International Journal of Computational Intelligence Research
ISSN 0973-1873 Volume 16, Number 2 (2020), pp. 87-104

https://dx.doi.org/10.37622/IJCIR/16.2.2020.87-104

Sentiment Analysis of Tweets and Texts Using Python on Stocks and COVID-19

Yuvraj Jain[1*] and Vineet Tirth[2]

[1]K.R. Mangalam World School, Greater Kailash II, Chittaranjan, New Delhi-110048, India
[2]College of Engineering, King Khalid University, Abha-61411, Asir, Kingdom of Saudi Arabia

ABSTRACT

In a day and age where billions of users regularly use social media and express their opinions online, there is potentially a lot of data that can be harvested and utilized; therefore, it is crucial to develop a quick way to garner data. This study aimed to develop a program using Python to do the same, trying to understand the sentiments of the authors of the text and tweets as well as other additional information about the top tweets and retweets. The main objective of the Twitter Sentiment Analysis is a query-based analysis of tweets. In simple words, Twitter Sentiment Analysis focuses on analyzing the tweets of a specific/particular topic that the user wants to analyze. An extensive collection of such sentiments could leverage to provide a fair reflection of public sentiment towards a specific topic. There are thousands of tweets that can be quickly processed for the sentimental impact, compared to the amount of time it would take a large team of people to complete the same task manually. There are tons of text documents that can also be processed for sentiments in seconds, much faster than just a team of people manually skimming through the text.

Keywords: Tokenization; Stemming; Lemmatization; Bag of Words; Text Frequency; Inverse Document Frequency.

Context: This is an independent study made during the lockdown period of COVID-19 pandemic. It took about five months to complete this study, summarized in this article.

Motivation: The use of social media and twitter has increasingly become popular in the pandemic period. The effect of the COVID-19 pandemic on the sentiments of the society, mainly due to the tweets and comments of the influential people of the society and the response to their tweets was the primary source of motivation behind this study.

Source code: The source code for the study will be made available in the public domain through the link to the GitHub.

1. INTRODUCTION

During the pandemic of COVID-19, people all over the world have been affected, ranging from ordinary people to organizations, geopolitical entities, and entire countries. People have been expressing their sentiments about the pandemic on twitter which is an essential outlet for people to discuss their feelings and opinions. This data can then be mined upon to extrapolate useful information that can be beneficial to gain an understanding of the effects caused by COVID-19 to companies, organizations and also to gain an insight into how people are emotionally reacting to the response of their respective governments to the global pandemic. For instance, during COVID-19 countries like Australia, Belgium and India were tweeting with a positive sentiment, whereas the Chinese had expressed negative sentiments for the same [1-2]. Sentiment analysis is the contextual mining of text, which identifies and withdraws subjective information from the source material that can be useful in many scientific and commercial areas, such as event detection, recommender systems, and opinion mining [3].

2. METHODOLOGY

The first step in determining the sentiment of tweets involves extracting the text and processing it first. This is done through *Tokenization*, which is just a process of breaking up a piece of text into many pieces, such as sentences and words. It works by separating words using spaces and punctuation (Figure 1).

```
In [1]: # Tokenization of paragraphs/sentences
        import nltk

In [3]: paragraph = "This paper describes a weakly supervised system for sentiment analysis in the movie review domain. The objective is

In [5]: nltk.download()
        showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml
Out[5]: True

In [6]: # Tokenizing sentences
        sentences = nltk.sent_tokenize(paragraph)

In [9]: # Tokenizing words
        words = nltk.word_tokenize(paragraph)
```

Figure 1: Tokenization of information.

Here, we obtain a list of individual words which can later be processed to determine the sentiment of the text. In Figure 2, 'paragraph' is a variable, which is currently holding a string of a random text to showcase this example of *Tokenization.*

```
In [4]: paragraph
Out[4]: 'This paper describes a weakly supervised system for sentiment analysis in the movie review domain. The objective is to classif
        y a movie review into a polarity class, positive or negative, based on those sentences bearing opinion on the movie alone, leav
        ing out other irrelevant text. Wikipedia incorporates the world knowledge of movie-specific features in the system which is use
        d to obtain an extractive summary of the review, consisting of the reviewer's opinions about the specific aspects of the movie.
        This filters out the concepts which are irrelevant or objective with respect to the given movie. The proposed system, WikiSent,
        does not require any labeled data for training. It achieves a better or comparable accuracy to the existing semi-supervised and
        unsupervised systems in the domain, on the same dataset. We also perform a general movie review trend analysis using WikiSent.'
```

Figure 2: An example to determine the text sentiment.

2.1 Problems Faced with Tokenization

Because Tokenization yields a list of every single word in the text, it is practically impossible to operate on the data that will be procured. This problem would just be aggravated by the enormous data set of tweets of millions of users. The data set needs to be reduced to a manageable amount. This is done through *Lemmatization, Stemming*, and *Stop-words.*

2.1.1 Removing Stop-words

Stop-words are words that do not contribute to the sentiment of the text, and they do not have much value in a sentiment analysis program. Hence, they are just noise in the text that may be removed [4]. Below are the stop-words, which have been removed.

['he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'only', 'other', 'some', 'such', 'no', 'nor', 'not', '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', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'hadn', "hadn't", 'hasn', "hasn't", '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', , 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'few', 'more', 'most', , 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't",'doesn', "doesn'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", 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'yours elf', 'yourselves']

2.1.2 Stemming with 'ntlk' Package

Through stemming, we decrease the inflection in words to their root forms. This is done by mapping a group of words to the same stem even if the root word or the stem itself is not a valid word in the language [5-6]. Through *stemming*, the data set can be reduced as a lot of similar words would be clubbed into a single group under a common title of their root word. The image below explains how *stemming* works.
A list of words was obtained, where first the stopwords have been stripped, and second the words have been reduced to their root form, as visible in 'Out[12]', which represent the output (Figure 4).

STEMMING LEMMATIZATION

Figure 3: Difference between *Stemming* and *Lemmatization* [5].

```
In [14]: import nltk
         from nltk.stem import PorterStemmer # stemming
         from nltk.corpus import stopwords

In [15]: paragraph = "This paper describes a weakly supervised system for sentiment analysis in the movie review domain. The objective is

In [16]: sentences = nltk.sent_tokenize(paragraph)
         stemmer = PorterStemmer()

In [6]:  # Stemming
         for i in range(len(sentences)):
             words = nltk.word_tokenize(sentences[i])
             words = [stemmer.stem(word) for word in words if word not in set(stopwords.words('english'))]
             sentences[i] = ' '.join(words)

In [12]: sentences[4]

Out[12]: 'the propos system , wikis , requir label data train .'
```

Figure 4: Tokenization

2.1.3 Lemmatization with 'ntlk' package

Lemmatization, unlike *Stemming*, reduces the inflected words properly i.e. making sure that the root word is grammatically correct and belongs in the language. In Lemmatization, the root word is called Lemma whereas the root is called stem in Stemming. (Figure 4).
In Figure 5, we obtain a list of words from the sentence. Firstly, the stop-words have been stripped, and secondly, the words have been lemmatized to form meaningful root words. Therefore, after performing these steps, our data set would be reduced to a manageable size that could be passed through a query and would yield results fairly efficiently, making the program run faster and smoother.

2.2 Bag of Words

Natural Language Processing (NLP) doesn't directly work on text or special symbols. It can only process numbers. Therefore, we cannot directly feed our text into that algorithm. Here, Bag of Words model is useful because it converts the text into a *vector*, which helps keep a count of the total frequency of most commonly occurring used words. A bag-of-words represents text by giving us information about the most

frequently word in the document. It involves two things. 1. A vocabulary of known words. 2. A measure of the presence of known words. It is called a "*bag*" of words, because any irrelevant information about the location and structure of data in the document is discarded by preprocessing the data (Figure 6). The model is only interested in the frequency with which the text appears in the document, and not the location where it does.

```
In [1]: import nltk
        from nltk.stem import WordNetLemmatizer
        from nltk.corpus import stopwords

In [2]: paragraph = "This paper describes a weakly supervised system for sentiment analysis in the movie review domain. The objective is

In [3]: sentences = nltk.sent_tokenize(paragraph)
        lemmatizer = WordNetLemmatizer()

In [5]: # Lemmatization
        for i in range(len(sentences)):
            words = nltk.word_tokenize(sentences[i])
            words = [lemmatizer.lemmatize(word) for word in words if word not in set(stopwords.words('english'))]
            sentences[i] = ' '.join(words)

In [6]: sentences[4]

Out[6]: 'The proposed system , WikiSent , require labeled data training .'
```

Figure 5: List of words extracted from a sentence.

```
In [1]: import nltk

In [2]: paragraph = "This paper describes a weakly supervised system for sentiment analysis in the movie review domain. The objective is

In [3]: # Cleaning the texts
        import re
        from nltk.corpus import stopwords
        from nltk.stem.porter import PorterStemmer
        from nltk.stem import WordNetLemmatizer

In [4]: ps = PorterStemmer()
        wordnet =   WordNetLemmatizer()
        sentences = nltk.sent_tokenize(paragraph)

In [6]: corpus = []
        for i in range(len(sentences)):
            review = re.sub('[^a-zA-Z]', ' ', sentences[i])
            review = review.lower()
            review = review.split()
            review = [ps.stem(word) for word in review if not word in set(stopwords.words('english'))]
            review = ' '.join(review)
            corpus.append(review)
```

Figure 6: Preprocessing the data.

2.2.1 Steps to Prepare Bag of Words Model

The steps to prepare Bag of Words model are given hereunder.

Step 1: Remove all non-word characters and punctuations. ('.', '?', '!', '1', '$')
Step 2: Convert all the text to lowercase.
Step 3: Obtain the *bag of words* model by obtaining the most frequent words in our text.
Step 4: Convert this data into an array. Figure 7 gives the code for creating a Bag of Words model.

```
In [9]: # Creating the Bag of Words model
        from sklearn.feature_extraction.text import CountVectorizer
        cv = CountVectorizer(max_features = 1500)
        X = cv.fit_transform(corpus).toarray()
```

Figure 7*:* Preparation of Bag of Words Model

2.2.2 Problems associated with the Bag of Words Model

Although the Bag of Words model is useful in finding the most frequent words in the text and arranging it in an array, it cannot assign importance to different words. While working in a sentiment analysis program, certain words are more important than others, and it is crucial to assign this importance. To address this, *TF*IDF (Text Frequency - Inverse Document Frequency)* is utilized (Equation 1 and 2).

2.2.3 About *TF*IDF*

TF*IDF combines the measurement of how regularly a term is used on a page and the measurement of how often that term appears in all pages of a collection to assign the importance of that term to the page. A short example is given to illustrate the benefit of using TF*IDF for assigning importance to words [7-8].

Term Frequency (TF): We can count the number of times each term occurs in each document to distinguish a text document. The number of times a term occurs in a document is called its *term frequency,* estimated by Equation 1 [9].

$$tf(t,d) = \frac{Count\ of\ t\ in\ d}{Total\ Word\ Count\ of\ d} \quad (1)$$

t = Any word in the text
d = Text document

Inverse Document Frequency (IDF): Every term is considered equally important while determining the term frequency; however, it is known that specific terms, such as "is", "of", and "that", are present numerous times in our document but have little to no significance in determining the sentiment. Thus, frequently appearing terms are assigned a low score whereas rarely appearing terms are assigned a higher score. IDF helps assign this importance justifiably. When IDF is calculated, it will be very low for the most occurring words such as stop words (because stop words such as "is" is present in most of the documents, and N/df will give a low value to that word). This finally gives the relative weightage (Equation 2).

$$idf(t) = \frac{N}{df} \quad (2)$$

N = Number of sentences in the text document
df = Count of occurrences of term t in the document set N

Now there are few other problems with the IDF, in case of a large corpus, say 100,000,000, the IDF value explodes. To avoid this, we take its logarithm. This is known as *log normalization* (Equation 3).

$$idf(t) = log(\frac{N}{df+1}) \quad (3)$$

Final Formula Obtained:
Upon multiplying term frequency and inverse document frequency (Figure 8), we obtain Equation 4.

$$tf * idf(t,d) = tf(t,d) * log(\frac{N}{(df+1)}) \quad (4)$$

A decimal value is hence obtained for all the different words in the text, giving the importance of the words in the document. The higher is the value; more is the importance assigned to it. This helps solve the problem of the Bag of Words model, which was unable to give importance to different words.

```
In [7]: # Creating the TF-IDF model
        from sklearn.feature_extraction.text import TfidfVectorizer
        cv = TfidfVectorizer()
        X = cv.fit_transform(corpus).toarray()
```

Figure 8: Creating a TF*IDF model.

'TfidVectoriser' is a function, which is a part of the 'sklearn' library is used in creating the TF-IDF model (Figure 8).
For instance, if the corpus contains the following (Figure 9), the output generated with the use of *TF-IDF* would be as shown in Figure 10.

```
In [6]: corpus
Out[6]: ['paper describes weakly supervised system sentiment analysis movie review domain',
         'objective classify movie review polarity class positive negative based sentence bearing opinion movie alone leaving irrelevan
        t text',
```

Figure 9: An example showcasing TD-IDF in action.

2.3 Stock Sentiment Analysis

By feeding input and the target output, a model can be trained to predict future outputs whenever a new input is given. This is the utilization of *supervised learning.* Supervised learning is the machine learning task where the labeled data i.e. input and the output have already been provided, and based on this data the machine trains itself to predict future outputs.(Figure 11). It infers to a function from labelled training data consisting of a set of training examples. Using this approach, we can create a model to predict whether the prices of a stock will increase or decrease in the future based on the data fed to it in the past [10].

```
In [8]: X

Out[8]: array([[0.        , 0.        , 0.        , 0.        , 0.31114743,
                0.        , 0.        , 0.        , 0.        , 0.        ,
                0.        , 0.        , 0.        , 0.        , 0.        ,
                0.        , 0.37483764, 0.31114743, 0.        , 0.        ,
                0.        , 0.        , 0.        , 0.        , 0.        ,
                0.        , 0.        , 0.        , 0.        , 0.2022683 ,
                0.        , 0.        , 0.        , 0.        , 0.37483764,
                0.        , 0.        , 0.        , 0.        , 0.        ,
                0.        , 0.23090726, 0.        , 0.        , 0.        ,
                0.37483764, 0.        , 0.        , 0.31114743, 0.23090726,
                0.        , 0.        , 0.        , 0.        , 0.        ,
                0.        , 0.37483764, 0.        , 0.        , 0.        ],
```

Figure 10: TF-IDF output of a sentence.

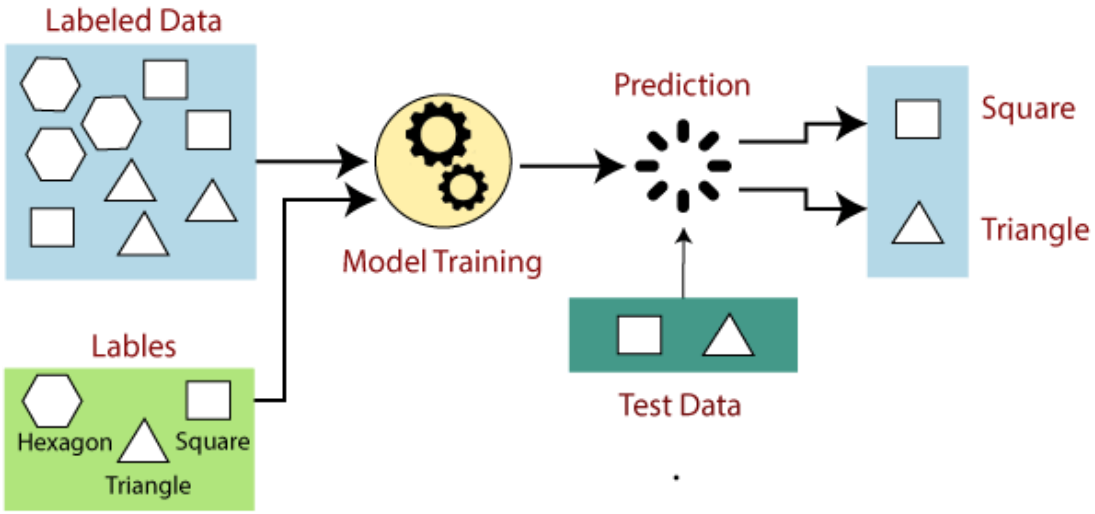

Figure 11: Flow chart depicting Supervised Learning.

Steps followed in stock sentiment analysis are given hereunder.

Step 1: Reading the Data: Firstly, a labelled set of data is required, which includes information about various stocks at different instances of time using a 'stock_analysis.csv' file that contains this information. This data is read by using the following code (Figure 12).

```
In [1]: import pandas as pd
        import nltk

In [2]: df=pd.read_csv('P:\\datasets\\stock_analysis.csv', encoding = "ISO-8859-1") # 8-bit single-byte coded graphic character sets
```

Figure 12*:* Reading the csv file.

Step 2: Segregating data b/w Training & Testing data: Second, the data is segregated into two different groups: a training set and a test set, respectively. The training data is used to train the model for predicting the output. After the model has

been trained, the test data is used to check the accuracy of the model. This code is divided based on the date when the tweet was published (Figure 13).

```
In [5]: train = df[df['Date'] < '20150101']
        test = df[df['Date'] > '20141231']
```

Figure 13: Segregating the data.

Step 3: Punctuation and Lower Case: The punctuation is removed, i.e. anything which is not a letter such as '#', '123','$' etc., is replaced with a space in the text (Figure 14). Then, the headlines are converted into lowercase (Figure 15) to prevent the discrepancy caused due to Python being Case sensitive and reading uppercase and lowercase words differently.

```
In [8]: # Removing punctuations
        data=train.iloc[:,2:27]
        data.replace("[^a-zA-Z]"," ",regex=True, inplace=True)

        # Renaming column names for ease of access
        list1= [i for i in range(25)]
        new_Index=[str(i) for i in list1]
        data.columns= new_Index
        data.head(5)
```

Figure 14: Removing punctuations.

```
In [9]: # Convertng headlines to lower case
        for index in new_Index:
            data[index]=data[index].str.lower()
        data.head(1)
```

Figure 15: Converting to lowercase.

Step 4: Implementing Bag of Words and Random Forest Classifier: Bag of words model is used to convert text to vectors as mentioned under section 2.2 using Random Forest Classifier (Figure 16) to train the model to predict the output. The Random Forest Classifier is a set of decision trees from a randomly selected subset of the training set. It sums the votes from different decision trees to decide the final class of the test object. Hence, it can train by providing input and output and obtaining a reasonably accurate model, which can predict the output of new future input.

Step 5: Predicting Output of Test data: After the model has been trained with the training set, the test data is given as input to the model and let it predict the output (Figure 17). Then, a comparison is made between the predicted output generated by the program with the labelled data already provided, determining the accuracy of the model.

```
In [13]: from sklearn.feature_extraction.text import CountVectorizer
         from sklearn.ensemble import RandomForestClassifier

In [14]: ## implement BAG OF WORDS
         countvector=CountVectorizer(ngram_range=(2,2))
         traindataset=countvector.fit_transform(headlines)

In [15]: # implement RandomForest Classifier
         randomclassifier=RandomForestClassifier(n_estimators=200,criterion='entropy')
         randomclassifier.fit(traindataset,train['Label'])
```

Figure 16: Implementing Bag of Words.

```
In [17]: ## Predict for the Test Dataset
         test_transform= []
         for row in range(0,len(test.index)):
             test_transform.append(' '.join(str(x) for x in test.iloc[row,2:27]))
         test_dataset = countvector.transform(test_transform)
         predictions = randomclassifier.predict(test_dataset)

In [21]: predictions # 1 indicates stock price will increse
                     # 0 indicates stock price  will decrease or stay same
```

Figure 17: Predicting Output of Test Data.

Step 6: Accuracy of the model: Using Sklearn, the accuracy of the model is inferred (Figure 18), which comes out as 86.24%, i.e. it will provide the correct output 86.24% of the time.

```
In [18]: ## Import library to check accuracy
         from sklearn.metrics import accuracy_score

In [19]: score=accuracy_score(test['Label'],predictions)
         print(score)

         0.8624338624338624
```

Figure 18: Accuracy.

2.4 Sentiment Analysis of Tweets on COVID-19

It is necessary to distinguish between positive, negative, and neutral sentiments, or even retrieve scores associated with a given opinion based only on text. There are two main approaches to teach an algorithm to distinguish between positive and negative emotions in writing. 1. Supervised Learning. 2. Unsupervised Learning. Unlike the stock sentiment model, there is no label data available for the tweets on COVID-19. Hence the problem is approached using unsupervised sentiment analysis. The main idea

behind unsupervised learning is not to give any previous assumptions and definitions to the model about the outcome of the variable fed into it. A pre-processed data is inserted for the model to learn the structure of the data itself. It is instrumental in cases to learn more about the nature of the process being analyzed, without making any previous assumptions about its outcome. The following flow chart shown in Figure 19 describes the approach taken to address this problem. Each process has been further elaborated.

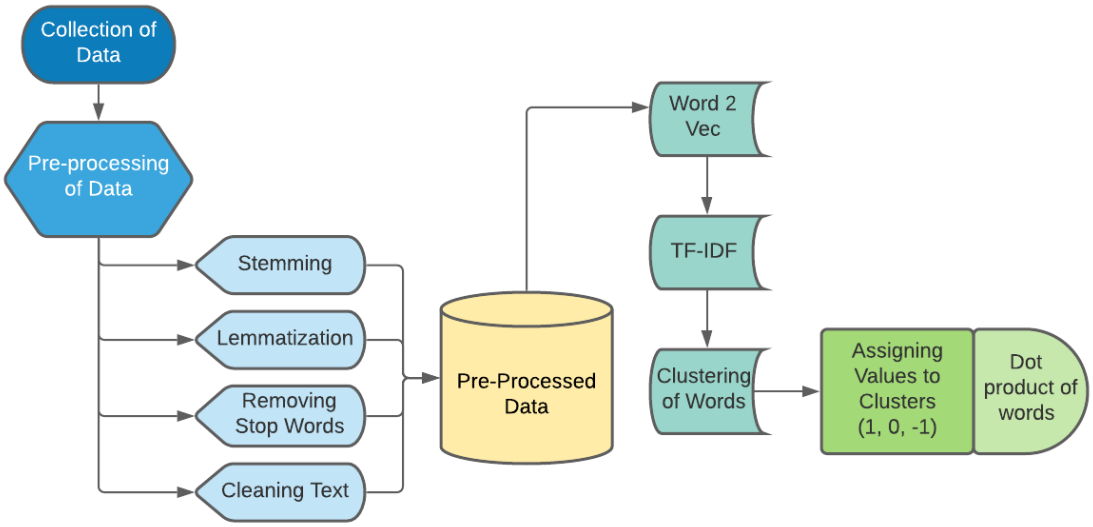

Figure 19: Flow Chart describing Approach.

The collection of data is firstly pre-processed through removing stop words (2.1.1), stemming (2.1.2), Lemmatization (2.1.3), removing stop words, and cleaning the text by removing punctuation to obtain clean data which contains only meaningful words. Then Word2vec algorithm is applied, which uses a neural network to learn the word associations and relations from a large corpus of text. TF-IDF is used to assign importance, and K-means clustering classifies the data into different groups based on their distance from the nearest mean. Then, each cluster is assigned a value of '1', '0', and '-1'. The steps are given hereunder.

Step 1: Reading the data from the csv file: The first step involves extracting the data from the csv files. This is done through the Pandas library (Figure 20). The CSV file used is 'covid19_tweets.csv', which is a file containing a large number of tweets and information about the tweets, including the username, the number of retweets or likes, etc.

```
In [1]: import pandas as pd
        import numpy as np

In [3]: df = pd.read_csv("P:\\datasets\\covid19_tweets.csv")
```

Figure 20: Reading the csv file.

Step 2: Pre-processing of Data: This step involves the cleaning of the text to reduce the volume of the text to a manageable amount, decreasing the amount of time taken for the model to run (Figure 21). Pre-processing of data has already been discussed under section 2.1.

```
In [8]: import re
        def clean_tweets(text):
            text = re.sub("RT @[\w]*:","",text)
            text = re.sub("@[\w]*","",text)
            text = re.sub("https?://[A-Za-z0-9./]*","",text)
            text = re.sub("\n","",text)
            return text

In [9]: df["text"] = df["text"].apply(lambda x : clean_tweets(x))
```

Figure 21: Cleaning the data.

Step 3: Word2Vec: After the pre-processed data has been obtained, the Word2Vec algorithm is used to learn relations between different words from a large document containing text. Once trained, such a model can detect synonymous words or suggest different words for a partial sentence. As the name implies, word2vec represents each specific word with a particular list of numbers called a vector [11].

Step 4: TF-IDF: TF-IDF is used for assigning importance to the different words in the text based on the frequency of their occurrence in the text. TF-IDF (Term Frequency - Inverse Document Frequency) has already been explained in detail under section 2.2.3.

Step 5: K-means Clustering: *K*-means clustering is a method of vector quantization that groups up all the observations into various different clusters in which each observation belongs to the cluster with the nearest mean (cluster centers or cluster centroid), serving as a prototype of the cluster. Then each of these clusters is assigned a value. In the case of sentiment analysis, tweets can be positive, negative or neutral, and these can be represented by '1', '-1' or '0' respectively.

Vader_Lexicon: VADER (Valence Aware Dictionary and Sentiment Reasoner) is a lexicon and rule-based sentiment analysis tool that is specifically attuned to sentiments expressed in social media, and works well on texts from other domains.
(Figure 22*)*. VADER is used to analyze the intensity of the sentiments, which returns a polarity score of the tweets that can have any value as a decimal. The close the value is to '1', the more positive the tweet is considered and vice versa.

```
In [11]: import nltk
         from nltk.sentiment.vader import SentimentIntensityAnalyzer
         nltk.download('vader_lexicon')

In [12]: sid = SentimentIntensityAnalyzer()

In [13]: df['sentiment']=df['text'].apply(lambda x: sid.polarity_scores(x))
```

Figure 22: Sentiment Analyzer.

3. RESULTS & DISCUSSIONS

3.1 Stock Sentiment Analysis

A model was developed using Python and successfully trained using supervised learning to predict the sentiment of news headlines of stocks with an accuracy of 86.24%. This highly accurate model has the following benefits.

1. Companies can predict which stock prices are going to increase, decrease, or stay the same and make their investments accordingly based on stock news.
2. This model can also be used to check the health of the company.
3. Companies can discover new marketing strategies and improve customer service as they can quickly identify the sentiments of consumers.

The accuracy of the model may be further increased by using other techniques.

3.2 Sentiment Analysis on COVID-19

3.2.1 Who is writing the tweet?

Since twitter is free media for sharing messages, these tweets can originate from any entity including a person, an organization, a cardinal, a geopolitical entity or maybe some other sources. Finding who is writing tweets can be useful as it can have many significant implications. The coding to assign the entities in a list is given in Figures 23, coding to divide the origin of tweets is given in Figure 24 and to represent the information in the form of a pie chart, coding is given in Figure 25.

The Python Library *'spacy',* which is an Advanced Natural Language Processing Library in Python is explicitly designed to build applications that process and 'understand' large volumes of text. It can be used to process text for deep or information extraction or learning to build natural language understanding systems. In Figure 24, five different entities have been created, namely, organization, person, geopolitical entity, norp (nationality or religious and political groups), and cardinal. The length of the lists of each group, i.e. the number of tweets made by each entity has been calculated. The 'matplotlib.pyplot' is used to obtain a pie chart given in Figure 26.

```
In [16]: import spacy
         nlp = spacy.load('en_core_web_sm')

In [17]: df['text'].apply(lambda x: [print("\tText : {}, Entity : {}"
                                    .format(ent.text, ent.label_)) if (not ent.text.startswith('#'))
                                      else ""  for ent in nlp(x).ents])

In [18]: df['entities']=df['text'].apply(lambda x: [(ent.text, ent.label_) if (not ent.text.startswith('#'))
                                                    else "" for ent in nlp(x).ents])

In [19]: entities_graph = []
         for i in df["entities"]:
             for j in i:
                 entities_graph.append(j)
```

Figure 23: Assigning entities in a list.

```
In [21]: organisation = []
         person = []
         geopolitical_entity = []
         norp = []
         cardinal = []
         for i in entities_graph:
             for j in i:
                 if j=="ORG":
                     organisation.append(j)
                 elif j=="PERSON":
                     person.append(j)

                 elif j=="GPE":
                     geopolitical_entity.append(j)
                 elif j=="NORP":
                     norp.append(j)

                 elif j=="CARDINAL":
                     cardinal.append(j)
                 else:
                     pass

In [22]: organisation = len(organisation)
         person = len(person)
         geopolitical_entity = len(geopolitical_entity)
         norp = len(norp)
         cardinal = len(cardinal)
         classes = [" organisation" , " person", " geopolitical_entity", "norp", "cardinal"]
```

Figure 24: Dividing the origin of tweets.

```
In [23]: import matplotlib.pyplot as plt

In [24]: graph = [organisation, person, geopolitical_entity, norp,cardinal]

In [25]: explode = [0,0.2,0.1,0,0]
         colors = ["c","b","r","g","y"]
         textprops = {"Fontsize":15}
         wedgeprops = {"linewidth": 4 , "width": 1 , "edgecolor" : "k"}
         plt.figure(figsize=(16,9))
         fontdict = {"Fontsize":25}
         plt.title("analysis of Entities", fontdict=fontdict)
         plt.pie(graph,labels = classes,explode = explode,colors = colors, autopct= "%0.2f%%",radius = 1,textprops = textprops,
                 pctdistance = 0.6,labeldistance = 1.1 ,wedgeprops = wedgeprops , rotatelabels=False )
         plt.legend(loc = 2)
         plt.savefig("entities_piechart" , dpi = 1000,quality =99)
         plt.show
```

Figure 25: Plotting Pie chart for analysis of entities

Upon running the following code given in Figure 25, analysis of entities are given in Figure 26. The source of the tweets have been categorized into five classes, and their percentage share is represented in Figure 26. Maximum tweets have originated from 'organizations'.

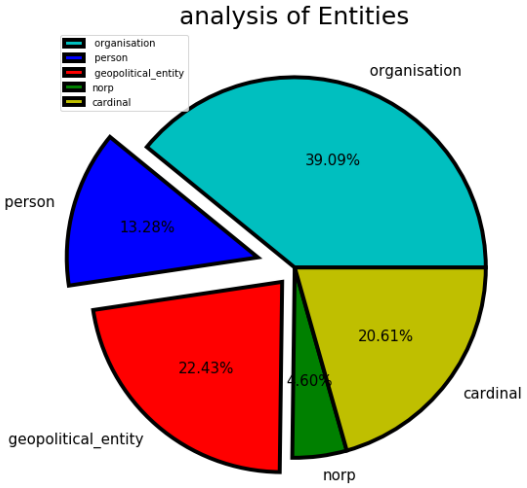

Figure 26*:* Analysis of entities in a chart.

3.2.2 Sentiment of Tweets

The tweets having a positive, negative, or neutral sentiment have already been determined. Here, the coding is done to display this information pictorially using *explode* library, to make a pie chart to display this data (Figures 27 and 28). The classification of the tweets in the three classes is 43.21% neutral, 38.79% negative, and 18% positive.

```
In [30]: a = []
         for i in df["sentiment"]:
             for key, value in i.items():
                 if value>0.5:
                     a.append(key)

In [32]: neutral = a.count("neu")
         positive = a.count("pos")
         negative = a.count("neg")

In [34]: classes2 = [ "neutral", "positive", "negative"]

In [35]: explode = [0,0.1,0]
         colors = ["c","b","r"]
         textprops = {"Fontsize":15}
         wedgeprops = {"linewidth": 4 , "width": 1 , "edgecolor" : "k"}
         plt.figure(figsize=(16,9))
         fontdict = {"Fontsize":25}
         plt.title("sentiment analysis of tweets", fontdict=fontdict)
         plt.pie([neutral,positive,negative], labels = classes2,explode = explode,colors = colors,
                 autopct= "%0.2f%%",radius = 1,textprops = textprops, pctdistance = 0.6,labeldistance = 1.1,
                 wedgeprops = wedgeprops , rotatelabels=False )
         plt.legend(loc = 4)
         plt.savefig("sentiment analysis pichart" , dpi = 1000,quality =99)
         plt.show()
```

Figure 27: Code for Plotting Sentiment of Tweets.

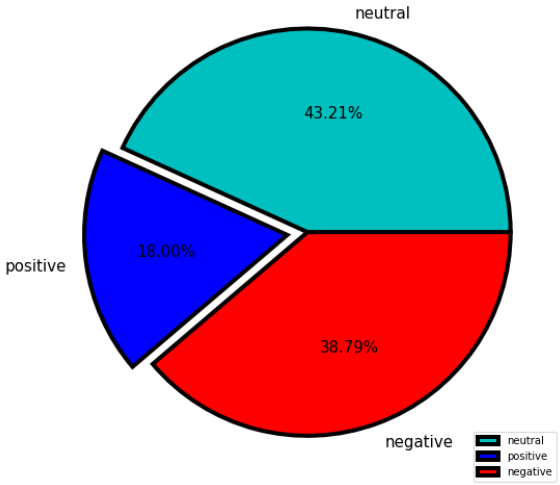

Figure 28: Pie Chart for Classification of Sentiments.

3.2.3 Sources of Tweets:

In order to obtain information about which device the users are writing their tweets from, codes shown in Figures 29 and 30 has been written. The source used to post tweets have been categorized among four categories, represented in Figure 30. The web platform is the most used source, followed by Android, iPhone, and Tweetdeck.

```
In [36]: source =[]
         for i in df["source"]:
             source.append(i)

In [38]: Twitter_for_iphone = source.count("Twitter for iPhone")
         twitter_for_android = source.count("Twitter for Android")
         twitter_web_app = source.count("Twitter Web App")
         tweetdect = source.count("TweetDeck")
         graph3 = [Twitter_for_iphone , twitter_for_android , twitter_web_app , tweetdect ]
         classes3 = ["Twitter for iphone","Twitter for Android", "twitter web app", "tweetdect" ]

In [39]: explode = [0,0.1,0,0]
         colors = ["c","b","r","y"]
         textprops = {"Fontsize":15}
         wedgeprops = {"linewidth": 4 , "width": 1 , "edgecolor" : "k"}
         plt.figure(figsize=(16,9))
         fontdict = {"Fontsize":25}
         plt.title("sources of Tweets", fontdict=fontdict)
         plt.pie(graph3, labels = classes3,explode = explode,colors = colors, autopct= "%0.2f%%",radius = 1,
                 textprops = textprops, pctdistance = 0.6,labeldistance = 1.1 ,wedgeprops = wedgeprops ,
                 rotatelabels=False )
         plt.legend(loc = 3)

         plt.savefig("source_piechart", dpi = 800,quality =50)
         plt.show()
```

Figure 29: Code for Sources of tweets.

The sentiments of tweets on COVID-19 using unsupervised learning has been done, which is also able to analyze the intensity of the sentiments of the tweets based on how positive, negative, or neutral they are. This data can have many benefits, such as:

1. Based on which entities (person, company, country, or an organization) the tweets come from, it may be identified that which entities have a problem with the current government of a country by analyzing the sentiments of these entities regarding the handling of COVID-19.

2. By using this data, organizations like WHO (World Health Organization) can come in picture with a helping hand for these entities and also get some help about publishing articles regarding these entities for a country.
3. By analyzing this data, the locations of the users or the organizations who are writing the tweets can be determined. The countries or regions may be analyzed based on the positive or negative sentiments on COVID-19.

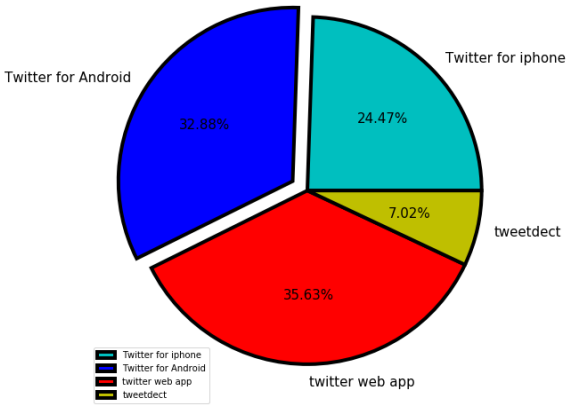

Figure 30: Sources of Tweets.

The information and data processing is the fuel for artificial intelligence. Based on the data analysis, decisions and policies may be framed by governments, agencies, organizations, and companies. The implementation of the coding samples and the analysis of the entities presented in this work is versatile and can be used in diversified applications.

4. CONCLUSION

A model has been successfully built with an accuracy of 86.24% for analyzing sentiments of headlines of stocks to predict the increase or decrease in stocks. Demonstration of the model has been successfully done by analyzing the tweets on COVID-19 to gather information, which can have many beneficial uses for companies, businesses, organizations, and even countries. The outcomes of the study have opened new avenues for data analysis using Python, which can fuel the artificial intelligence algorithms.

Author Contributions: Conceptualization, methodology, validation, formal analysis, investigation, data curation, Yuvraj Jain; Mentorship, formatting, writing and drafting, review, Vineet Tirth.

Acknowledgements: The first author thankfully acknowledges the guidance, motivation and support provided by the teachers of K.R Mangalam World School, New Delhi, India.

Conflicts of Interest: The authors declare no conflict of interest.

REFERENCES

[1] K. H. Manguri, R. N. Ramadhan, and P. R. Mohammed Amin, "Twitter Sentiment Analysis on Worldwide COVID-19 Outbreaks," *Kurdistan J. Appl. Res.*, no. May, pp. 54–65, 2020.
[2] A. D. Dubey, "Twitter Sentiment Analysis during COVID19 Outbreak," pp. 1-9.
[3] K. Sailunaz and R. Alhajj, "Emotion and sentiment analysis from Twitter text," *J. Comput. Sci.*, vol. 36, p. 101003, 2019.
[4] V. Wisdom, R. Gupta, "An Introduction to Data Analysis in Python," pp. 1-5, 2002.
[5] Stemming and Lemmatization in Python. H Jabeen - DataCamp, October, 2018. Accessed by: https://www.datacamp.com/community/tutorials/stemming-lemmatization-python
[6] Stemming and Lemmatization. https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html
[7] J. Martineau *et al.*, "Delta TFIDF: An Improved Feature Space for Sentiment Analysis," *Proc. Second Int. Conf. Weblogs Soc. Media (ICWSM*, vol. 29, no. May, pp. 490–497, 2008.
[8] TF*IDF Equation Explained. https://forum.seo-autopilot.eu/printthread.php?tid=122
[9] Y. Hamdaoui, "TF(Term Frequency)-IDF(Inverse Document Frequency) from scratch in python,". https://towardsdatascience.com/tf-term-frequency-idf-inverse-document-frequency-from-scratch-in-python-6c2b61b78558
[10] T. H. Nguyen, K. Shirai, and J. Velcin, "Sentiment analysis on social media for stock movement prediction," *Expert Syst. Appl.*, vol. 42, no. 24, pp. 9603–9611, 2015.
[11] A. Rexha, M. Kröll, M. Dragoni, R. Kern, "Polarity Classification for Target Phrases in Tweets: A Word2Vec Approach," *European Semantic Web Conference,* pp. 217-223, 2016. https://doi.org/10.1007/978-3-319-47602-5_40.