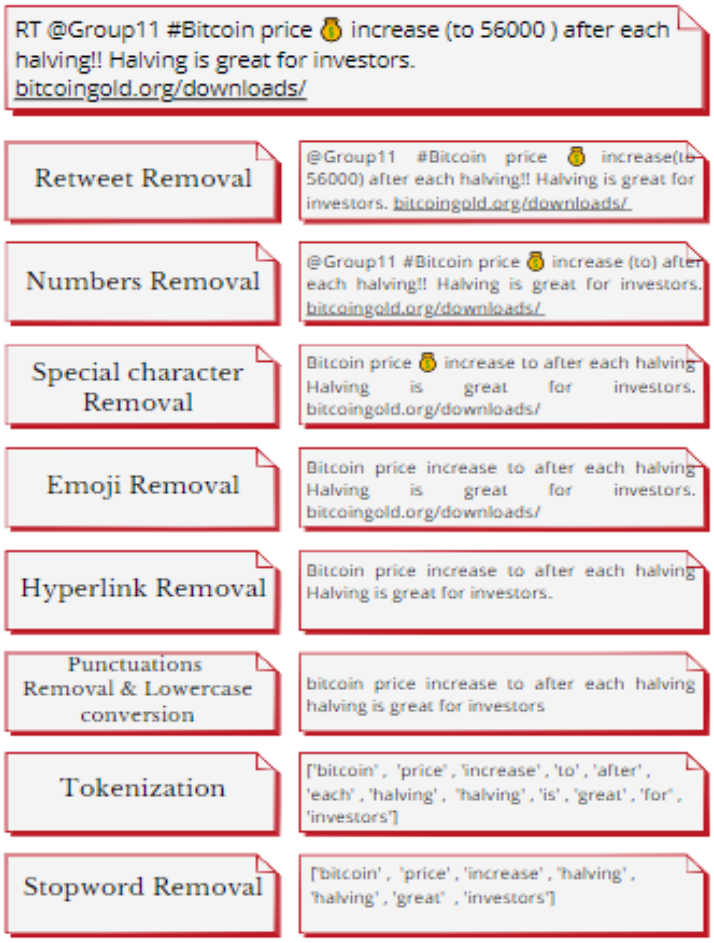

Figure 5: Example of cleaned text

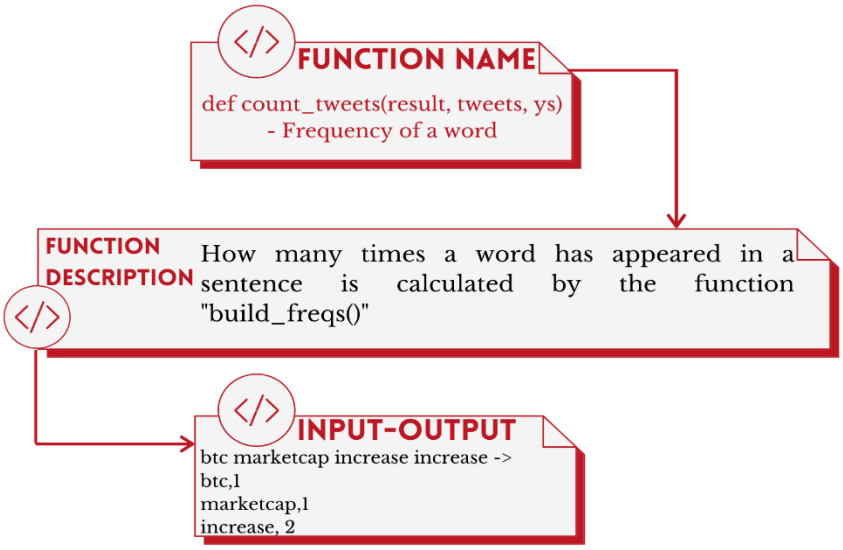

Figure 6: Frequency of tweets function

- **Naïve Bayes**

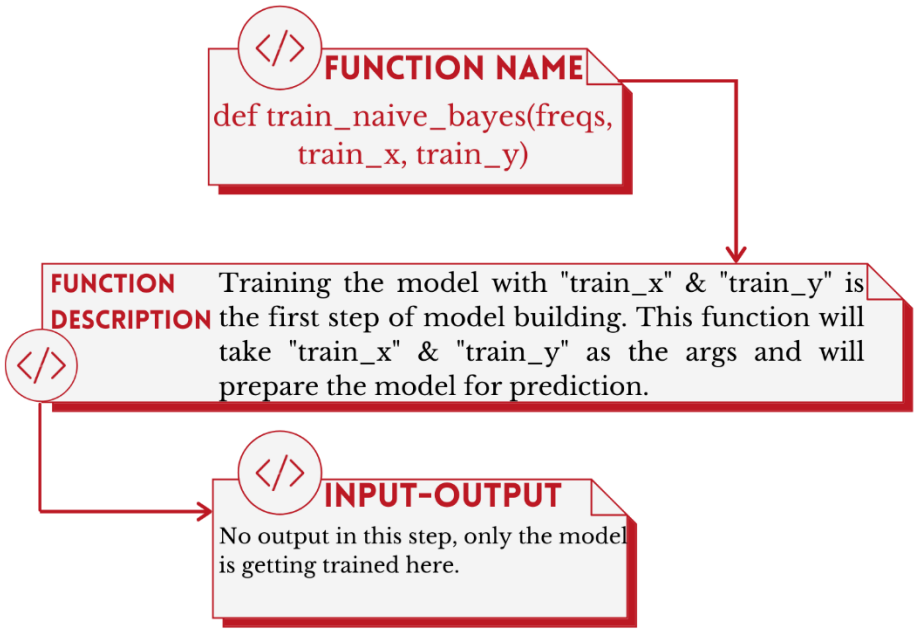

Figure 7: Naive Bayes model train function

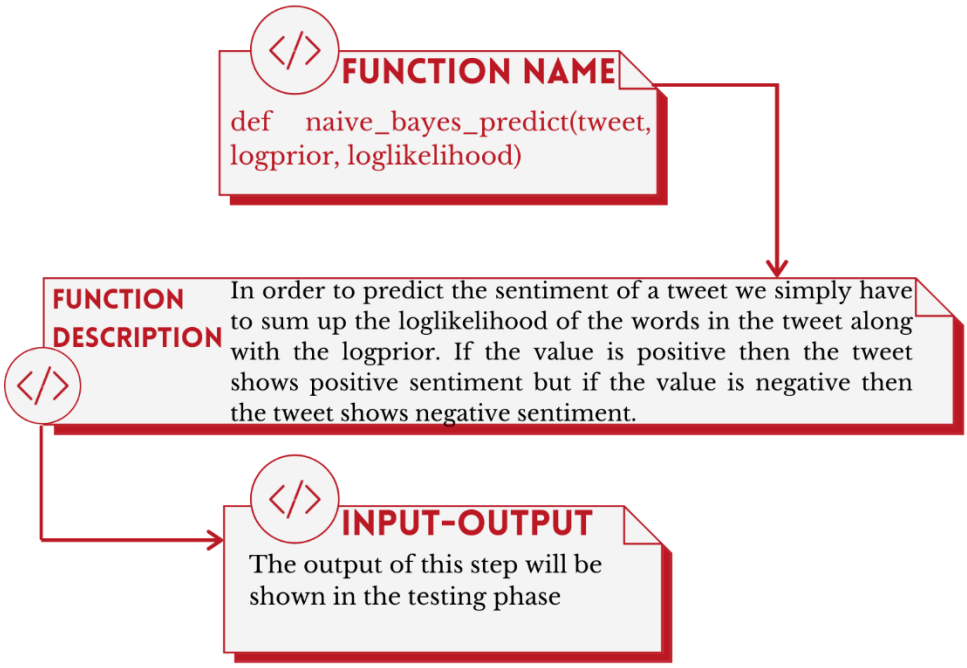

Figure 8: Naive Bayes predict function

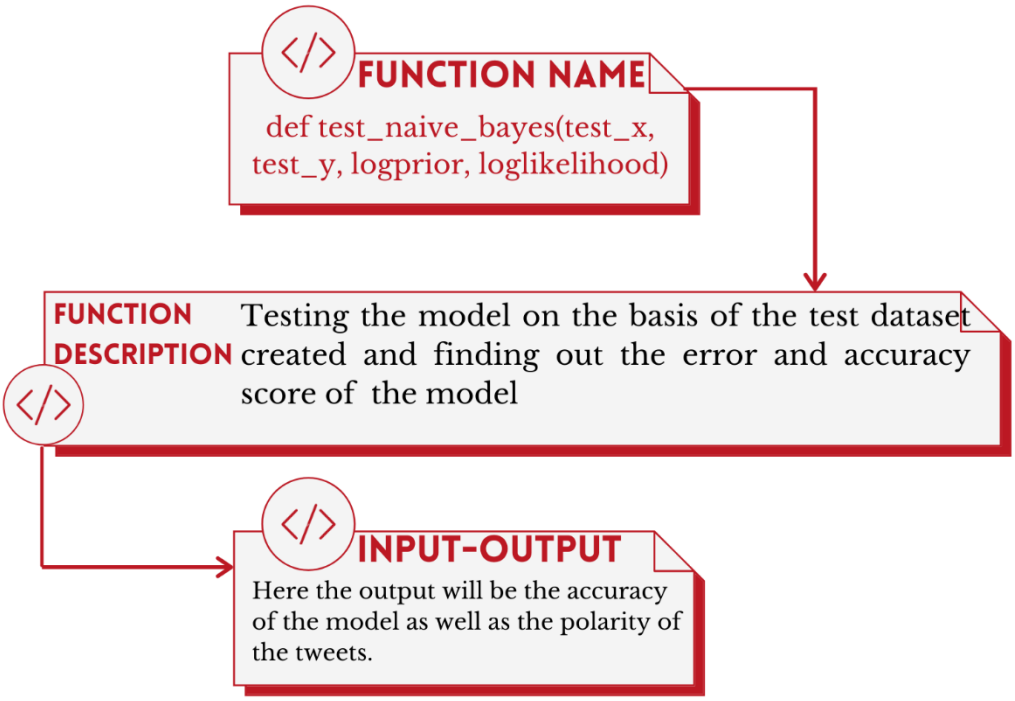

Figure 9: Naive Bayes test function

- **Logistic Regression**

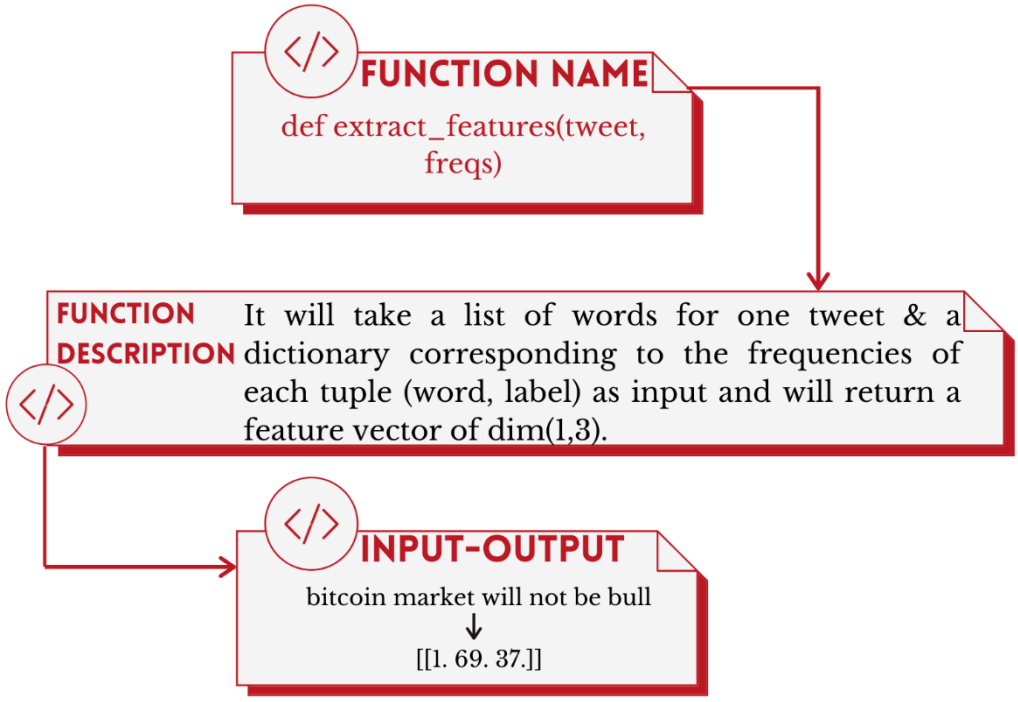

Figure 10: Feature extraction function

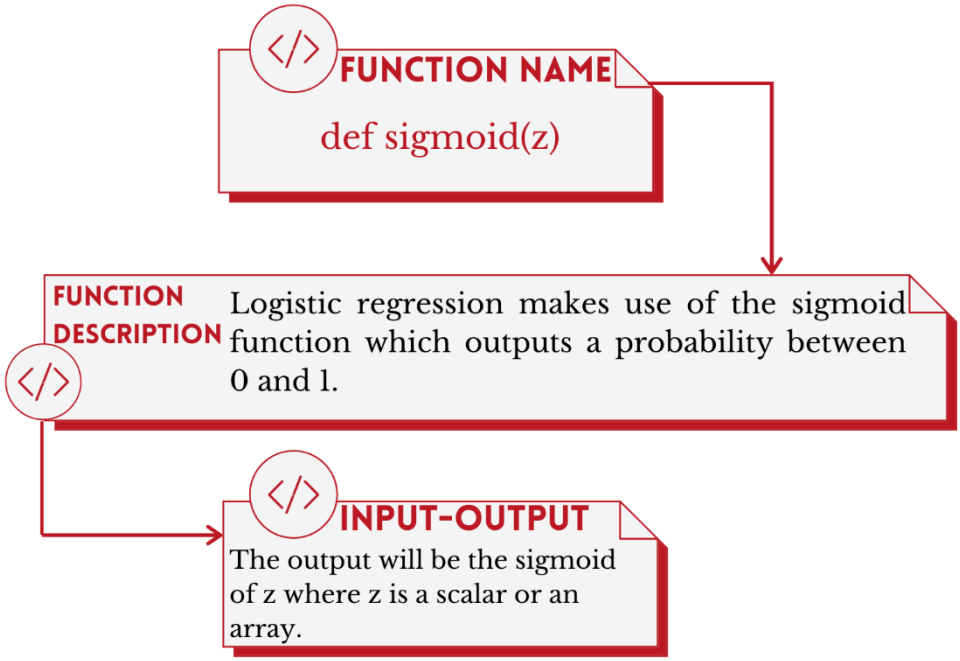

Figure 11: Sigmoid function

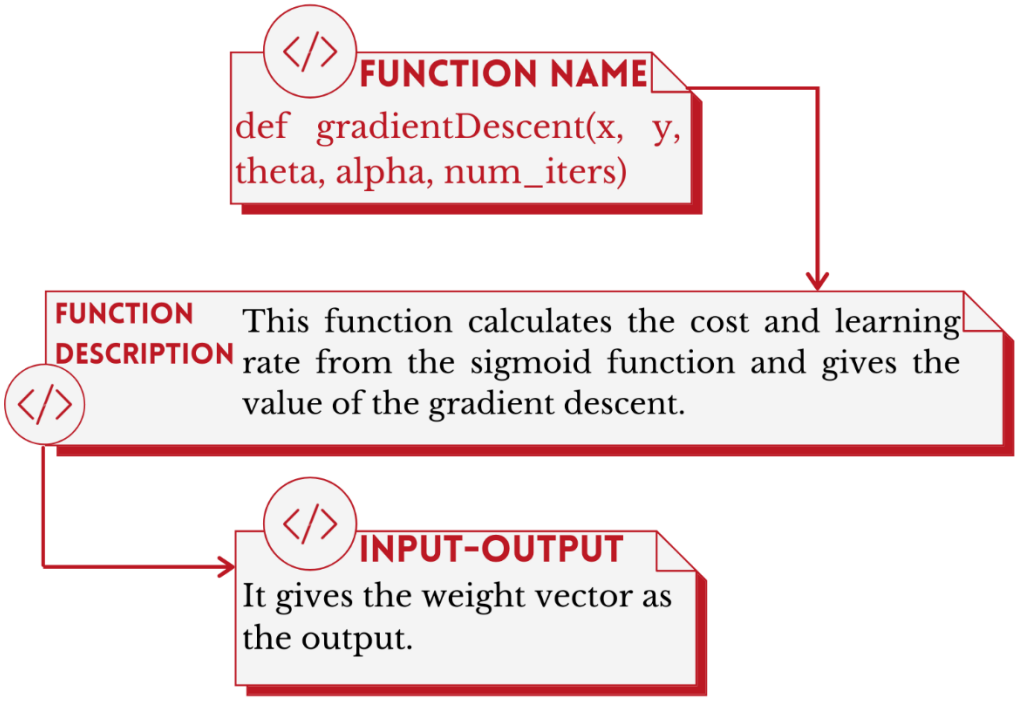

Figure 12: Gradient descent function

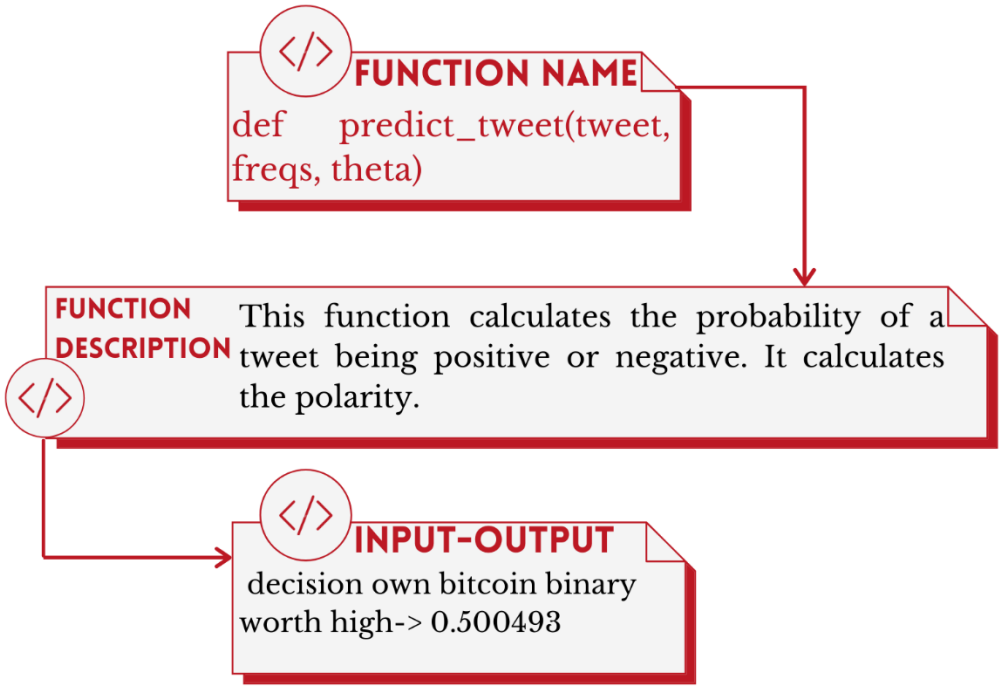

Figure 13: Logistic Regression model predict function

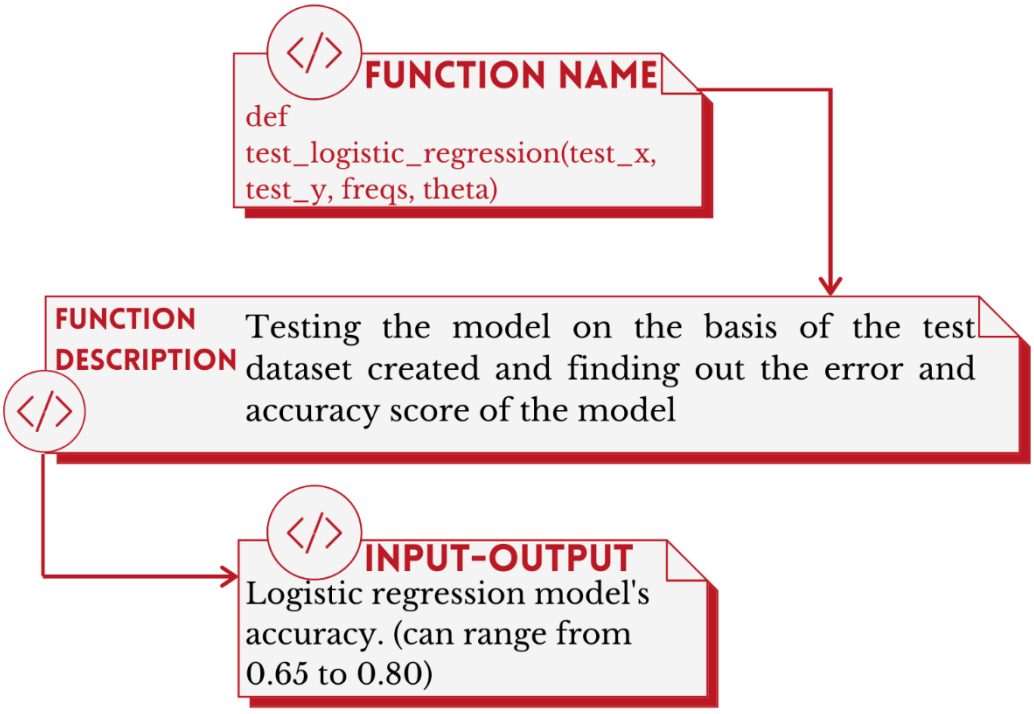

Figure 14: Logistic Regression test function

7. OUTPUT SCREEN

Naïve bayes

```
Naive Bayes accuracy = 0.6125
```

```
for tweet in ['Bitcoin market is bullish', 'Bitcoin is good', 'Invest in Bitcoin as price
    # print( '%s -> %f' % (tweet, naive_bayes_predict(tweet, logprior, loglikelihood)))
    p = naive_bayes_predict(tweet, logprior, loglikelihood)
#     print(f'{tweet} -> {p:.2f} ({p_category})')
    print(f'{tweet} -> {p:.2f}')
```

```
['bitcoin', 'market', 'is', 'bullish']
Bitcoin market is bullish -> 1.46
['bitcoin', 'is', 'good']
Bitcoin is good -> 2.82
['invest', 'in', 'bitcoin', 'as', 'prices', 'are', 'going', 'up']
Invest in Bitcoin as prices are going up -> 3.91
['bitcoin', 'prices', 'are', 'bad', 'and', 'going', 'down']
Bitcoin prices are bad and going down -> -1.84
['bitcoin', 'invetment', 'is', 'bad']
Bitcoin invetment is bad -> -1.25
```

Figure 15: Output screen of NB

Logistic Regression

```
tmp_accuracy = test_logistic_regression(test_x, test_y, freqs, theta)
print(f"Logistic regression model's accuracy = {tmp_accuracy:.4f}")
```

```
Logistic regression model's accuracy = 0.6522
```

Figure 16: Output screen of LR

8. RESULT ANALYSIS

On testing both the models on the test data set we get an accuracy of 65% for Logistic regression and 61% Naive Bayes. Naive Bayes has slightly less accurate as compared to Logistic Regression. This can be due to the fact that the Logistic Regression algorithm doesn't make as many assumptions as that of the Naive Bayes algorithm.
The assumptions made by Naive Bayes are as follows:

1. Independence assumption

"It is sunny and hot in the Sahara desert."

Figure 17: Example sentence 1

"It's always cold and snowy in ___."

Figure 18: Example sentence 2

In figure 15, you can see the word sunny and hot tend to depend on each other and are correlated to a certain extent with the word "desert". If you were to fill in the sentence in figure 16, this naïve bayes model will assign equal weight to the words "spring, summer, fall, winter".
Naive Bayes assumes that the features are conditionally independent. Real data sets are never perfectly independent, but they can be close. In short Naive Bayes has a higher bias but lower variance compared to logistic regression. If the data set follows the bias, then Naive Bayes will be a better classifier.

2. Relative frequencies

On Twitter, there are usually more positive tweets than negative ones. However, some "clean" datasets you may find are artificially balanced to have the same amount of positive and negative tweets. Just keep in mind, that in the real world, the data could be much noisier.

Comparison between the algorithms:

Comparison Factor	Naive bayes	Logistic regression
Type of problems	Used to determine if a sample belongs to a certain class i.e Classification Use Case	Used to determine if a sample belongs to a certain class i.e Classification Use Case
Learning mechanism	Naive Bayes is a generative model that models the joint distribution of the feature X and target Y, and then predicts the posterior probability given as P(y|x)	Logistic regression is a discriminative model that directly models the posterior probability of P(y|x) by learning the input to output mapping by minimizing the error.
Model assumptions	Naïve Bayes assumes all the features to be conditionally independent. So, if some of the features are in fact dependent on each other (in case of a large feature space), the prediction might be poor.	Logistic regression splits feature space linearly, and typically works reasonably well even when some of the variables are correlated.
Approach to be followed to improve model results	When the training data size is small relative to the number of features, the information/data on prior probabilities help in improving the results.	When the training data size is small relative to the number of features, including regularization such as Lasso and Ridge regression can help reduce overfitting and result in a more generalized model.

Table 1: Comparison between the algorithms

9. CONCLUSION

From the above results, we can see that the Logistic Regression algorithm has performed relatively well as compared to the Naive Bayes algorithm. As we have clearly discussed about accuracies of both the models in the previous section, we can see that the Logistic Regression algorithm has performed relatively well as compared to the Naive Bayes algorithm. The reason being the assumptions made by the Naïve Bayes classifier.

Thus the following approaches can be adopted to improve model results:

Naïve Bayes: When the training data size is small relative to the number of features, the information/data on prior probabilities help in improving the results.

Logistic Regression: When the training data size is small relative to the number of features, including regularization such as Lasso and Ridge regression can help reduce overfitting and result in a more generalized model.

The primary conclusion drawn was that when the analyzed tweets yield a positive sentiment, the Bitcoin prices supposedly goes high and it goes down in case of a negative sentiment. Also studies have proved that the most accurate aggregated time to make predictions over was 1 hour, indicating a Bitcoin price change 4 hours into the future.

Further improvements of the analysis would begin with the lexicon and bi-gram approach, as improving the classifier by adding a domain-specific lexicon would identify financial and cryptocurrency terms and similarly proceeding through a bi-gram approach will yield a more representative sentiment, hopefully improving the prediction accuracy of the model.

10. APPENDIX I PROJECT CODE

Link to the Google colab link:

1) Logistic Regression

https://colab.research.google.com/drive/1zZp1gQWXL7TuwGiUiMC6tx4BlnjqZqOF?usp=sharing

2)Naïve Bayes

https://colab.research.google.com/drive/1Ul3ZRRJGoLeR5TylIcBhUEcaF49jNbsX?usp=sharing

```
import pandas as pd
import string
import re
import pandas as pd
df = pd.read_excel("UpdatedMinorDataset.xlsx")
df
df1=pd.read_excel('VerbForms.xlsx')
df1
df1.drop(['Unnamed: 4', 'Unnamed: 5','Unnamed: 6','Unnamed: 7'], axis =
1,inplace=True)
df1.rename(columns={'abash':'v0','abashed':'v1','abashing':'v3','abashes':'v2'
},inplace = True)
df1['v00']=df1['v0']
df1.set_index('v00',inplace=True)
df1
def lem(i):
        if((df1['v1']==i).any()):
            a1=df1[df1['v1']==i].index.values.tolist()
            i=i.replace(i,a1[0])
        if((df1['v2']==i).any()):
            a2=df1[df1['v2']==i].index.values.tolist()
            i=i.replace(i,a2[0])
            #print (i)
        if((df1['v3']==i).any()):
            a3=df1[df1['v3']==i].index.values.tolist()
            i=i.replace(i,a3[0])
        return(i)
lem('halving')

import re
import string
import nltk
```

```
import numpy as np

#Preprocessing tweets
def process_tweet(tweet):
    #Remove old style retweet text "RT"
    tweet2 = re.sub(r'^RT[\s]','', tweet)
    #remove number
    tweet2 = re.sub(r'\d+', "number",tweet2)
    #remove usermentions
    tweet2 = re.sub("(@[A-Za-z0-9_]+)", "USERMENTION", tweet2)
    #url and emoji removal
    text = re.sub('([^0-9A-Za-z \t])|(\w+:\/\/\S+)', ' ', tweet2)
    #Remove hyperlinks
    tweet2 = re.sub(r'https?:\/\/.*[\r\n]*','', tweet2)
    #conversion to lowercase
    tweet2 = tweet2.lower()
    #puntuation removal
    tweet2= tweet2.translate(str.maketrans('', '', string.punctuation))


    #Remove hastags
    tweet2 = re.sub(r'#','',tweet2)


    tweet_tokens =tweet2.split()

    #Stop word dictionary
    stop_words =['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves',
'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself',
'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers',
'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs',
'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being',
'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an',
'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of',
'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through',
'during', 'before', 'after', 'above', 'below', 'to', 'from', 'in', 'out',
'on', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there',
'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'other',
'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than',
'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'should',
"should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren',
"aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn',
"hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma',
'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan',
"shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won',
"won't", 'wouldn', "wouldn't"]
```

```
    text_list = []
    for token in tweet_tokens:
      token=re.sub(r'\d+', "number", token)#basic emoji like -_- or >_<
removal
      if len(token) > 3 and '_' in token:
        token = token.replace('_', ' ')

      if token[0] == '<' and token[-1] == '>':
        token = token[1:-1]
      text_list.append(token)
    final_words = []
    for word in text_list:#stop word removal
      word=lem(word)
      if word not in stop_words:
        final_words.append(word)
    return (final_words)


#Frequency generating function
def build_freqs(tweets, ys):
    yslist = np.squeeze(ys).tolist()

    freqs = {}
    for y, tweet in zip(yslist, tweets):
        for word in process_tweet(tweet):
            pair = (word, y)
            freqs[pair] = freqs.get(pair, 0) + 1

    return freqs

process_tweet('Hi! I am mad')


import pdb
import numpy as np
import pandas as pd

import string
df
df.columns
df.columns= ['date','Label','TWEET']
df
pos=pd.read_excel('Positive.xlsx')
pos.columns= ['date','Label','TWEET']
pos
neg=pd.read_excel('negative.xlsx')
neg.columns= ['date','Label','TWEET']
```

```
neg
'''neg=neg[:325]
neg'''
pos.columns=['time','Label','Tweet']
pos
neg.columns=['time','Label','Tweet']
neg
positive_tweets=[]
for i in pos['Tweet']:
  positive_tweets.append(i)
positive_tweets
negative_tweets=[]
for i in neg['Tweet']:
  negative_tweets.append(i)
len(negative_tweets)
len(positive_tweets)
test_pos = positive_tweets[55:]
train_pos = positive_tweets[:55]
test_neg = negative_tweets[27:]
train_neg = negative_tweets[:27]
train_x = train_pos + train_neg
test_x = test_pos + test_neg
train_x
print(len(test_pos) , len(test_neg))
print(len(train_neg) )
import numpy as np
train_y = np.append(np.ones((len(train_pos), 1)), np.zeros((len(train_neg),
1)), axis=0)
test_y = np.append(np.ones((len(test_pos), 1)), np.zeros((len(test_neg), 1)),
axis=0)
train_y
print("train_y.shape = " + str(train_y.shape))
print("test_y.shape = " + str(test_y.shape))
print(process_tweet(positive_tweets[3]))

def count_tweets(result, tweets, ys):
    '''
    Input:
        result: a dictionary that will be used to map each pair to its
frequency
        tweets: a list of tweets
        ys: a list corresponding to the sentiment of each tweet (either 0 or
1)
    Output:
        result: a dictionary mapping each pair to its frequency
    '''
```

```
    ### START CODE HERE (REPLACE INSTANCES OF 'None' with your code) ###
    for y, tweet in zip(ys, tweets):
        for word in process_tweet(tweet):
            # define the key, which is the word and label tuple
            pair = (word,y)

            # if the key exists in the dictionary, increment the count
            if pair in result:
                result[pair] += 1

            # else, if the key is new, add it to the dictionary and set the
count to 1
            else:
                result[pair] = 1
    ### END CODE HERE ###

    return result

result = {}
tweets = ['Bitcoin market is bull', 'Bitcoin is good', 'Invest in Bitcoin as
it is good', 'Bitcoin prices are bad and going down','Bitcoin invetment is
bad']
ys = [1, 1, 1, 0,0]
count_tweets(result, tweets, ys)
arr = np.array(train_x)
arr
train_y=train_y.tolist()
type(train_y)
train__y=[]
for i in train_y:
  for j in i:
    train__y.append(j)

train_y=train__y
freqs = count_tweets({}, train_x, train_y)
```

NAÏVE BAYES

```
def train_naive_bayes(freqs, train_x, train_y):
    '''
    Input:
        freqs: dictionary from (word, label) to how often the word appears
        train_x: a list of tweets
        train_y: a list of labels correponding to the tweets (0,1)
    Output:
        logprior: the log prior. (equation 3 above)
```

```
        loglikelihood: the log likelihood of you Naive bayes equation.
(equation 6 above)
    '''
    loglikelihood = {}
    logprior = 0


    # calculate V, the number of unique words in the vocabulary
    vocab = set([pair[0] for pair in freqs.keys()])
    V = len(vocab)

    # calculate N_pos and N_neg
    N_pos = N_neg = 0
    for pair in freqs.keys():
        # if the label is positive (greater than zero)
        if pair[1] > 0:

            # Increment the number of positive words by the count for this
(word, label) pair
            N_pos += freqs.get(pair, 1)

        # else, the label is negative
        else:

            # increment the number of negative words by the count for this
(word,label) pair
            N_neg += freqs.get(pair, 1)

    # Calculate D, the number of documents
    D = len(train_y)

    # Calculate D_pos, the number of positive documents (*hint: use
sum(<np_array>))
    D_pos = sum(train_y)

    # Calculate D_neg, the number of negative documents (*hint: compute using
D and D_pos)
    D_neg = D-D_pos

    # Calculate logprior
    logprior = np.log(D_pos) - np.log(D_neg)

    # For each word in the vocabulary...
    for word in vocab:
        # get the positive and negative frequency of the word
        freq_pos = freqs.get((word, 1),0)
        freq_neg = freqs.get((word, 0),0)
```

```
        # calculate the probability that each word is positive, and negative
        p_w_pos = (freq_pos + 1)/(N_pos + V)
        p_w_neg = (freq_neg + 1)/(N_neg + V)

        # calculate the log likelihood of the word
        loglikelihood[word] = np.log(p_w_pos/p_w_neg)

    ### END CODE HERE ###

    return logprior, loglikelihood

logprior, loglikelihood = train_naive_bayes(freqs, train_x, train_y)
print(logprior)
print(len(loglikelihood))
def naive_bayes_predict(tweet, logprior, loglikelihood):
    '''
    Input:
        tweet: a string
        logprior: a number
        loglikelihood: a dictionary of words mapping to numbers
    Output:
        p: the sum of all the logliklihoods of each word in the tweet (if
found in the dictionary) + logprior (a number)
    '''
    # process the tweet to get a list of words
    word_l = process_tweet(tweet)

    # initialize probability to zero
    p = 0

    # add the logprior
    p += logprior

    for word in word_l:

        # check if the word exists in the loglikelihood dictionary
        if word in loglikelihood:
            # add the log likelihood of that word to the probability
            p += loglikelihood[word]

    return p

my_tweet = 'Bitcoin is up'
p = naive_bayes_predict(my_tweet, logprior, loglikelihood)
print('The expected output is', p)
def test_naive_bayes(test_x, test_y, logprior, loglikelihood):
    """
```

```
    Input:
        test_x: A list of tweets
        test_y: the corresponding labels for the list of tweets
        logprior: the logprior
        loglikelihood: a dictionary with the loglikelihoods for each word
    Output:
        accuracy: (# of tweets classified correctly)/(total # of tweets)
    """
    accuracy = 0  #return this properly

    y_hats = []
    for tweet in test_x:
        # if the prediction is > 0
        if naive_bayes_predict(tweet, logprior, loglikelihood) > 0:
            # the predicted class is 1
            y_hat_i = 1
        else:
            # otherwise the predicted class is 0
            y_hat_i = 0

        # append the predicted class to the list y_hats
        y_hats.append(y_hat_i)

    # error is the average of the absolute values of the differences between
y_hats and test_y
    error = np.mean(np.absolute(y_hats - test_y))

    # Accuracy is 1 minus the error
    accuracy = 1 - error

    ### END CODE HERE ###

    return accuracy

print("Naive Bayes accuracy = %0.4f" %
      (test_naive_bayes(test_x, test_y, logprior, loglikelihood)))
for tweet in ['Bitcoin market is bullish', 'Bitcoin is good', 'Invest in
Bitcoin as prices are going up', 'Bitcoin prices are bad and going
down','Bitcoin invetment is bad']:
    # print( '%s -> %f' % (tweet, naive_bayes_predict(tweet, logprior,
loglikelihood)))
    p = naive_bayes_predict(tweet, logprior, loglikelihood)
#     print(f'{tweet} -> {p:.2f} ({p_category})')
    print(f'{tweet} -> {p:.2f}')
my_tweet = 'RBI ready to launch its own digital currency, bad for the market
price would go down'
print(my_tweet)
```

```
process_tweet(my_tweet)
y_hat = naive_bayes_predict(my_tweet, logprior, loglikelihood)
print(y_hat)
if y_hat > 0:
    print('Positive sentiment')
else:
    print('Negative sentiment')
```

LOGISTIC REGRESSION

```
def extract_features(tweet, freqs):
    '''
    Input:
        tweet: a list of words for one tweet
        freqs: a dictionary corresponding to the frequencies of each tuple (word, label)
    Output:
        x: a feature vector of dimension (1,3)
    '''
    # process_tweet tokenizes, stems, and removes stopwords
    word_l = text_process(tweet)

    # 3 elements in the form of a 1 x 3 vector
    x = np.zeros((1, 3))

    #bias term is set to 1
    x[0,0] = 1

    # loop through each word in the list of words
    for word in word_l:

        # increment the word count for the positive label 1
        x[0,1] += freqs.get((word,1),0)

        # increment the word count for the negative label 0
        x[0,2] += freqs.get((word,0),0)

    assert(x.shape == (1, 3))
    return x

# test on training data
tmp1 = extract_features(train_x[0], freqs)
print(tmp1)
tmp2 = extract_features('bitcoin market will not be bull', freqs)
print(tmp2)
```

```
# collect the features 'x' and stack them into a matrix 'X'
X = np.zeros((len(train_x), 3))
for i in range(len(train_x)):
    X[i, :]= extract_features(train_x[i], freqs)

X.shape
Y.shape
def sigmoid(z):
    '''
    Input:
        z: is the input (can be a scalar or an array)
    Output:
        h: the sigmoid of z
    '''
    # calculate the sigmoid of z
    h = 1/(1 + np.exp(-z))

    return h
# Testing your function
if (sigmoid(0) == 0.5):
    print('SUCCESS!')
else:
    print('Oops!')

if (sigmoid(4.92) == 0.9927537604041685):
    print('CORRECT!')
else:
    print('Oops again!')

def gradientDescent(x, y, theta, alpha, num_iters):
    '''
    Input:
        x: matrix of features which is (m,n+1)
        y: corresponding labels of the input matrix x, dimensions (m,1)
        theta: weight vector of dimension (n+1,1)
        alpha: learning rate
        num_iters: number of iterations you want to train your model for
    Output:
        J: the final cost
        theta: your final weight vector
    Hint: you might want to print the cost to make sure that it is going down.
    '''

    m = len(x)

    for i in range(0, num_iters):

        # get z, the dot product of x and theta
        z = np.dot(x,theta)

        # get the sigmoid of z
        h = sigmoid(z)
```

```
        # calculate the cost function
        J = (-1/m)*(np.dot(y.T,np.log(h)) + np.dot((1-y).T,np.log(1-h)))

        # update the weights theta
        theta = theta - (alpha/m)*np.dot(x.T, h-y)

    J = float(J)
    return J, theta
# Check the function
# Construct a synthetic test case using numpy PRNG functions
np.random.seed(1)
# X input is 10 x 3 with ones for the bias terms
tmp_X = np.append(np.ones((10, 1)), np.random.rand(10, 2) * 2000, axis=1)
# Y Labels are 10 x 1
tmp_Y = (np.random.rand(10, 1) > 0.35).astype(float)

# Apply gradient descent
tmp_J, tmp_theta = gradientDescent(tmp_X, tmp_Y, np.zeros((3, 1)), 1e-8, 700)
print(f"The cost after training is {tmp_J:.8f}.")
print(f"The resulting vector of weights is {[round(t, 8) for t in
np.squeeze(tmp_theta)]}")
# training labels corresponding to X
Y = train_y

# Apply gradient descent
J, theta = gradientDescent(X, Y, np.zeros((3, 1)), 1e-9, 1500)
print(f"The cost after training is {J:.8f}.")
print(f"The resulting vector of weights is {[round(t, 8) for t in np.squeeze(theta)]}")

def predict_tweet(tweet, freqs, theta):
    '''
    Input:
        tweet: a string
        freqs: a dictionary corresponding to the frequencies of each tuple (word, label)
        theta: (3,1) vector of weights
    Output:
        y_pred: the probability of a tweet being positive or negative
    '''

    # extract the features of the tweet and store it into x
    x = extract_features(tweet, freqs)

    # make the prediction using x and theta
    z = np.dot(x,theta)
    y_pred = sigmoid(z)


    return y_pred

for tweet in [test_x[20], test_x[12], test_x[5], ]:
    print( '%s -> %f' % (tweet, predict_tweet(tweet, freqs, theta)))
```

```
def test_logistic_regression(test_x, test_y, freqs, theta):
    y_hat = []
    for tweet in test_x:
        # get the label prediction for the tweet
        y_pred = predict_tweet(tweet, freqs, theta)

        if y_pred > 0.5:
            # append 1.0 to the list
            y_hat.append(1)
        else:
            # append 0 to the list
            y_hat.append(0)
    m=len(y_hat)
    y_hat=np.array(y_hat)
    y_hat=y_hat.reshape(m)
    test_y=test_y.reshape(m)

    c=y_hat==test_y
    j=0
    for i in c:
        if i==True:
            j=j+1
    accuracy = j/m
    return accuracy
tmp_accuracy = test_logistic_regression(test_x, test_y, freqs, theta)
print(f"Logistic regression model's accuracy = {tmp_accuracy:.4f}")
tmp_accuracy = test_logistic_regression(train_x, train_y, freqs, theta)
print(f"Logistic regression model's accuracy = {tmp_accuracy:.4f}")
# Some error analysis done for you
print('Label Predicted Tweet')
for x,y in zip(test_x,test_y):
    y_hat = predict_tweet(x, freqs, theta)
    if np.abs(y - (y_hat > 0.5)) > 0:
        print('THE TWEET IS:', x)
        print('THE PROCESSED TWEET IS:', text_process(x))
        print('%d\t%0.8f\t%s' % (y, y_hat, ' '.join(text_process(x)).encode('ascii',
'ignore')))

# Feel free to change the tweet below
my_tweet = test_x[20]
print(my_tweet)
print(text_process(my_tweet))
y_hat = predict_tweet(my_tweet, freqs, theta)
print(y_hat)
if y_hat > 0.500100:
    print('Positive sentiment')
else:
    print('Negative sentiment')
```

11. REFERENCES

[1] J. M. H. &. Z. X. Bollen, "Twitter mood predicts the stock market.," Journal of computational science, vol. 2(1), pp. 1-8, 2011.

[2] C. &. S. O. Oh, Investigating predictive power of stock micro blog sentiment in forecasting future stock price directional movement, 2011, pp. 1-19.

[3] J. G. M. L. N. &. Ž. Smailović, Stream-based active learning for sentiment analysis in the financial domain. Information sciences,, 2014, pp. 285, 181-203.

[4] G. A. D. C. G. G. M. &. M. I. Ranco, The effects of Twitter sentiment on stock price returns, vol. PloS one, 2015, p. 10(9).

[5] F. B. Q. S. J. W. X. S. &. C. H. Mai, The impacts of social media on Bitcoin performance., 2015.

[6] S. R. S. &. S. M. Colianni, "Algorithmic trading of cryptocurrency based on Twitter sentiment analysis," 2015.

[7] E. &. L. J. Stenqvist, Predicting Bitcoin price fluctuation with Twitter sentiment analysis, 2017.

[8] J. H. D. N. J. &. I. J. Abraham, "Cryptocurrency price prediction using tweet volumes and sentiment analysis.," SMU Data Science Review, vol..

[9] J. Kaminski, Nowcasting the bitcoin market with twitter signals, 2014.