

COVID TWEET ANALYSIS USING NLP

Submitted in partial fulfillment of the requirements for
the award of
Bachelor of Engineering degree in Computer Science and Engineering
by

**RAVURI JYOTHI SAI KRISHNA
J.VENKAT VINAY KISHAN**

(Reg. No. 38110456,38110209)



**DEPARTMENT OF COMPUTER SCIENCE AND
ENGINEERING SCHOOL OF COMPUTING**

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

**Accredited with Grade "A" by NAAC
JEPPIAAR NAGAR, RAJIV GANDHI
SALAI, CHENNAI 600 119**

MARCH - 2022



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited with "A" grade by NAAC

Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai – 600 119

www.sathyabama.ac.in



DEPARTMENT OF COMPUTER SCIENCE AND

ENGINEERING BONAFIDE CERTIFICATE

This is to certify that this project report is the bonafide work of **RAVURI JYOTHI SAI KRISHNA, J.V. VINAY KISHAN (Reg.No.38110456, 38110209)** project who carried out the project entitled "**COVID TWEET ANALYSIS USING NLP**" under my supervision from November 2021 to March 2022.

Internal Guide

Dr. T. ANANDHI, M.E.(Ph.D.),

Head of the Department

Dr. S. Vigneswari, M.E., Ph.D.,

Submitted for Viva voce Examination held on

Internal Examiner

External Examiner

DECLARATION

I **RAVURI JYOTHI SAI KRISHNA,J.VENKAT VINAY KISHAN** hereby declare that the Project Report entitled “**COVID TWEET ANALYSIS USING NLP**” is done by me under the guidance of **Dr.T. ANANDHI, M.E.(Ph.D.)**, Department of Computer Science and Engineering at Sathyabama Institute of Science and Technology is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in Computer Science and Engineering.

R.JYOTHI SAI KRISHNA

DATE:
PLACE: CHENNAI

J.VENKAT VINAY KISHAN
SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management of SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sasikala, M.E., Ph.D., Dean**, School of Computing, **Dr. S. Vigneswari, M.E., Ph.D., and Dr. L. Lakshmanan, M.E., Ph.D., Heads of the Department** of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. T. ANANDHI, M.E.(Ph.D.), Assistant Professor**, for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

Abstract

Covid or COVID-19 originally showed up in December 2019 in Wuhan, China. Individuals tweeted forcefully on twitter around then. This task investigated the tweets with respect to COVID-19 in India and its effect. All tweets are classified into categories (Positive, Negative and Neutral). Twitter online media stage, considering the principal patterns with Natural Language Processing and with Sentiment Classification utilizing Recurrent Neural Network. Where we break down, assemble, picture insights, and sum up for additional handling the opinion order and grouped them sure, negative and nonpartisan feeling scores. We have additionally thought about the exhibition of the proposed calculation on specific boundaries like accuracy, review and precision with Recurrent Neural Network (RNN) with LSTM, then, at that point, we explored different avenues regarding distinctive retinal highlights as contribution to RNN for powerful order of Corona virus tweets.

TABLE OF CONTENTS

CHAPTER No	TITLE	PAGE No
	ABSTRACT	iv
	LIST OF FIGURES	vii
1.	INTRODUCTION	1
	1.1 INTRODUCTION	1
	1.2 OUTLINE OF THE PROJECT	2
	1.3 PROBLEM IN EXISTING SYSTEM	3
2.	LITERATURE SURVEY	4
3.	SYSTEM DESIGN	9
	3.1 PYTHON	9
	3.2 ANACONDA NAVIGATOR	10
	3.3 PROJECT REQUIREMENTS	17
	3.4 FEASIBILITY STUDY	18
	3.5 SYSTEM ARCHITECTURE	19
	3.6 WORKFLOW DIAGRAM	20
	3.7 DESIGN ARCHITECTURE	20
	3.8 ENVIRONMENT REQUIREMENTS	25
	3.9 METHODOLOGY	26
	3.10 LIST OF MODULES	38

	3.11 COVID TWEET CLASSIFICATION	48
	3.12 TENSOR FLOW	49
4.	RESULTS AND DISCUSSION	52
	4.1 EXPERIMENTAL RESULTS	52
	4.2 DISCUSSION	53
5.	CONCLUSION AND FUTURE WORK	56
	5.1 CONCLUSION	56
	5.2 FUTURE WORK	56

LIST OF FIGURES

FIGURE No.	FIGURE NAME	PAGE No.
1.2	OUTLINE	2
3.2	NAVIGATOR	11
3.2	NAVIGATOR INSTALL	12
3.5	ARCHITECTURE	19
3.6	FLOW DIAGRAM	20
3.7	USECASE DIAGRAM	21
3.7	CLASS DIAGRAM	22
3.7	ACTIVITY DIAGRAM	22
3.7	SEQUENCE DIAGRAM	23
3.7	ER DIAGRAM	24
3.7	COLLABORATION DIAGRAM	25
3.9	ANN	31
3.9	LSTM	34
3.11	COVID TWEET CLASSIFICATION	48
	OUTPUT SCREENSHOTS	74

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

Natural Language Processing (NLP):

Natural language processing(NLP) allows machines to read and understand human language. A sufficiently powerful natural language processing system would enable natural-language user interfaces and the acquisition of knowledge directly from human-written sources, such as newswire texts. Some straightforward applications of natural language processing include information retrieval, text mining, question Answering and machine translation. Many current approaches use word co occurrence frequencies to construct syntactic representations of text. “Keyword spotting” strategies for search are popular and scalable but dumb; a search query for “dog” might only match documents with the literal word “dog” and miss a document with the word “poodle”. “Lexical affinity” strategies use the occurrence of words such as “accident” to assess the sentiment of a document. Modern statistical NLP approaches can combine all these strategies as well as others, and often achieve acceptable accuracy at the page or paragraph level. Beyond semantic NLP, the ultimate goal of “narrative” NLP is to embody a full understanding of commonsense reasoning. By 2019, transformer-based deep learning architectures could generate coherent text.

DEEP LEARNING:

Deep learning is a branch of machine learning which is completely based on artificial neural networks, as neural network is going to mimic the human brain so deep learning is also a kind of mimic of human brain. It’s on hype nowadays because earlier we did not have that much processing power and a lot of data. A formal definition of deep learning is- neurons Deep learning is a particular kind of machine learning that achieves great power and flexibility by learning to represent the world as a nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones. In brain approximately 100 billion neurons all together this is a

picture of an individual neuron and each neuron is connected through thousands of their neighbors. The question here is how it recreates these neurons in a computer. So, it creates an artificial structure called an artificial neural net where we have nodes or neurons. It has some neurons for input value and some for output value and in between, there may be lots of neurons interconnected in the hidden layer.

1.2 OUTLINE OF THE PROJECT:

It need to identify the actual problem in order to get the right solution and it should be understood, the feasibility of the Deep Learning should also be checked (whether it should fit Deep Learning or not). It needs to identify the relevant data which should correspond to the actual problem and should be prepared accordingly. Choose the Deep Learning Algorithm appropriately. Algorithm should be used while training the dataset. Final testing should be done on the dataset.

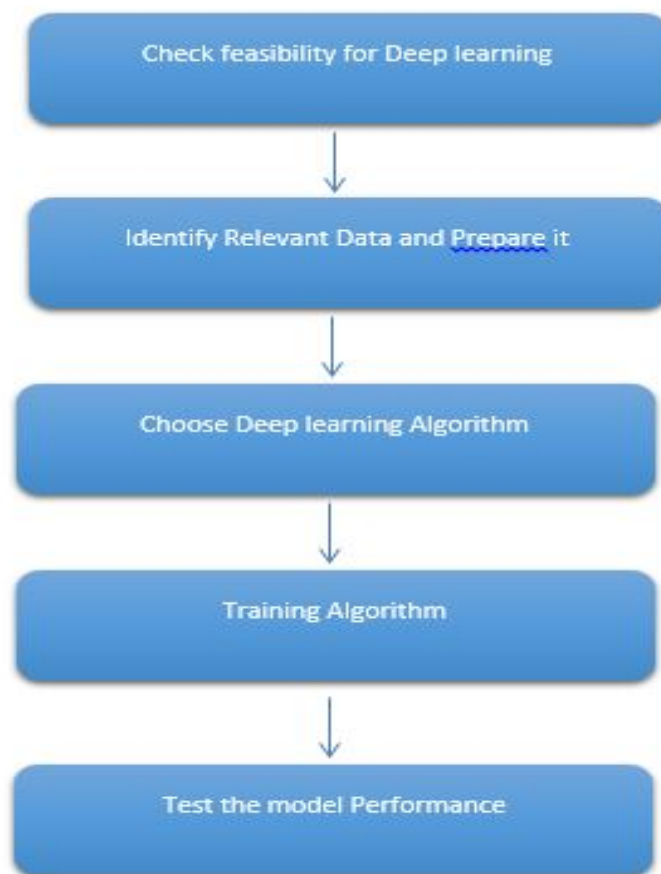


Fig 1.2.1.OUTLINE

1.3 PROBLEM IN EXISTING SYSTEM:

They proposed for the issue of Twitter sentiment on COVID-19-related Twitter posts. We benchmark sentiment analysis methods in the analysis of COVID-19-related sentiment. This gives rise to the need to create analytic methods that could be rapidly deployed to understand information flows and to interpret how mass sentiment among the population develops in pandemic scenarios. They tweets has been labeled into positive, negative, and neutral sentiment classes. We analyzed the collected tweets for sentiment classification using different sets of features and classifiers. Negative opinion played an important role in conditioning public sentiment, for instance, we observed that people favored lockdown earlier in the pandemic.

CHAPTER 2

LITERATURE SURVEY

2. LITERATURE SURVEY:

ZIJIAN LONG , RAJWA ALHARTHI, ABDULMOTALEB EL SADDIK et.al [1] in his paper on "**A Tweet Analysis Platform to study Human Needs during the COVID-19 pandemic in New York State**". Governments and municipalities need to understand their citizens' psychological needs in critical times and dangerous situations. COVID-19 brings lots of challenges to deal with. We propose Need Full, an interactive and scalable tweet analysis platform, to help governments and municipalities to understand residents' real psychological needs during those periods. The platform mainly consists of four parts: data collection module, data storage module, data analysis module and data visualization module. In this paper, we propose Need Full, an interactive and scalable tweet analysis platform, which mainly consists of four main parts: data collection module, data storage module, data analysis module and data visualization module. The four parts interact with each other and provide users with a thorough human needs analysis based on their queries. This awareness of people's affects is a crucial step for governments and municipalities to understand their citizens' psychological needs especially in critical times and dangerous situations. However, the human need detection model we employed can only analyze text contents. That is also the reason we cannot extend our platform to Instagram where people only share images. For future work, we plan to extend the platform with human needs analysis of other social media contents such as image and video

Mabrook S. Al-Rakhami, Atif M. Al-Amr et.al [2] in his paper on "**Detecting COVID-19 Misinformation in Twitter**" Online social networks (ONSs) such as Twitter have grown to be very useful tools for the dissemination of information. However, they have also become a fertile ground for the spread of false

information, particularly regarding the ongoing coronavirus disease 2019 (COVID-19) pandemic. Best described as an infodemic, there is a great need, now more than ever, for scientific fact-checking and misinformation detection regarding the dangers posed by these tools with regards to COVID-19. In this paper, we analyze the credibility of information shared on Twitter pertaining the COVID-19 pandemic. For our analysis, we propose an ensemble-learning-based framework for verifying the credibility of a vast number of tweets. In particular, we carry out analyses of a large dataset of tweets conveying information regarding COVID-19. The ongoing COVID-19 pandemic is a threat to human beings. Unlike other global challenges, such as global warming, containing and defeating COVID-19 will depend much on the quality and credibility of information shared amongst people. However, research has shown that misinformation has spread rapidly on OSNs regarding the pandemic. In this work, we conducted a comprehensive experiment using real data from the Twitter social network. The results revealed that the proposed ensemble-learning model had better performance than single machine-learning-based models. We enhanced the performance of our stacking model by assessing meta-models and weak-learners. We concluded that the final model size can contain fewer features, and it performed slightly better than the original model. As of now, our model has been designed to detect two categories of tweet credibility: credible or non-credible. We acknowledge that there is room for improvement, part of which shall include the following. First is the incorporation of complex tweets containing news and emotional content. Additionally, we shall take into consideration other OSNs that will help us enhance our dataset. Secondly, we shall look out for updated ensemble techniques and machine-learning methods to enhance the current model

Manoj Sethi, Sarthak Pandey, et.al [3] in her paper on “**Sentiment Identification in COVID-19 Specific Tweets**”. In 2020, our world has been hit by a global pandemic of COVID-19, belonging to the family of Corona virus. Due to the rapid increase in the infection and the death rate, people have started to develop mixed feelings regarding this situation. Therefore, in this study, our sole focus is to analyze the emotions expressed by people using social media such as Twitter etc.

Accumulating and studying the concerning tweets will provide aid to elicitate the real emotions during this hard time. The goal of this study is to present a domain-specific approach to understand sentiments manifested within people around the globe regarding this situation. In order to attain this, corona specific tweets are acquired from twitter platform. After gathering the tweets, they are labelled and a model is developed which is effective for detecting the actual sentiment behind a tweet related to COVID-19. The study in this paper was done with the goal of creating a model which can effectively predict the sentiment expressed by people on social media platforms amidst this COVID-19 pandemic. For this study, datasets are manually created by gathering tweets from twitter API using hashtags #COVID19 and #corona virus. From the experiments performed in this study, it is concluded that both SVM and Decision Tree has performed extremely well but the SVM classifier was the more robust and consistent throughout all the experiments. classifiers performed better in bi-class setting than in multi class setting. In bi-class setting, XGBoost performed best with maximum accuracy of 82%, whereas in multi-class setting, Random Forest performed better having maximum accuracy of about 46%. On comparing the overall performance of both the binary class and multi-class setting, it has been concluded that binary-class setting is more suitable as it performs good consistently in all 3 datasets along with much better performance in cross-dataset evaluation.

A. Mourad, A. Srour et.al [4] in his paper on “**Critical Impact of Social Networks Infodemic on Defeating Coronavirus COVID-19 Pandemic: Twitter-Based Study and Research Directions**”. News creation and consumption has been changing since the advent of social media. An estimated 2.95 billion people in 2019 used social media worldwide. The widespread of the Coronavirus COVID-19 resulted with a tsunami of social media. Most platforms were used to transmit relevant news, guidelines and precautions to people. According to WHO, uncontrolled conspiracy theories and propaganda are spreading faster than the COVID-19 pandemic itself, creating an infodemic and thus causing psychological panic, misleading medical advises, and economic disruption. Accordingly, discussions have been initiated with the objective of moderating all COVID-19’s communications, except those initiated from trusted sources such as the WHO

and authorized governmental entities. This paper presents a large-scale study based on data mined from Twitter. Extensive analysis has been performed on approximately one million COVID-19 related tweets collected over a period of two months. Furthermore, the profiles of 288, 000 users were analyzed including unique users' profiles, meta-data and tweets' context. The study noted various interesting conclusions including the critical impact in term of reach level of the (1) exploitation of the COVID-19 crisis to redirect readers to irrelevant topics and (2) widespread of unauthentic medical precautions and information. That paper investigated the COVID-19 infodemic negative impact on the major efforts to defeat the pandemic through a novel large-scale Twitter-based study, which provided quantitative assessment using real-life experiments reflecting the actual environments. The empirical analysis of 1 million COVID-19-related tweets belonging to 288K unique users illustrated the severe impact of misleading people and spreading unreliable information. Inferred insights showed that (1) the potential reachability of the 16.1% none relevant tweets that might or might not be misled users by redirecting them to out of scope and/or malicious content is 5.6 billion, and (2) a minimum of 93.7% of the remaining within-context 83.9% tweets (i.e. with around 17M Interactions and 30B Reach counts) were initiated by users with non-reliable medical and/or relevant specialty profiles, and consequently might be disseminating misleading non-credible medical information. Moreover, different insights highlighted the low reachability (i.e. 1% of the total Reach counts, which is equivalent 300M out of 30B) of the unique users with key context-relevant specialties and occupations such as doctor, writer, reporter, journalist, editor and governor. As previously explored, the number of tweets initiated by users claimed to be "Doctors" having credible medical profiles is really low. The results shed the light on the importance of identifying non-medical key influencers for assisting in spreading legitimate information relevant in such situations. Finally, the paper elaborated on few computing and non-computing implications as well as future research directions to highlight the potential solutions and future work in such a promising field.

TING DA, LIANG YANG et.al [5] in his paper on "**Local COVID-19 Severity and Social Media Responses: Evidence From China**".Unexpected but exceedingly

consequential, the COVID-19 outbreak has undermined livelihoods, disrupted the economy, induced upheavals, and posed challenges to government decision-makers. Under various behavioural regulations, such as social distancing and transport limitations, social media has become the central platform on which people from all regions, regardless of local COVID-19 severity, share their feelings and exchange thoughts. Our study illustrates the evolution of moods expressed on social media regarding COVID-19-related issues and empirically confirms the hypothesis that the severity of the pandemic substantially correlates with these sentiments by analysing tweets on Sina Weibo (China's central social media platform). Methodologically, we leveraged Sentiment Knowledge Enhanced Pre-training, the most state-of-the-art natural language processing pre-trained sentiment-related multipurpose model, to label Sina Weibo tweets during the most distressed period in 2020. Given that the model itself does not provide a feature explanation, we utilize a random forest and linear probit model with the labelled sample to demonstrate how each word plays a role in the prediction. applied the most state-of-the-art pre-trained sentiment classification model, SKEP, to classify COVID-19- related social media tweets in China. Because SKEP itself does not provide a feature explanation, we implemented an RF followed by an LP model with SKEP prediction as the response variable, and we confirmed the SKEP classification. The visualization of the RF and LP models shows that COVID-19-related words and common sentiment words play a substantial role in driving the prediction. Finally, we aggregated sentiment classification from the tweet level to the province level for all COVID-19-related tweets with recognizable GPS information. Univariate and multivariate linear regression results corroborate the hypothesis that more severely affected regions tend to have a greater share of pessimistic moods. In fact, confirmed COVID-19 cases alone could explain approximately 68% of the variation in sentiment across provinces. Technically, to further improve the explanatory power of our RF and LP models, one could design and perform even finer text pre processing to filter out additional noise. Future works could be devoted to improving the pre-trained SKEP model so that it simultaneously makes predictions and explains feature roles. In terms of drawing a broader socioeconomic conclusion, one could also apply SKEP with an RF and LP model on COVID-19-related tweets using data from other regions around the world.

CHAPTER 3

SYSTEM DESIGN

3. SYSTEM DESIGN

3.1 PYTHON:

Python is an interpreted high-level general-purpose programming language. Its design philosophy emphasizes code readability with its use of significant indentation. Its language constructs as well as its object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects. Python is dynamically-typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented and functional programming. It is often described as a "batteries included" language due to its comprehensive standard library. Guido van Rossum began working on Python in the late 1980s, as a successor to the ABC programming language, and first released it in 1991 as Python 0.9.0. Python 2.0 was released in 2000 and introduced new features, such as list comprehensions and a garbage collection system using reference counting. Python 3.0 was released in 2008 and was a major revision of the language that is not completely backward-compatible. Python 2 was discontinued with version 2.7.18 in 2020. Python consistently ranks as one of the most popular programming languages

History

Python was conceived in the late 1980s by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to ABC programming language, which was inspired by SETL, capable of exception handling and interfacing with the Amoeba operating system. Its implementation began in December 1989. Van Rossum shouldered sole responsibility for the project, as the lead developer, until 12 July 2018, when he announced his "permanent vacation" from his responsibilities as Python's Benevolent Dictator For Life, a title the Python community bestowed upon him to reflect his long-term commitment as the project's chief decision-maker. In January 2019, active Python

core developers elected a 5-member "Steering Council" to lead the project. As of 2021, the current members of this council are Barry Warsaw, Brett Cannon, Carol Willing, Thomas Wouters, and Pablo Galindo Salgado. Python 2.0 was released on 16 October 2000, with many major new features, including a cycle-detecting garbage collector and support for Unicode. Python 3.0 was released on 3 December 2008. It was a major revision of the language that is not completely backward-compatible. Many of its major features were backported to Python 2.6.x and 2.7.x version series. Releases of Python 3 include the 2 to 3 utility, which automates (at least partially) the translation of Python 2 code to Python 3. Python 2.7's end-of-life date was initially set at 2015 then postponed to 2020 out of concern that a large body of existing code could not easily be forward-ported to Python 3. No more security patches or other improvements will be released for it. With Python 2's end-of-life, only Python 3.6.x and later are supported. Python 3.9.2 and 3.8.8 were expedited as all versions of Python (including 2.7) had security issues, leading to possible remote code execution and web cache poisoning.

3.2 ANACONDA NAVIGATOR:

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution that allows you to launch applications and easily manage conda packages, environments, and channels without using command-line commands. Navigator can search for packages on Anaconda.org or in a local Anaconda Repository. Anaconda. Now, if you are primarily doing data science work, Anaconda is also a great option. Anaconda is created by Continuum Analytics, and it is a Python distribution that comes preinstalled with lots of useful python libraries for data science. Anaconda is a distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. In order to run, many scientific packages depend on specific versions of other packages. Data scientists often use multiple versions of many packages and use multiple environments to separate these different versions.

The command-line program conda is both a package manager and an environment manager. This helps data scientists ensure that each version of each package has all the dependencies it requires and works correctly.

Navigator is an easy, point-and-click way to work with packages and environments without needing to type conda commands in a terminal window. You can use it to find the packages you want, install them in an environment, run the packages, and update them – all inside Navigator.

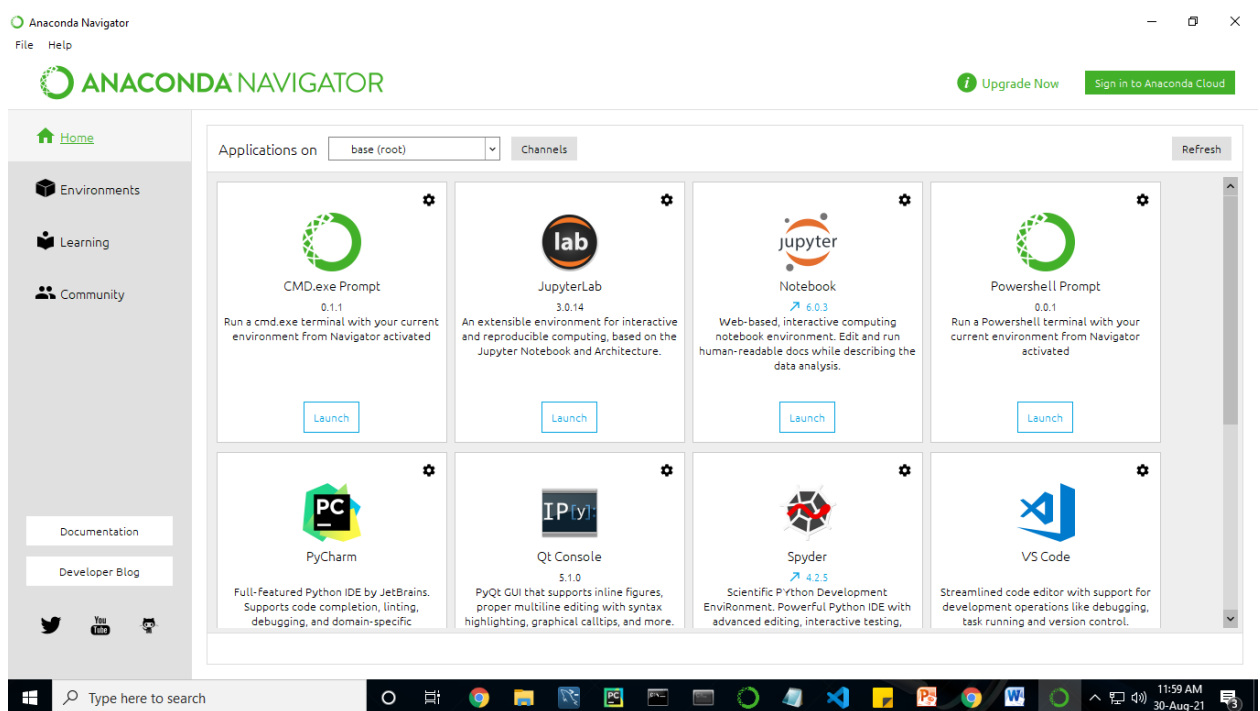


Fig 3.2.1. Navigator

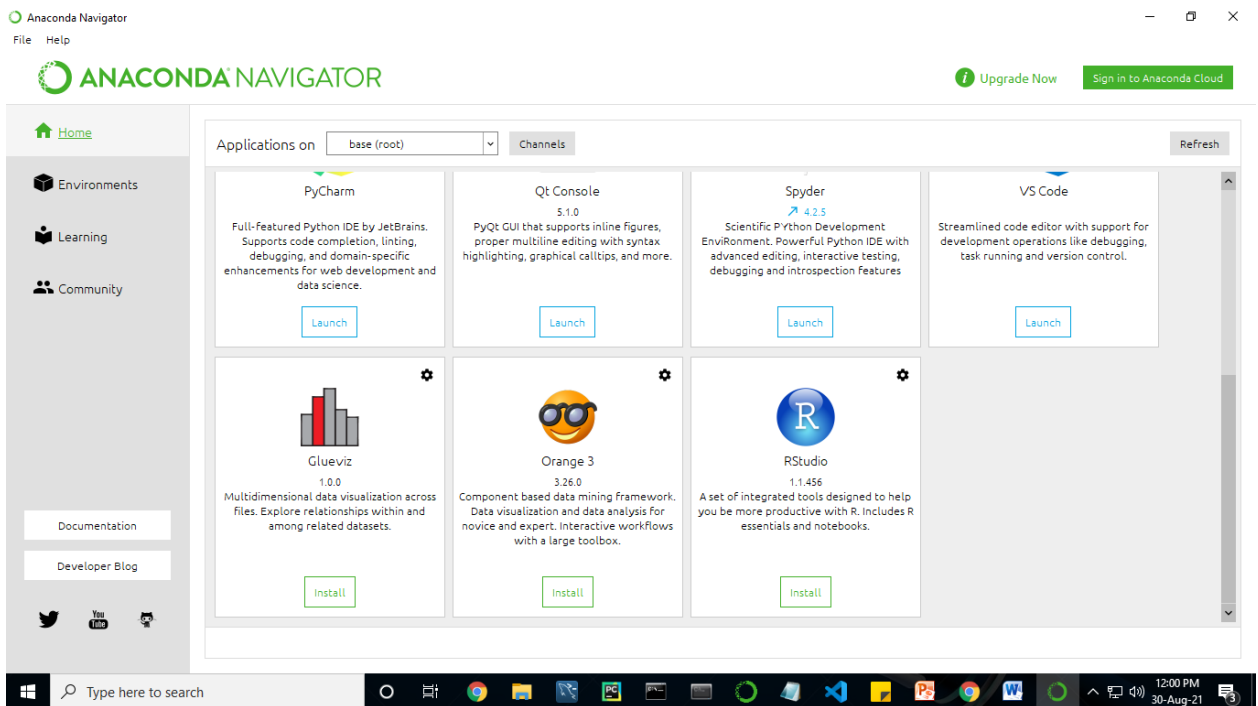


Fig 3.2.2.Navigator Install

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution.

Navigator allows you to launch common Python programs and easily manage conda packages, environments, and channels without using command-line commands. Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository.

Anaconda comes with many built-in packages that you can easily find with `conda list` on your anaconda prompt. As it has lots of packages (many of which are rarely used), it requires lots of space and time as well. If you have enough space, time and do not want to burden yourself to install small utilities like JSON, YAML, you better go for Anaconda.

Conda :

Conda is an open source, cross-platform, language-agnostic package manager and environment management system that installs, runs, and updates packages and their dependencies. It was created for Python programs, but it can

package and distribute software for any language (e.g., R), including multi-language projects. The conda package and environment manager is included in all versions of Anaconda, Miniconda, and Anaconda Repository.

Anaconda is freely available, open source distribution of python and R programming languages which is used for scientific computations. If you are doing any machine learning or deep learning project then this is the best place for you. It consists of many softwares which will help you to build your machine learning project and deep learning project. these softwares have great graphical user interface and these will make your work easy to do. you can also use it to run your python script. These are the software carried by anaconda navigator.

JUPYTER NOTEBOOK:

This website acts as “meta” documentation for the Jupyter ecosystem. It has a collection of resources to navigate the tools and communities in this ecosystem, and to help you get started.

Project Jupyter is a project and community whose goal is to "develop open-source software, open-standards, and services for interactive computing across dozens of programming languages". It was spun off from IPython in 2014 by Fernando Perez.

Notebook documents are documents produced by the Jupyter Notebook App, which contain both computer code (e.g. python) and rich text elements (paragraph, equations, figures, links, etc...). Notebook documents are both human-readable documents containing the analysis description and the results (figures, tables, etc.) as well as executable documents which can be run to perform data analysis.

Installation: The easiest way to install the *Jupyter Notebook App* is installing a scientific python distribution which also includes scientific python packages. The most common distribution is called Anaconda

Running the Jupyter Notebook Launching *Jupyter Notebook App*: The Jupyter Notebook App can be launched by clicking on the *Jupyter Notebook* icon installed by Anaconda in the start menu (Windows) or by typing in a terminal (*cmd* on Windows): “jupyter notebook”

This will launch a new browser window (or a new tab) showing the Notebook Dashboard, a sort of control panel that allows (among other things) to select which notebook to open.

When started, the Jupyter Notebook App can access only files within its start-up folder (including any sub-folder). No configuration is necessary if you place your notebooks in your home folder or subfolders. Otherwise, you need to choose a Jupyter Notebook App start-up folder which will contain all the notebooks.

Save notebooks: Modifications to the notebooks are automatically saved every few minutes. To avoid modifying the original notebook, make a copy of the notebook document (menu file -> make a copy...) and save the modifications on the copy.

Executing a notebook: Download the notebook you want to execute and put it in your notebook folder (or a sub-folder of it).

Launch the jupyter notebook app

- In the Notebook Dashboard navigate to find the notebook: clicking on its name will open it in a new browser tab.
- Click on the menu *Help -> User Interface Tour* for an overview of the Jupyter Notebook App user interface.
- You can run the notebook document step-by-step (one cell a time) by pressing *shift + enter*.
- You can run the whole notebook in a single step by clicking on the menu *Cell -> Run All*.
- To restart the kernel (i.e. the computational engine), click on the menu *Kernel -> Restart*. This can be useful to start over a computation from scratch (e.g. variables are deleted, open files are closed, etc...).

Purpose: To support interactive data science and scientific computing across all programming languages.

File Extension: An **IPYNB** file is a notebook document created by Jupyter Notebook, an interactive computational environment that helps scientists manipulate and analyze data using Python.

JUPYTER Notebook App: The *Jupyter Notebook App* is a server-client application that allows editing and running notebook documents via a web browser. The *Jupyter Notebook App* can be executed on a local desktop requiring no internet access (as described in this document) or can be installed on a remote server and accessed through the internet.

In addition to displaying/editing/running notebook documents, the *Jupyter Notebook App* has a “Dashboard” (Notebook Dashboard), a “control panel” showing local files and allowing to open notebook documents or shutting down their kernels.

kernel: A notebook *kernel* is a “computational engine” that executes the code contained in a Notebook document. The ipython kernel, referenced in this guide, executes python code. Kernels for many other languages exist. When you open a Notebook document, the associated *kernel* is automatically launched. When the notebook is *executed* (either cell-by-cell or with menu *Cell -> Run All*), the *kernel* performs the computation and produces the results. Depending on the type of computations, the *kernel* may consume significant CPU and RAM. Note that the RAM is not released until the *kernel* is shut-down

Notebook Dashboard: The *Notebook Dashboard* is the component which is shown first when you launch Jupyter Notebook App. The *Notebook Dashboard* is mainly used to open notebook documents, and to manage the running kernels (visualize and shutdown).

The *Notebook Dashboard* has other features similar to a file manager, namely navigating folders and renaming/deleting files

Working Process:

Download and install anaconda and get the most useful package for machine learning in Python.

Load a dataset and understand its structure using statistical summaries and data visualization.

Machine learning models, pick the best and build confidence that the accuracy is reliable.

Python is a popular and powerful interpreted language. Unlike R, Python is a complete language and platform that you can use for both research and development and developing production systems. There are also a lot of modules and libraries to choose from, providing multiple ways to do each task. It can feel overwhelming.

The best way to get started using Python for machine learning is to complete a project.

It will force you to install and start the Python interpreter (at the very least).

- It will give you a bird's eye view of how to step through a small project.
- It will give you confidence, maybe to go on to your own small projects.

When you are applying machine learning to your own datasets, you are working on a project. A machine learning project may not be linear, but it has a number of well-known steps:

- Define Problem.
- Prepare Data.
- Evaluate Algorithms.
- Improve Results.
- Present Results.

The best way to really come to terms with a new platform or tool is to work through a machine learning project end-to-end and cover the key steps. Namely, from loading data, summarizing data, evaluating algorithms and making some predictions.

Here is an overview of what we are going to cover:

1. Installing the Python anaconda platform.
2. Loading the dataset.
3. Summarizing the dataset.
4. Visualizing the dataset.
5. Evaluating some algorithms.
6. Making some predictions.

3.3 PROJECT REQUIREMENTS

General:

Requirements are the basic constraints that are required to develop a system. Requirements are collected while designing the system. The following are the requirements that are to be discussed.

1. Functional requirements
2. Non-Functional requirements
3. Environment requirements
 - A. Hardware requirements
 - B. software requirements

Functional requirements:

The software requirements specification is a technical specification of requirements for the software product. It is the first step in the requirements analysis process. It lists requirements of a particular software system. The following details to follow the special libraries like tensorflow, keras, matplotlib, LSTM.

Non-Functional Requirements:

Process of functional steps,

1. Problem define
2. Preparing data
3. Evaluating algorithm
4. Improving results
5. Prediction the result

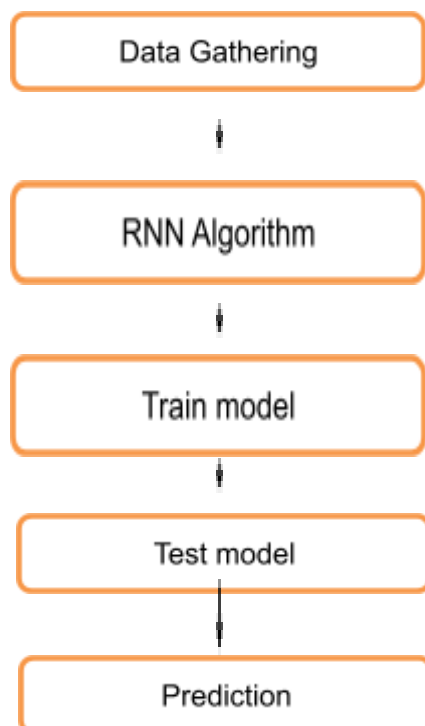
3.4 FEASIBILITY STUDY

Splitting the dataset:

The data use is usually split into training data and test data. The training set contains a known output and the model learns on this data in order to be generalized to other data later on. It has the test dataset (or subset) in order to test our models and it will do this using Tensor flow library in Python using the LSTM method.

Construction of a Detecting Model:

Deep learning needs data gathering have lot of past image data's. Training and testing this model working and predicting correctly.

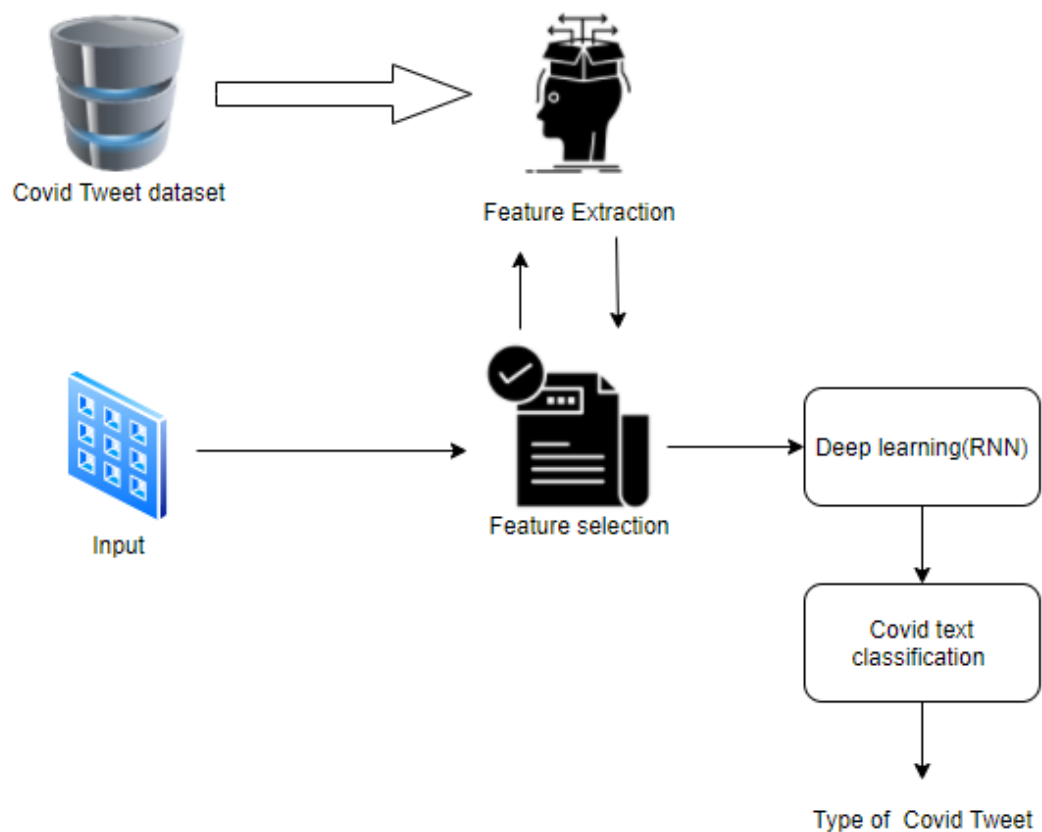


DESIGN ENGINEERING

General

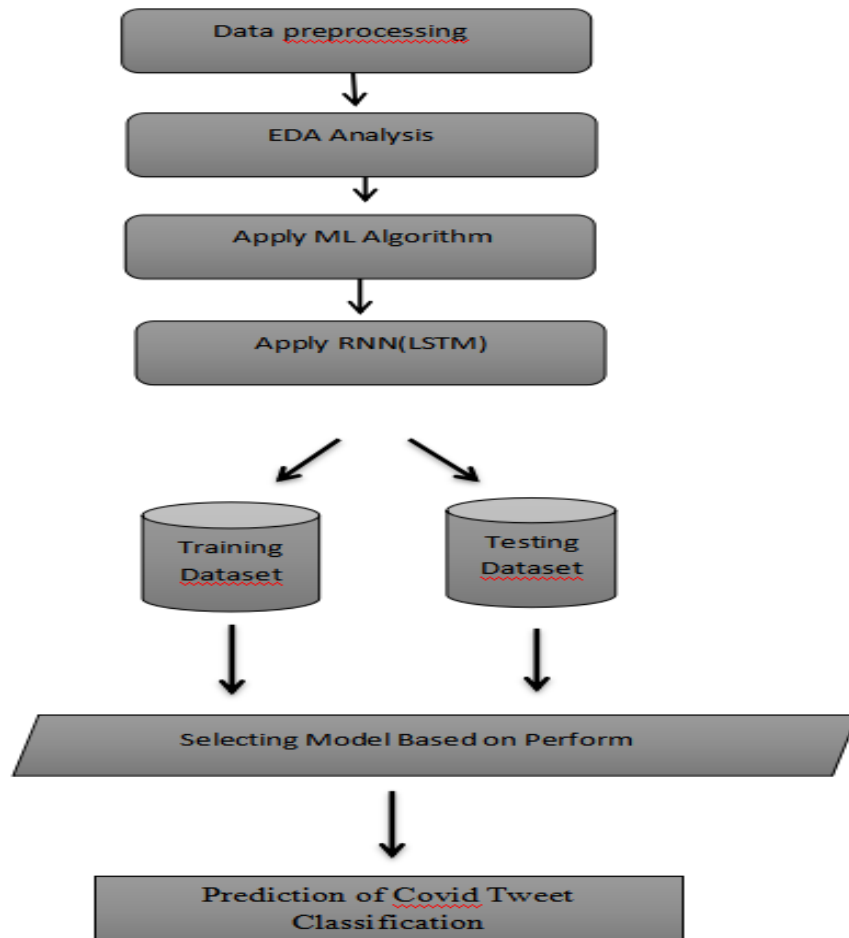
Design is meaningful engineering representation of something that is to be built. Software design is a process design is the perfect way to accurately translate requirements in to a finished software product. Design creates a representation or model, provides detail about software data structure, architecture, interfaces and components that are necessary to implement a system.

3.5 System Architecture:



3.5.1.Architecture

3.6 Work flow diagram:



3.6.1.Flow Diagram

3.7 DESIGN ARCHITECTURE:

3.7.1 USECASE DIAGRAM:

Use case diagrams are considered for high level requirement analysis of a system. So when the requirements of a system are analyzed the functionalities are captured in use cases. So, it can say that uses cases are nothing but the system functionalities written in an organized manner.

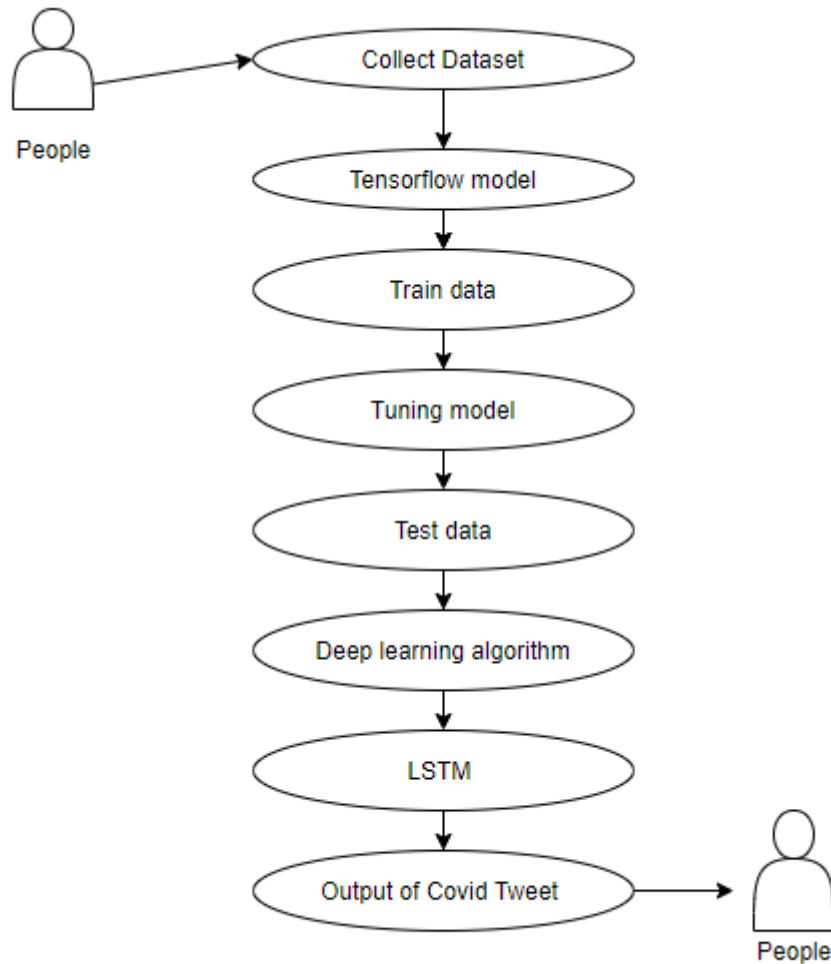


Fig 3.7.1

3.7.2 CLASS DIAGRAM:

Class diagram is basically a graphical representation of the static view of the system and represents different aspects of the application. So a collection of class diagrams represent the whole system. The name of the class diagram should be meaningful to describe the aspect of the system. Each element and their relationships should be identified in advance Responsibility (attributes and methods) of each class should be clearly identified for each class minimum number of properties should be specified and because, unnecessary properties will make the diagram complicated. Use notes whenever required to describe some aspect of the diagram and at the end of the drawing it should be understandable to the developer/coder. Finally, before making the final version,

the diagram should be drawn on plain paper and rework as many times as possible to make it correct.

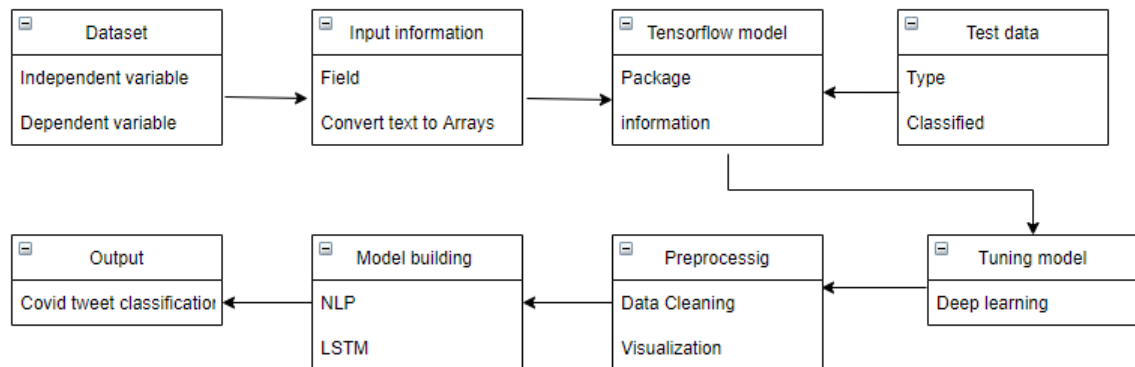


Fig 3.7.2

3.7.3 ACTIVITY DIAGRAM:

Activity is a particular operation of the system. Activity diagrams are not only used for visualizing dynamic nature of a system but they are also used to construct the executable system by using forward and reverse engineering techniques. The only missing thing in activity diagram is the message part. It does not show any message flow from one activity to another. Activity diagram is some time considered as the flow chart. Although the diagrams looks like a flow chart but it is not. It shows different flow like parallel, branched, concurrent and single.

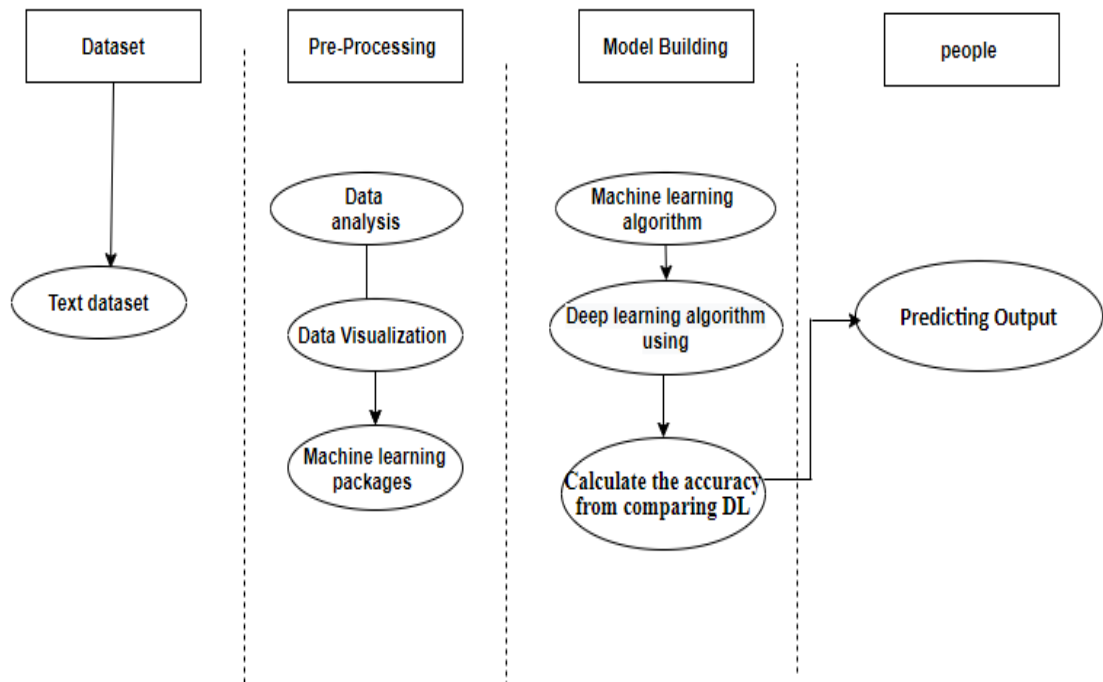


Fig 3.7.3

3.7.4 SEQUENCE DIAGRAM:

Sequence diagrams model the flow of logic within your system in a visual manner, enabling you both to document and validate your logic, and are commonly used for both analysis and design purposes. Sequence diagrams are the most popular UML artifact for dynamic modeling, which focuses on identifying the behavior within your system. Other dynamic modeling techniques include activity diagramming, communication diagramming, timing diagramming, and interaction overview diagramming. Sequence diagrams, along with class diagrams and physical data models are in my opinion the most important design-level models for modern business application development.

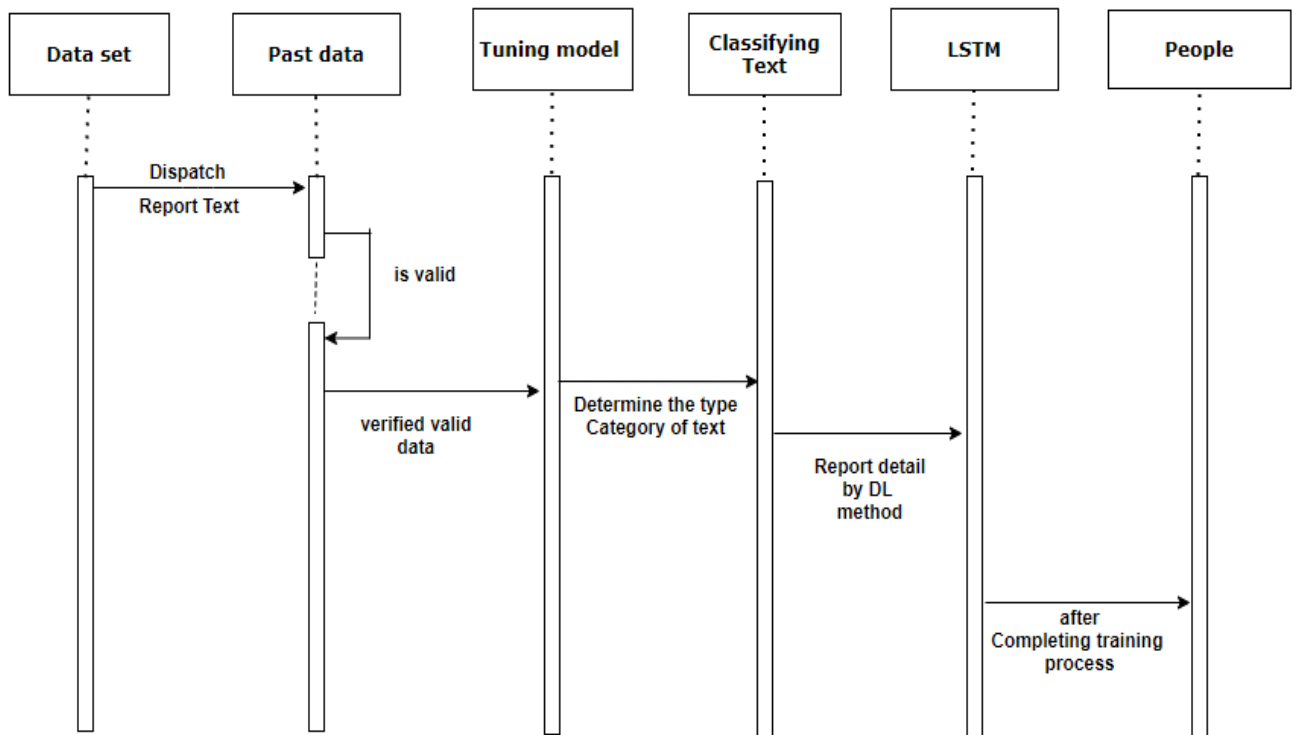


Fig 3.7.4

3.7.5 ER DIAGRAM:

An entity relationship diagram (ERD), also known as an entity relationship model, is a graphical representation of an information system that depicts the relationships among people, objects, places, concepts or events within that system. An ERD is a data modeling technique that can help define business processes and be used as the foundation for a relational database. Entity relationship diagrams provide a visual starting point for database design that can also be used to help determine information system requirements throughout an organization. After a relational database is rolled out, an ERD can still serve as a referral point, should any debugging or business process re-engineering be needed later.

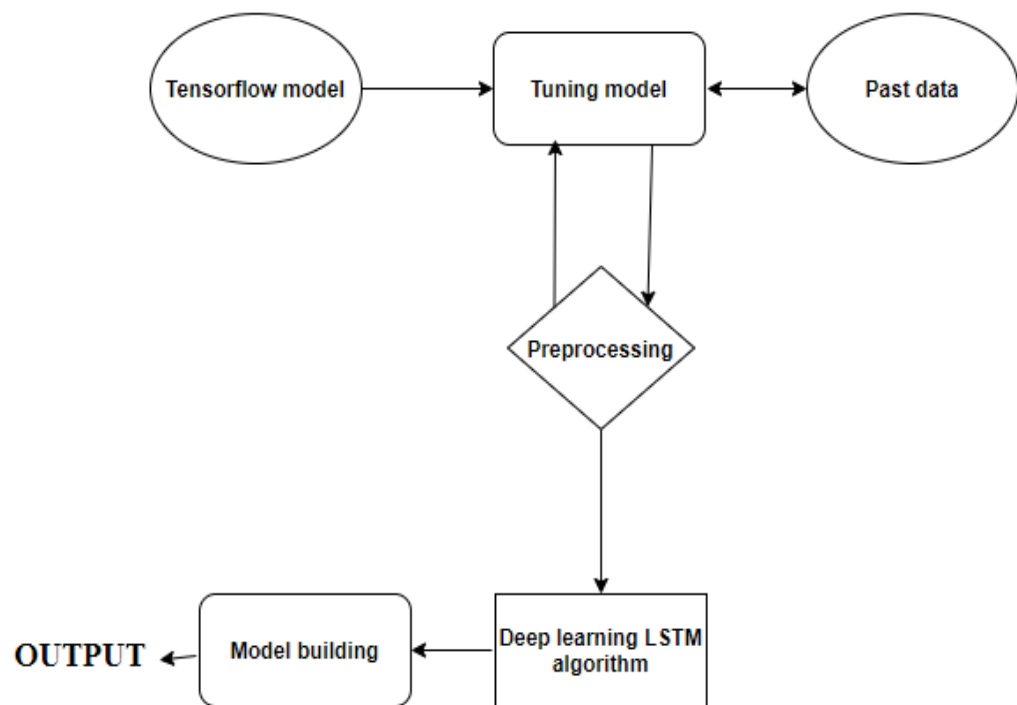


Fig 3.7.5

3.7.6 COLLABORATION DIAGRAM:

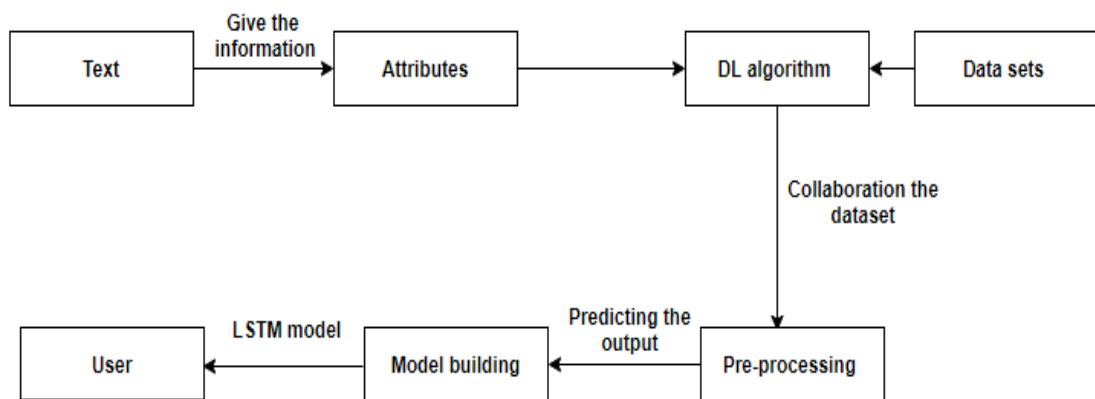


Fig 3.7.6

3.8 Environment Requirements:

1. Software Requirements:

Operating System : Windows / Linux

Simulation Tool : Anaconda with Jupyter Notebook

2. Hardware requirements:

Processor : Pentium IV/III

Hard disk : minimum 80 GB

RAM : minimum 2 GB

1. SOFTWARE DESCRIPTION

Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. Package versions are managed by the package management system “Conda”. The Anaconda distribution is used by over 12 million users and includes more than 1400 popular data-science packages suitable for Windows, Linux, and MacOS. So, Anaconda distribution comes with more than 1,400

packages as well as the Conda package and virtual environment manager called Anaconda Navigator and it eliminates the need to learn to install each library independently. The open source packages can be individually installed from the Anaconda repository with the conda install command or using the pip install command that is installed with Anaconda. Pip packages provide many of the features of conda packages and in most cases they can work together. Custom packages can be made using the conda build command, and can be shared with others by uploading them to Anaconda Cloud, PyPI or other repositories. The default installation of Anaconda2 includes Python 2.7 and Anaconda3 includes Python 3.7. However, you can create new environments that include any version of Python packaged with conda.

3.9.METHODOLOGY:

3.9.1 Natural Language Processing:

Natural language processing (NLP) refers to the branch of computer science and more specifically, the branch of artificial intelligence or AI concerned with giving computers the ability to understand text and spoken words in much the same way human beings can.

NLP combines computational linguistics rule-based modeling of human language with statistical, machine learning, and deep learning models. Together, these technologies enable computers to process human language in the form of text or voice data and to ‘understand’ its full meaning, complete with the speaker or writer’s intent and sentiment.

NLP drives computer programs that translate text from one language to another, respond to spoken commands, and summarize large volumes of text rapidly even in real time. There’s a good chance you’ve interacted with NLP in the form of voice-operated GPS systems, digital assistants, speech-to-text dictation software, customer service chatbots, and other consumer conveniences. But NLP also plays a growing role in enterprise solutions that help streamline business operations, increase employee productivity, and simplify mission-critical business processes.

Python and the Natural Language Toolkit (NLTK)

The Python programming language provides a wide range of tools and libraries for attacking specific NLP tasks. Many of these are found in the Natural Language Toolkit, or NLTK, an open source collection of libraries, programs, and education resources for building NLP programs.

The NLTK includes libraries for many of the NLP tasks listed above, plus libraries for subtasks, such as sentence parsing, word segmentation, stemming and lemmatization (methods of trimming words down to their roots), and tokenization (for breaking phrases, sentences, paragraphs and passages into tokens that help the computer better understand the text). It also includes libraries for implementing capabilities such as semantic reasoning, the ability to reach logical conclusions based on facts extracted from text.

Statistical NLP, machine learning, and deep learning

The earliest NLP applications were hand-coded, rules-based systems that could perform certain NLP tasks, but couldn't easily scale to accommodate a seemingly endless stream of exceptions or the increasing volumes of text and voice data.

Enter statistical NLP, which combines computer algorithms with machine learning and deep learning models to automatically extract, classify, and label elements of text and voice data and then assign a statistical likelihood to each possible meaning of those elements. Today, deep learning models and learning techniques based on convolutional neural networks (CNNs) and recurrent neural networks (RNNs) enable NLP systems that 'learn' as they work and extract ever more accurate meaning from huge volumes of raw, unstructured, and unlabeled text and voice data sets.

Lemmatization

The task of removing inflectional endings only and to return the base dictionary form of a word which is also known as a lemma. Lemmatization is another technique for reducing words to their normalized form. But in this case, the transformation actually uses a dictionary to map words to their actual form.

Morphological segmentation

Separate words into individual morphemes and identify the class of the morphemes. The difficulty of this task depends greatly on the complexity of the morphology (*i.e.*, the structure of words) of the language being considered. English has fairly simple morphology, especially inflectional morphology, and thus it is often possible to ignore this task entirely and simply model all possible forms of a word (*e.g.*, "open, opens, opened, and opening") as separate words. In languages such as Turkish or Meitei,^[24] a highly agglutinated Indian language, however, such an approach is not possible, as each dictionary entry has thousands of possible word forms.

Part-of-speech tagging

Given a sentence, determine the part of speech (POS) for each word. Many words, especially common ones, can serve as multiple parts of speech. For example, "book" can be a noun ("the book on the table") or verb ("to book a flight"); "set" can be a noun, verb or adjective; and "out" can be any of at least five different parts of speech.

Stemming

The process of reducing inflected (or sometimes derived) words to a base form (*e.g.*, "close" will be the root for "closed", "closing", "close", "closer" etc.). Stemming yields similar results as lemmatization, but does so on grounds of rules, not a dictionary.

3.9.2 NLP Concept:

Most higher-level NLP applications involve aspects that emulate intelligent behaviour and apparent comprehension of natural language. More broadly speaking, the technical operationalization of increasingly advanced aspects of cognitive behaviour represents one of the developmental trajectories of NLP (see trends among CoNLL shared tasks above).

Cognition refers to "the mental action or process of acquiring knowledge and understanding through thought, experience, and the senses."^[38] Cognitive science is the interdisciplinary, scientific study of the mind and its processes.^[39] Cognitive linguistics is an interdisciplinary branch of linguistics,

combining knowledge and research from both psychology and linguistics. Especially during the age of symbolic NLP, the area of computational linguistics maintained strong ties with cognitive studies.

As an example, George Lakoff offers a methodology to build natural language processing (NLP) algorithms through the perspective of cognitive science, along with the findings of cognitive linguistics,^[41] with two defining aspects:

1. Apply the theory of conceptual metaphor, explained by Lakoff as “the understanding of one idea, in terms of another” which provides an idea of the intent of the author. For example, consider the English word “big”. When used in a comparison (“That is a big tree”), the author's intent is to imply that the tree is “physically large” relative to other trees or the authors experience. When used metaphorically (“Tomorrow is a big day”), the author’s intent to imply “importance”. The intent behind other usages, like in “She is a big person” will remain somewhat ambiguous to a person and a cognitive NLP algorithm alike without additional information.

$$w_{x,y} = tf_{x,y} \times \log \left(\frac{N}{df_x} \right)$$

TF-IDF

Term x within document y

$tf_{x,y}$ = frequency of x in y

df_x = number of documents containing x

N = total number of documents

2. Assign relative measures of meaning to a word, phrase, sentence or piece of text based on the information presented before and after the piece of text being analyzed, e.g., by means of a probabilistic context-free grammar (PCFG). The mathematical equation for such algorithms is presented.

Training Process:

`train_datagen.flow_from_directory` is the function that is used to prepare data from the `train_dataset` directory. `Target_size` specifies the target size of the image. `Test_datagen.flow_from_directory` is used to prepare test data for the model and all is similar as above. `fit_generator` is used to fit the data into the model made above, other factors used are `steps_per_epochs` tells us about the number of times the model will execute for the training data.

Epochs:

It tells us the number of times model will be trained in forward and backward pass.

Validation process:

`Validation_data` is used to feed the validation/test data into the model. `Validation_steps` denotes the number of validation/test samples.

3.9.3.ARTIFICIAL NEURAL NETWORK:

Artificial Neural Networks (ANN) are multi-layer fully-connected neural nets that look like the figure below. They consist of an input layer, multiple hidden layers, and an output layer. Every node in one layer is connected to every other node in the next layer. We make the network deeper by increasing the number of hidden layers.

Artificial Neural Network(ANN) uses the processing of the brain as a basis to develop algorithms that can be used to model complex patterns and prediction problems.

Key Points related to the architecture:

The network architecture has an input layer, hidden layer (there can be more than 1) and the output layer. It is also called MLP (Multi Layer Perceptron) because of the multiple layers. The hidden layer can be seen as a “distillation layer” that distills some of the important patterns from the inputs and passes it onto the next layer to

see. It makes the network faster and efficient by identifying only the important information from the inputs leaving out the redundant information

3. The activation function serves two notable purposes:

- It captures non-linear relationship between the inputs
- It helps convert the input into a more useful output.

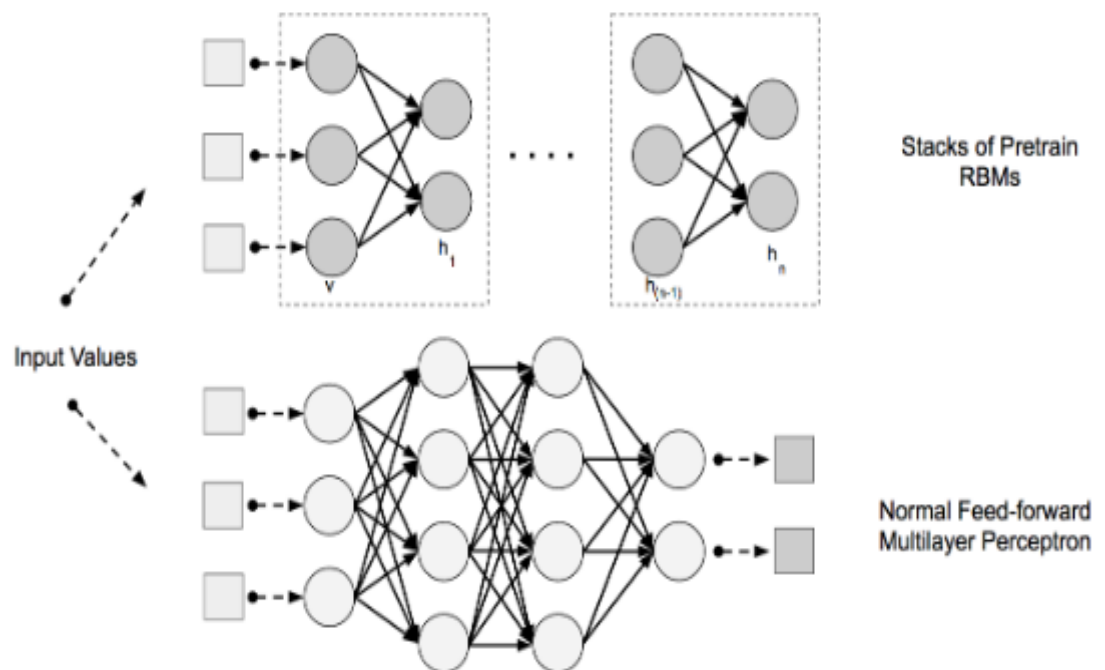


Fig 3.9.3.1

Key advantages of neural Networks:

ANNs have some key advantages that make them most suitable for certain problems and situations:

1. ANNs have the ability to learn and model non-linear and complex relationships,

which is really important because in real-life, many of the relationships between inputs and outputs are non-linear as well as complex.

2. ANNs can generalize — After learning from the initial inputs and their relationships, it can infer unseen relationships on unseen data as well, thus making the model generalize and predict on unseen data.

3. Unlike many other prediction techniques, ANN does not impose any restrictions on the input variables (like how they should be distributed). Additionally, many studies have shown that ANNs can better model heteroskedasticity i.e. data with high volatility and non-constant variance, given its ability to learn hidden relationships in the data without imposing any fixed relationships in the data. This is something very useful in financial time series forecasting (e.g. stock prices) where data volatility is very high.

2. ARCHITECTURE OF RNN

Recurrent neural networks (RNN) are a class of neural networks that are helpful in modeling sequence data. Derived from feed forward networks, RNNs exhibit similar behavior to how human brains function.

Long Short-term memory (LSTM)

Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture used in the field of deep learning. Unlike standard feedforward neural networks, LSTM has feedback connections. It can process not only single data points (such as images), but also entire sequences of data (such as speech or video). For example, LSTM is applicable to tasks such as unsegmented, connected handwriting recognition, speech recognition and anomaly detection in network traffic or IDSs (intrusion detection systems).

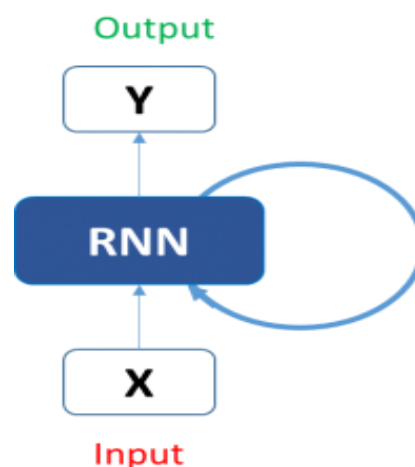
A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three *gates* regulate the flow of information into and out of the cell.

LSTM networks are well-suited to classifying, processing and making predictions based on time series data, since there can be lags of unknown duration between important events in a time series. LSTMs were developed to deal with the vanishing gradient problem that can be encountered when training traditional RNNs. Relative insensitivity to gap length is an advantage of LSTM over RNNs, hidden Markov models and other sequence learning methods in numerous applications.

Take an example of sequential data, which can be the stock market's data for a particular stock. A simple machine learning model or an Artificial Neural Network may learn to predict the stock prices based on a number of features: the volume of the stock, the opening value etc. While the price of the stock depends on these features, it is also largely dependent on the stock values in the previous days. In fact for a trader, these values in the previous days (or the trend) is one major deciding factor for predictions.

In the conventional feed-forward neural networks, all test cases are considered to be independent. That is when fitting the model for a particular day, there is no consideration for the stock prices on the previous days.

This dependency on time is achieved via Recurrent Neural Networks. A typical RNN looks like:



This may be intimidating at first sight, but once unfolded, it looks a lot simpler:

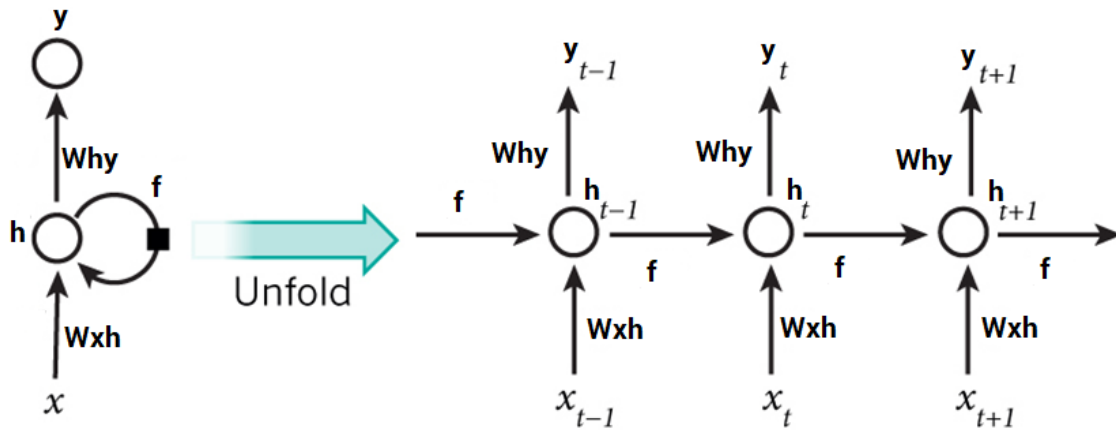


Fig 3.9.3.2

Now it is easier for us to visualize how these networks are considering the trend of stock prices, before predicting the stock prices for today. Here every prediction at time t (h_t) is dependent on all previous predictions and the information learned from them.

RNNs can solve our purpose of sequence handling to a great extent but not entirely. We want our computers to be good enough to write Shakespearean sonnets. Now RNNs are great when it comes to short contexts, but in order to be able to build a story and remember it, we need our models to be able to understand and remember the context behind the sequences, just like a human brain.

3.9.4.Models API:

There are three ways to create Keras models:

- The Sequential model, which is very straightforward (a simple list of layers), but is limited to single-input, single-output stacks of layers (as the name gives away).
- The Functional API, which is an easy-to-use, fully-featured API that supports arbitrary model architectures. For most people and most use cases, this is what you should be using. This is the Keras "industry strength" model.
- Model subclassing, where you implement everything from scratch on your own. Use this if you have complex, out-of-the-box research use cases.

Types of Keras Models

Models in keras are available in two types:

- Keras Sequential Model
- Keras Functional API

1. Sequential Model in Keras

It allows us to create models layer by layer in sequential order. But it does not allow us to create models that have multiple inputs or outputs.

It is best for simple stack of layers which have 1 input tensor and 1 output tensor.

This model is not suited when any of the layer in the stack has multiple inputs or outputs. Even if we want non-linear topology, it is not suited.

2. Functional API in Keras

It provides more flexibility to define a model and add layers in keras. Functional API allows us to create models that have multiple input or output.

It also allows us to share these layers. In other words. we can make graphs of layers using Keras functional API.

As functional API is a data structure, it is easy to save it as a single file that helps in recreating the exact model without having the original code. Also its easy to model the graph here and access its nodes as well.

3.9.5.TYPES OF ALGORITHMS:

- Random Forest Classifier
- Logistic regression

Random Forest Classifier:

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of over fitting to their training set. Random forest is a type of supervised machine learning algorithm based on ensemble learning. Ensemble learning is a type of learning

where you join different types of algorithms or same algorithm multiple times to form a more powerful prediction model. The random forest algorithm combines multiple algorithm of the same type i.e. multiple decision trees, resulting in a forest of trees, hence the name "Random Forest". The random forest algorithm can be used for both regression and classification tasks.

The following are the basic steps involved in performing the random forest algorithm:

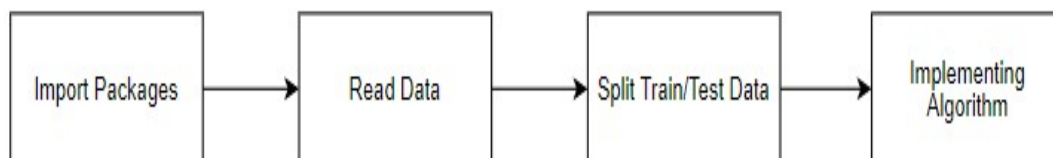
Pick N random records from the dataset.

Build a decision tree based on these N records.

Choose the number of trees you want in your algorithm and repeat steps 1 and 2.

In case of a regression problem, for a new record, each tree in the forest predicts a value for Y (output). The final value can be calculated by taking the average of all the values predicted by all the trees in forest. Or, in case of a classification problem, each tree in the forest predicts the category to which the new record belongs. Finally, the new record is assigned to the category that wins the majority vote.

MODULE DIAGRAM



GIVEN INPUT EXPECTED OUTPUT

input : data

output : getting accuracy

Logistic Regression:

It is a statistical method for analysing a data set in which there are one or more independent variables that determine an outcome. The outcome is measured with a dichotomous variable (in which there are only two possible outcomes). The goal of logistic regression is to find the best fitting model to describe the relationship between the dichotomous characteristic of interest (dependent variable = response or outcome variable) and a set of independent (predictor or explanatory) variables. Logistic regression is a Machine Learning classification algorithm that is used to predict the probability of a categorical dependent variable. In logistic regression, the dependent variable is a binary variable that contains data coded as 1 (yes, success, etc.) or 0 (no, failure, etc.).

In other words, the logistic regression model predicts $P(Y=1)$ as a function of X . Logistic regression Assumptions:

Binary logistic regression requires the dependent variable to be binary.

For a binary regression, the factor level 1 of the dependent variable should represent the desired outcome.

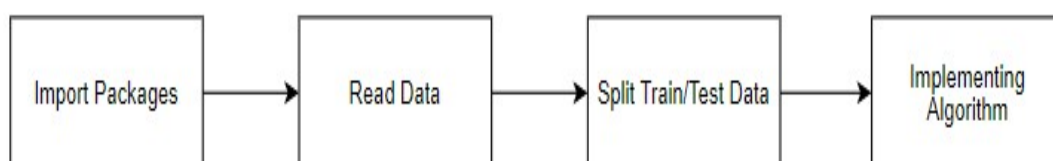
Only the meaningful variables should be included.

The independent variables should be independent of each other. That is, the model should have little.

The independent variables are linearly related to the log odds.

Logistic regression requires quite large sample sizes.

MODULE DIAGRAM



GIVEN INPUT EXPECTED OUTPUT

input : data

output : getting accuracy

3.10 LIST OF MODULES

1. Data validation Process EDA
2. Exploration data analysis of visualization
3. Comparing Algorithm with prediction in the form of best accuracy result
4. Deep Learning RNN with LSTM get best accuracy result
5. Output for prediction of sentiment by giving input sentences.

MODULE DESCRIPTION

3.10.1 Data validation process EDA:

Validation techniques in machine learning are used to get the error rate of the Machine Learning (ML) model, which can be considered as close to the true error rate of the dataset. If the data volume is large enough to be representative of the population, you may not need the validation techniques. However, in real-world scenarios, to work with samples of data that may not be a true representative of the population of given dataset. To finding the missing value, duplicate value and description of data type whether it is float variable or integer. The sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyper parameters.

The evaluation becomes more biased as skill on the validation dataset is incorporated into the model configuration. The validation set is used to evaluate a given model, but this is for frequent evaluation. It as machine learning engineers uses this data to fine-tune the model hyper parameters. Data collection, data analysis, and the process of addressing data content, quality, and structure can add up to a time-consuming to-do list. During the process of data identification, it

helps to understand your data and its properties; this knowledge will help you choose which algorithm to use to build your model. For example, time series data can be analyzed by regression algorithms; classification algorithms can be used to analyze discrete data. For example to show the data type format of given dataset. A number of different data cleaning tasks using Python's Pandas library and specifically, it focus on probably the biggest data cleaning task, missing values and it able to more quickly clean data. It wants to spend less time cleaning data, and more time exploring and modeling. Some of these sources are just simple random mistakes. Other times, there can be a deeper reason why data is missing. It's important to understand these different types of missing data from a statistics point of view. The type of missing data will influence how to deal with filling in the missing values and to detect missing values, and do some basic imputation and detailed statistical approach for dealing with missing data. Before, joint into code, it's important to understand the sources of missing data. Here are some typical reasons why data is missing:

- User forgot to fill in a field.
- Data was lost while transferring manually from a legacy database.
- There was a programming error.
- Users chose not to fill out a field tied to their beliefs about how the results would be used or interpreted.

Variable identification with Uni-variate, Bi-variate and Multi-variate analysis:

import libraries for access and functional purpose and read the given dataset

General Properties of Analyzing the given dataset

Display the given dataset in the form of data frame

show columns

shape of the data frame

To describe the data frame

Checking data type and information about dataset

Checking for duplicate data

Checking Missing values of data frame

- Checking unique values of data frame
- Checking count values of data frame
- Rename and drop the given data frame
- To specify the type of values
- To create extra columns

Data Validation/ Cleaning/Preparing Process

Importing the library packages with loading given dataset. To analyzing the variable identification by data shape, data type and evaluating the missing values, duplicate values. A validation dataset is a sample of data held back from training your model that is used to give an estimate of model skill while tuning model's and procedures that you can use to make the best use of validation and test datasets when evaluating your models. Data cleaning / preparing by rename the given dataset and drop the column etc. to analyze the uni-variate, bi-variate and multi-variate process. The steps and techniques for data cleaning will vary from dataset to dataset. The primary goal of data cleaning is to detect and remove errors and anomalies to increase the value of data in analytics and decision making.

3.10.2 Exploration data analysis of visualization

Data visualization is an important skill in applied statistics and machine learning. Statistics does indeed focus on quantitative descriptions and estimations of data. Data visualization provides an important suite of tools for gaining a qualitative understanding. This can be helpful when exploring and getting to know a dataset and can help with identifying patterns, corrupt data, outliers, and much more. With a little domain knowledge, data visualizations can be used to express and demonstrate key relationships in plots and charts that are more visceral and stakeholders than measures of association or significance. Data visualization and exploratory data analysis are whole fields themselves and it will recommend a deeper dive into some the books mentioned at the end.

Sometimes data does not make sense until it can look at in a visual form, such as with charts and plots. Being able to quickly visualize of data samples and others is an important skill both in applied statistics and in applied machine learning. It will discover the many types of plots that you will need to know when

visualizing data in Python and how to use them to better understand your own data.

How to chart time series data with line plots and categorical quantities with bar charts.

How to summarize data distributions with histograms and box plots.

How to summarize the relationship between variables with scatter plots.

Data Visualization types:

- Introduction to Matplotlib
- Line Plot
- Bar Chart
- Histogram Plot
- Box and Whisker Plot
- Scatter Plot

There are many excellent plotting libraries in Python and it recommend exploring them in order to create presentable graphics. For quick and dirty plots intended for your own use, it recommends using the matplotlib library. It is the foundation for many other plotting libraries and plotting support in higher-level libraries such as Pandas. The matplotlib provides a context, one in which one or more plots can be drawn before the image is shown or saved to file and context can be accessed via functions on pyplot.

Bar Chart:

A bar chart is generally used to present relative quantities for multiple categories. The x-axis represents the categories and are spaced evenly. The y-axis represents the quantity for each category and is drawn as a bar from the baseline to the appropriate level on the y-axis.

Box and Plot:

A box and whisker plot, or boxplot for short, is generally used to summarize the distribution of a data sample. The x-axis is used to represent the data sample, where multiple boxplots can be drawn side by side on the x-axis if desired. The boxplot is a graphical technique that displays the distribution of variables. It helps

us see the location, skewness, spread, tile length and outlying points. The boxplot is a graphical representation of the Five Number Summary.

Heat map

A heat map is a graphical representation of data where the individual values contained in a matrix are represented as colors. It is a bit like looking a data table from above. It is really useful to display a general view of numerical data, not to extract specific data point. It is quite straight forward to make a heat map, as shown on the examples below.

However be careful to understand the underlying mechanisms. You will probably need to normalize your matrix, choose a relevant color palette, use cluster analysis and thus permute the rows and the columns of the matrix to place similar values near each other according to the clustering.

Advantages of train/test split:

1. This runs K times faster than Leave One Out cross-validation because K-fold cross-validation repeats the train/test split K-times.
2. Simpler to examine the detailed results of the testing process.

3.10.3 Comparing algorithm with prediction in the form of best accuracy result

It is important to compare the performance of multiple different machine learning algorithms consistently and it will discover to create a test harness to compare multiple different machine learning algorithms in Python with scikit-learn. It can use this test harness as a template on your own machine learning problems and add more and different algorithms to compare. Each model will have different performance characteristics. Using resampling methods like cross validation, you can get an estimate for how accurate each model may be on unseen data. It needs to be able to use these estimates to choose one or two best models from the suite of models that you have created. When have a new dataset, it is a good idea to visualize the data using different techniques in order to look at the data from different perspectives. The same idea applies to model selection. You should

use a number of different ways of looking at the estimated accuracy of your machine learning algorithms in order to choose the one or two to finalize. A way to do this is to use different visualization methods to show the average accuracy, variance and other properties of the distribution of model accuracies.

In the next section you will discover exactly how you can do that in Python with scikit-learn. The key to a fair comparison of machine learning algorithms is ensuring that each algorithm is evaluated in the same way on the same data and it can achieve this by forcing each algorithm to be evaluated on a consistent test harness.

Random Forest

Logistic Regression

The K-fold cross validation procedure is used to evaluate each algorithm, importantly configured with the same random seed to ensure that the same splits to the training data are performed and that each algorithm is evaluated in precisely the same way. Before that comparing algorithm, building a Machine Learning Model using install Scikit-Learn libraries. In this library package have to done preprocessing, linear model with logistic regression method, cross validating by KFold method, ensemble with random forest method and tree with decision tree classifier. Additionally, splitting the train set and test set. To predicting the result by comparing accuracy.

Prediction result by accuracy:

Logistic regression algorithm also uses a linear equation with independent predictors to predict a value. The predicted value can be anywhere between negative infinity to positive infinity. We need the output of the algorithm to be classified variable data. Higher accuracy predicting result is logistic regression model by comparing the best accuracy.

True Positive Rate (TPR) = $TP / (TP + FN)$

False Positive rate (FPR) = $FP / (FP + TN)$

Accuracy: The Proportion of the total number of predictions that is correct

otherwise overall how often the model predicts correctly defaulters and non-defaulters.

Accuracy calculation:

$$\text{Accuracy} = (TP + TN) / (TP + TN + FP + FN)$$

Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations. One may think that, if we have high accuracy then our model is best. Yes, accuracy is a great measure but only when you have symmetric datasets where values of false positive and false negatives are almost same.

Precision: The proportion of positive predictions that are actually correct. (When the model predicts default: how often is correct?)

$$\text{Precision} = TP / (TP + FP)$$

Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. The question that this metric answer is of all passengers that labeled as survived, how many actually survived? High precision relates to the low false positive rate. We have got 0.788 precision which is pretty good.

Recall: The proportion of positive observed values correctly predicted. (The proportion of actual defaulters that the model will correctly predict)

$$\text{Recall} = TP / (TP + FN)$$

Recall (Sensitivity) - Recall is the ratio of correctly predicted positive observations to the all observations in actual class - yes.

ALGORITHM AND TECHNIQUES

Algorithm Explanation

In machine learning and statistics, classification is a supervised learning approach in which the computer program learns from the data input given to it and then uses this learning to classify new observation. This data set may simply be bi-class (like identifying whether the person is male or female or that the mail is spam or non-spam) or it may be multi-class too. Some examples of classification problems are: speech recognition, handwriting recognition, bio metric

identification, document classification etc. In Supervised Learning, algorithms learn from labeled data. After understanding the data, the algorithm determines which label should be given to new data based on pattern and associating the patterns to the unlabeled new data.

Used Python Packages:

sklearn:

- In python, sklearn is a machine learning package which include a lot of ML algorithms.
- Here, we are using some of its modules like `train_test_split`, `DecisionTreeClassifier` or `Logistic Regression` and `accuracy_score`.

NumPy:

- It is a numeric python module which provides fast maths functions for calculations.
- It is used to read data in numpy arrays and for manipulation purpose.

Pandas:

- Used to read and write different files.
- Data manipulation can be done easily with data frames.

Matplotlib:

- Data visualization is a useful way to help with identify the patterns from given dataset.
- Data manipulation can be done easily with data frames.

Random Forest

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of over fitting to their training set. Random forest is a type of supervised machine learning algorithm based on ensemble learning. Ensemble learning is a type of learning where you join different types of algorithms or same algorithm multiple times to form a more powerful prediction model. The random forest algorithm combines multiple algorithm of the same type i.e. multiple decision *trees*, resulting in a *forest*

of *trees*, hence the name "Random Forest". The random forest algorithm can be used for both regression and classification tasks.

The following are the basic steps involved in performing the random forest algorithm:

- Pick N random records from the dataset.

- Build a decision tree based on these N records.

- Choose the number of trees you want in your algorithm and repeat steps 1 and 2.

In case of a regression problem, for a new record, each tree in the forest predicts a value for Y (output). The final value can be calculated by taking the average of all the values predicted by all the trees in forest. Or, in case of a classification problem, each tree in the forest predicts the category to which the new record belongs. Finally, the new record is assigned to the category that wins the majority vote.

Logistic Regression

It is a statistical method for analyzing a data set in which there are one or more independent variables that determine an outcome. The outcome is measured with a dichotomous variable (in which there are only two possible outcomes). The goal of logistic regression is to find the best fitting model to describe the relationship between the dichotomous characteristic of interest (dependent variable = response or outcome variable) and a set of independent (predictor or explanatory) variables. Logistic regression is a Machine Learning classification algorithm that is used to predict the probability of a categorical dependent variable. In logistic regression, the dependent variable is a binary variable that contains data coded as 1 (yes, success, etc.) or 0 (no, failure, etc.).

In other words, the logistic regression model predicts $P(Y=1)$ as a function of X. Logistic regression Assumptions:

- Binary logistic regression requires the dependent variable to be binary.

For a binary regression, the factor level 1 of the dependent variable should represent the desired outcome.

Only the meaningful variables should be included.

The independent variables should be independent of each other. That is, the model should have little.

The independent variables are linearly related to the log odds.

Logistic regression requires quite large sample sizes.

3.10.4 Deep Learning RNN with LSTM get best accuracy result.

Recurrent Neural Network is a generalization of feedforward neural network that has an internal memory. RNN is recurrent in nature as it performs the same function for every input of data while the output of the current input depends on the past one computation. After producing the output, it is copied and sent back into the recurrent network. For making a decision, it considers the current input and the output that it has learned from the previous input. Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition. In other neural networks, all the inputs are independent of each other. But in RNN, all the inputs are related to each other. First, it takes the $X(0)$ from the sequence of input and then it outputs $h(0)$ which together with $X(1)$ is the input for the next step. So, the $h(0)$ and $X(1)$ is the input for the next step. Similarly, $h(1)$ from the next is the input with $X(2)$ for the next step and so on. This way, it keeps remembering the context while training.

Advantages of Recurrent Neural Network:

RNN can model sequence of data so that each sample can be assumed to be dependent on previous ones. Recurrent neural network are even used with convolutional layers to extend the effective pixel neighborhood.

3.10.5 Output for prediction of sentiment by giving input sentences.

code2vec is a neural model that learns analogies relevant to source code. The model was trained on the Java code database but you can apply it to any codebase. Then there's GloVe. the GloVe vocabularies from the website. We took the largest one because then there's a higher chance of it finding all of our words. You can choose where you want to download it but, for convenience, it's better to store it in the working directory, Now we can find out-of-vocabulary words and count the percentage of these words for code2vec vocabulary. The following code will also work for Glove. We've tried three different word embeddings algorithms for OpenAPI specification. Despite the fact that all three perform quite well on this dataset, an extra comparison of the most similar words shows.

3.11 COVID TWEET CLASSIFICATION:

We give input image using keras preprocessing package. That input Image converted into array value using pillow and image to array function package. We have already classified food image dataset. It classifies the Category of Food. Then we have to predict our food image using predict function.

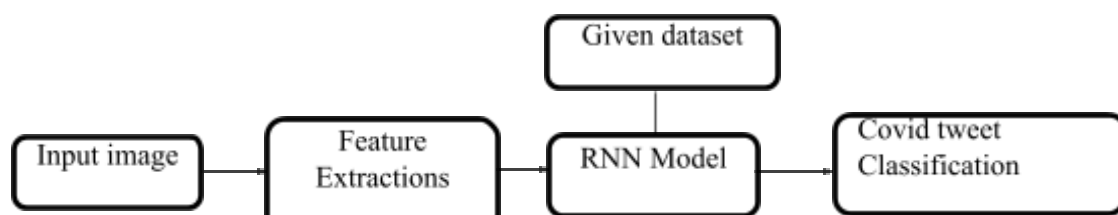


Fig 3.11

The food recognition method is based on a two-channel architecture that is able to recognize classification of category of food. The food images are used as the input into the inception layer of the RNN. The Training phase involves the feature extraction and classification using convolution neural network.

Libraries Required:

- ✓ **tensorflow**: Just to use the tensor board to compare the loss and adam curve our result data or obtained log.
- ✓ **keras**: To pre-process the image dataset.
- ✓ **matplotlib**: To display the result of our predictive outcome.
- ✓ **os**: To access the file system to read the image from the train and test directory from our machines.

3.12 TensorFlow:

TensorFlow is a Python library for fast numerical computing created and released by Google. It is a foundation library that can be used to create Deep Learning models directly or by using wrapper libraries that simplify the process built on top of TensorFlow.

TensorFlow is a software library or framework, designed by the Google team to implement machine learning and deep learning concepts in the easiest manner. It combines the computational algebra of optimization techniques for easy calculation of many mathematical expressions.

Let us now consider the following important features of TensorFlow –

- It includes a feature of that defines, optimizes and calculates mathematical expressions easily with the help of multi-dimensional arrays called tensors.
- It includes a programming support of deep neural networks and machine learning techniques.
- It includes a high scalable feature of computation with various data sets.
- TensorFlow uses GPU computing, automating management. It also includes a unique feature of optimization of same memory and the data used.

3.12.1 Keras:

Keras runs on top of open source machine libraries like TensorFlow, Theano or Cognitive Toolkit (CNTK). Theano is a python library used for fast numerical computation tasks. TensorFlow is the most famous symbolic math library used for creating neural networks and deep learning models. TensorFlow is very flexible and the primary benefit is distributed computing. CNTK is deep learning framework developed by Microsoft. It uses libraries such as Python, C#,

C++ or standalone machine learning toolkits. Theano and TensorFlow are very powerful libraries but difficult to understand for creating neural networks.

Keras is based on minimal structure that provides a clean and easy way to create deep learning models based on TensorFlow or Theano. Keras is designed to quickly define deep learning models. Well, Keras is an optimal choice for deep learning applications.

Features

Keras leverages various optimization techniques to make high level neural network API easier and more performant. It supports the following features –

- Consistent, simple and extensible API.
- Minimal structure - easy to achieve the result without any frills.
- It supports multiple platforms and backends.
- It is user friendly framework which runs on both CPU and GPU.
- Highly scalability of computation.

Matplotlib:

Matplotlib is one of the most popular Python packages used for data visualization. It is a cross-platform library for making 2D plots from data in arrays. Matplotlib is written in Python and makes use of NumPy, the numerical mathematics extension of Python. It provides an object-oriented API that helps in embedding plots in applications using Python GUI toolkits such as PyQt, WxPython or Tkinter. It can be used in Python and IPython shells, Jupyter notebook and web application servers also.

Matplotlib has a procedural interface named the Pylab, which is designed to resemble MATLAB, a proprietary programming language developed by MathWorks. Matplotlib along with NumPy can be considered as the open source equivalent of MATLAB.

OS:

The OS module in Python comes with various functions that enables developers to interact with the Operating system that they are currently working on. In this article we'll be learning mainly to create and delete a directory/folder, rename a directory and even basics of file handling.

Python OS module provides the facility to establish the interaction between the user and the operating system. It offers many useful OS functions that are used to perform OS-based tasks and get related information about operating system.

The OS comes under Python's standard utility modules. This module offers a portable way of using operating system dependent functionality.

CHAPTER 4

Results and Discussion

4.1 Experimental Results:

Our test dataset features COVID-19 India which features tweets associated with COVID-19 from March to September 2020. It consists of more than 150,000 tweets from India. We generate two other datasets from this which feature the regions of Maharashtra (state) and Delhi (union territory) which contains around 18,000 tweets each, respectively.

We first provide visualisation of the dataset and report features of interest to language models such as bi-grams, and tri-grams and distribution of monthly tweets. It shows the distribution of number of tweets for selected months along with the number of cases for the respective datasets.

We notice that the number of tweets in the case of India follows the same trend as the number of novel COVID-19 cases until July, afterwards the number of tweets decline . There is a similar pattern for case of Maharashtra. The case of Delhi is slightly different as the first peak was reached in July with increasing tweets that declined afterwards and did not keep up with the second peak of cases in September. This indicates that as more cases were observed in the early months, there was much concern which eased before the major peak was reached and the number of tweets were drastically decreased. There could be elements of fear, depression and anxiety as the tweets decreased drastically after July with increasing cases, which we analyse in the following section.

It shows a visualisation of the number of bi-grams and tri-grams for the case of India. In case of bi-grams , we find that the term “corona—virus” is mostly used followed by “covid 19” which are essentially the same. Next, it is interesting that the terms “folded hand” are mostly used followed by “positive case” and “social distancing”. The term “folded hand” shows Indian social and religious symbol that denotes keeping the faith, giving respect and also a sign of acknowledgement and appreciation. We note that the “folded hand” is an emotion icon (emoji) used predominantly in social media, and hence been processed as a text during prepossessing , which is actually is not a bi-gram from a semantic viewpoint. In order to provide better understanding of the context of the tweets, we give examples . In the case of tri-grams , we find more information in tweets such as “backhand—index—pointing” which is an emoji; hence, we provide some of the tweets that fall in this category .

4.2.DISCUSSION:

Humans can express more than one sentiment at a time; however, there are variations in the number of sentiments that can be expressed by facial expressions, when compared to spoken or written emotions such as tweets. As shown in , majority of the tweets have one sentiment, both in hand labelled and predicted datasets . This is followed by two sentiments, while a minority have 3 sentiments. We note that there is a significant portion of tweets with no sentiments and there are no tweets of more than three sentiments at a time. The study of emotions with technology has gained much interest in last decade which gave much progress in understanding sentiments. Trampe . presented a study of everyday emotional experiences through an experience sampling smartphone application that monitored real-time emotions of more than 11,000 participants and found the group experienced at least one emotion 90% of the time with joy as the most frequent emotion, followed by love and anxiety. The type of emotion would be highly dependent on the study region which featured Europeans with majority French participants. Cowen and Keltner reported twenty seven distinct categories of emotions bridged by continuous gradients by data from emotionally evocative short videos with varying situational content. These studies contributed to better understanding of emotions given historical viewpoints about context and definitions of emotions and associations between them . However, we did not find any study that reviews the number of emotions that can be expressed at a time in relation to catastrophic events that keep changing with time, such as the rise of COVID-19 cases.

We revisit the case of Indian dataset , where the monthly tweets did not sharply increase with number of novel cases with a nationwide peak of novel . The number of tweets gradually increased with peak of tweets in July . When India had a peak of novel cases, we found that the number of tweets significantly lowered. Hence, people have been alarmed by rising cases, but a month before the peak of novel cases was seen, the tweets were reduced. Moreover, we find that the “optimistic”, “annoyed” and “joking” tweets are mostly dominating the monthly tweets for India and Maharashtra , with a mix of annoyed sentiments in case of Delhi . There is significantly lower number of negative sentiments for the respective datasets .

We note that a limitation of the framework is due to the Senwave training data which considered tweets worldwide during COVID-19 by a group of experts; however, there can be limitations on how experts perceive humour in different regions. Humour is expressed differently in different cultural groups, i.e. a tweet that may be humorous in USA may not be taken as humorous in India due to cultural and language dimensions. There are several studies about the effect of humour in changing cultural or regional settings . A good example is that in orthodox Chinese literature, humour was not expressed due to religious taboo in ancient Buddhism which later eased with Zen Buddhism ; however, the Hindu literature had a different or eased attitude towards humour as expressed in ancient Hindu texts . Although historic textual composition cannot be related to how tweets are expressed in India, it is important to note the cultural and language differences in how humour has been expressed.

Another limitation of the study is regarding the uncertainty of the data. Our results are based on the tweets expressed; however, we note that only a small fraction of the population generally express their views on Twitter. Moreover, we need to be aware that due to restrictions on freedom of speech and political biases, not everyone can express themselves freely. Social media networks have been active in monitoring how people express themselves in order to limit the rise of anti-vaccine sentiments. Furthermore, our framework's training data is based on tweets expressed worldwide, whereas the test data is based on tweets from India during the pandemic. There is a large language diversity in India and at times, people express themselves with a combination of languages with geographic and culture specific jargon. Although, we convert them to English, certain limitations in capturing the context will exist.

We note that there has not been much work in uncertainty quantification for the predictions and there are different level of uncertainties, particularly in model parameters and data. We note that the training data is hand-labelled, and at times two or three sentiments have been expressed at once. This could be something open to interpretation by experts as it is hard to formally detect more than one sentiment from a tweet of only thirty words. Hence, the expert labelled training

dataset adds to uncertainty in model predictions. In future work, Bayesian deep learning can provide a way to address uncertainty in model predictions .

CHAPTER 5

Conclusion and Future Work

5.1 CONCLUSION:

The analytical process started from data cleaning and processing, missing

value, exploratory analysis and finally model building and evaluation. The Machine algorithms like Logistic regression, Decision Tree, Random forest are applied and the accuracy are compared with deep learning algorithm which is RNN(Recurrent Neural Network) with LSTM (Long Short Term Memory).The Deep learning algorithm performance is better than machine learning. The Tweets are classified as Positive, Negative and Neutral based on the given new input tweets.

5.2 FUTURE WORK:

Further improvement on the network's accuracy and generalization can be achieved through the following practices. The first one is to use the whole dataset during the optimization. Using batch optimization is more suitable for larger datasets. Another technique is to evaluate foods one by one. This can lead to detect which category of foods are more difficult to classify. Finally, using a larger dataset for training seems beneficial. However, such a dataset might not exist nowadays. Using several datasets might be a solution, but a careful procedure to normalize them is required. Finally, using full dataset for training, pre-training on each food images, and using a larger dataset seem to have the possibility to improve the network's performance. Thus, they should be addressed in future research on this topic.

APPENDCIES

CODING:

Module 1:

Data validation and pre-processing technique:

```

#import library packages
import pandas as p
import numpy as n
import warnings
warnings.filterwarnings("ignore")
#Load given dataset
data = p.read_csv('Corona_NLP_train.csv',encoding='latin1')
Before drop the given dataset:

data.head()
#shape
data.shape
After drop the given dataset:

df=data.dropna()
df.head()
df.shape
#show columns
df.columns
#To describe the dataframe
df.describe()
#Checking datatype and information about dataset
df.info()
df.Sentiment.unique()
Checking duplicate values of dataframe:

#Checking for duplicate data
df.duplicated()
#find sum of duplicate data
sum(df.duplicated())
#Checking sum of missing values
df.isnull().sum()
Splitting Train/Test:

#preprocessing, split test and dataset, split response variable
X = df.drop(labels='Sentiment', axis=1)
#Response variable

```

```

y = df.loc[:, 'Sentiment']
#We'll use a test size of 30%. We also stratify the split on the response variable,
which is very important to do because there are so few fraudulent transactions.
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=1, stratify=y)
print("Number of training dataset: ", len(X_train))
print("Number of test dataset: ", len(X_test))
print("Total number of dataset: ", len(X_train)+len(X_test))

df.groupby('Sentiment').describe()
#for counting frequently occurrence of spam and ham.
# from collections import Counter
# count1 = Counter(" ".join(df[df['fraudulent']==0]["text"]).split()).most_common(30)
# data1 = p.DataFrame.from_dict(count1)
# data1 = data1.rename(columns={0: "words of Truthful", 1 : "count"})
# count2 = Counter("
".join(df[df['fraudulent']==1]["text"]).split()).most_common(30)
# data2 = p.DataFrame.from_dict(count2)
# data2 = data2.rename(columns={0: "words of Opinion spam", 1 : "count_"})

from collections import Counter
count1 = Counter("
".join(df[df['Sentiment']==0]["OriginalTweet"]).split()).most_common(30)
data1 = p.DataFrame.from_dict(count1)
data1 = data1.rename(columns={0: "Neutral", 1 : "count"})

count2 = Counter("
".join(df[df['Sentiment']==1]["OriginalTweet"]).split()).most_common(30)
data2 = p.DataFrame.from_dict(count2)
data2 = data2.rename(columns={0: "Positive", 1 : "count_"})

```

```

count3 = Counter("
.join(df[df['Sentiment']==2]['OriginalTweet']).split()).most_common(30)
data3 = p.DataFrame.from_dict(count3)
data3 = data1.rename(columns={0: "Negative", 1 : "count"})

count4 = Counter("
.join(df[df['Sentiment']==3]['OriginalTweet']).split()).most_common(30)
data4= p.DataFrame.from_dict(count4)
data4= data1.rename(columns={0: "Extremely Positive", 1 : "count"})

count5 = Counter("
.join(df[df['Sentiment']==4]['OriginalTweet']).split()).most_common(30)
data5 = p.DataFrame.from_dict(count5)
data5 = data1.rename(columns={0: "Extremely Negative", 1 : "count"})
df['Sentiment'].value_counts()
df.head()

```

MODULE 2:

EDA of visualization and training a model by given attributes:

```

#import library packages
import pandas as p
import matplotlib.pyplot as plt
import seaborn as s

```

```

import numpy as n
import warnings
warnings.filterwarnings("ignore")
#Load given dataset
data = p.read_csv('Corona_NLP_train.csv',encoding='latin1')
df=data.dropna()
df
Splitting Train/Test:

df.groupby('Sentiment').describe()
#plotting graph for distribution
import matplotlib.pyplot as plt
import seaborn as sns
sns.countplot(x = "Sentiment", data = df)
df.loc[:, 'Sentiment'].value_counts()
plt.title('Distribution of Tweet')
df.Sentiment.unique()
# plotting graph by length.
truthful =df[df['Sentiment'] == 'Neutral']['OriginalTweet'].str.len()
sns.distplot(truthful, label='0')
deceptive = df[df['Sentiment'] == 'Positive']['OriginalTweet'].str.len()
sns.distplot(deceptive, label='1')
deceptive = df[df['Sentiment'] == 'Negative']['OriginalTweet'].str.len()
sns.distplot(deceptive, label='2')
deceptive = df[df['Sentiment'] == 'Extremely Positive']['OriginalTweet'].str.len()
sns.distplot(deceptive, label='3')
deceptive = df[df['Sentiment'] == 'Extremely Negative']['OriginalTweet'].str.len()
sns.distplot(deceptive, label='4')

plt.title('Distribution by Length')
plt.legend()
#plotting graph for non-digits.
truthful2 = df[df['Sentiment'] == 'Neutral']['OriginalTweet'].str.replace(r"\\w+",
").str.len()

```

```

sns.distplot(truthful2, label= 0 )
deceptive2 = df[df['Sentiment'] == 'Positive']['OriginalTweet'].str.replace(r'\w+',
").str.len()
sns.distplot(deceptive2, label= 1)
deceptive2 = df[df['Sentiment'] == 'Negative']['OriginalTweet'].str.replace(r'\w+',
").str.len()
sns.distplot(deceptive2, label= 2)
deceptive2 = df[df['Sentiment'] == 'Extremely
Positive']['OriginalTweet'].str.replace(r'\w+', ").str.len()
sns.distplot(deceptive2, label= 3)
deceptive2 = df[df['Sentiment'] == 'Extremely
Negative']['OriginalTweet'].str.replace(r'\w+', ").str.len()
sns.distplot(deceptive2, label= 4)
plt.title('Distribution of Non-Digits')
plt.legend()

```

Training model:

#!/pip install nltk

import nltk

nltk.download('stopwords')

import nltk

from nltk.corpus **import** stopwords

from nltk.stem.porter **import** PorterStemmer

import re

import string

remove whitespaces

df['OriginalTweet']=df['OriginalTweet'].str.strip()

lowercase the text

df['OriginalTweet'] = df['OriginalTweet'].str.lower()

#remove punctuation

punc = string.punctuation

table = str.maketrans("","",punc)

df['OriginalTweet']=df['OriginalTweet'].apply(**lambda** x: x.translate(table))

tokenizing each message

```

df['word_tokens']=df.apply(lambda x: x['OriginalTweet'].split(' '),axis=1)
# removing stopwords
df['cleaned_text'] = df.apply(lambda x: [word for word in x['word_tokens'] if word
not in stopwords.words('english')],axis=1)
# stemming
ps = PorterStemmer()
df['stemmed']= df.apply(lambda x: [ps.stem(word) for word in
x['cleaned_text']],axis=1)
# remove single letter words
df['final_text'] = df.apply(lambda x: ' '.join([word for word in x['stemmed'] if
len(word)>1]),axis=1)
# divide the set in training and test
from sklearn.model_selection import train_test_split
X,X_test,y,y_test =
train_test_split(df.loc[:, 'OriginalTweet'],df['Sentiment'],test_size=0.2)
# Now we'll create a vocabulary for the training set with word count
from collections import defaultdict
vocab=defaultdict(int)
for text in X['final_text'].values:
    for elem in text.split(' '):
        vocab[elem]+=1
#!/pip install wordcloud
from wordcloud import WordCloud
# Now we look at the types of words in ham and spam. We plot wordclouds for both
ham_text=' '.join(X.loc[y=='Neutral','final_text'].values)
ham_wordcloud =
WordCloud(background_color='white',max_words=2000).generate(ham_text)
spam_text=' '.join(X.loc[y=='Positive','final_text'].values)
spam_wordcloud =
WordCloud(background_color='white',max_words=2000).generate(spam_text)
plt.figure(figsize=[20,30])
plt.subplot(1,2,1)
plt.imshow(spam_wordcloud,interpolation='bilinear')

```

```
plt.title('Positive')
plt.axis('off')
plt.subplot(1,2,2)
plt.imshow(ham_wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Neutral')
```

MODULE 3:

Performance measurements of ML algorithms (Supervised):

```
#import library packages
import pandas as p
import numpy as n
import warnings
warnings.filterwarnings("ignore")
#Load given dataset
data = p.read_csv('Corona_NLP_train.csv',encoding='latin1')
df=data.dropna()
df.columns
from sklearn.preprocessing import LabelEncoder
var_mod = ['UserName', 'ScreenName', 'Location', 'TweetAt',
'OriginalTweet','Sentiment']
le = LabelEncoder()
for i in var_mod:
    df[i] = le.fit_transform(df[i]).astype(int)
#According to the cross-validated MCC scores, the random forest is the
best-performing model, so now let's evaluate its performance on the test set.
from sklearn.metrics import confusion_matrix, classification_report,
matthews_corrcoef, cohen_kappa_score, accuracy_score,
average_precision_score, roc_auc_score
X = df.drop(labels='Sentiment', axis=1)
#Response variable
y = df.loc[:, 'Sentiment']
#We'll use a test size of 30%. We also stratify the split on the response variable,
```

which is very important to do because there are so few fraudulent transactions.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=1, stratify=y)
```

Random Forest:

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
```

```
rfc.fit(X_train,y_train)
predictR = rfc.predict(X_test)
print("")
x = (accuracy_score(y_test,predictR)*100)
print('Accuracy result of Random Forest: is:', x)
print("")
```

```
print("")
print('Classification report of Random Forest: Results:')
print("")
```

```
print(classification_report(y_test,predictR))
xrf = (accuracy_score(y_test,predictR)*100)
```

```
cm2=confusion_matrix(y_test,predictR)
print('Confusion Matrix result of Random Forest: is:\n',
confusion_matrix(y_test,predictR))
print("")
sensitivity1 = cm2[0,0]/(cm2[0,0]+cm2[0,1])
print('Sensitivity : ', sensitivity1 )
print("")
specificity1 = cm2[1,1]/(cm2[1,0]+cm2[1,1])
print('Specificity : ', specificity1)
```

Logistic Regression:

```

from sklearn.linear_model import LogisticRegression
logR= LogisticRegression()
logR.fit(X_train,y_train)
predictR = logR.predict(X_test)
print("")
x = (accuracy_score(y_test,predictR)*100)
print('Accuracy result of Logistic Regression is:', x)
print("")

print("")
print('Classification report of Logistic Regression : Results:')
print("")

print(classification_report(y_test,predictR))
xl = (accuracy_score(y_test,predictR)*100)

cm2=confusion_matrix(y_test,predictR)
print('Confusion Matrix result of Logistic Regression : is:\n',
confusion_matrix(y_test,predictR))
print("")

sensitivity1 = cm2[0,0]/(cm2[0,0]+cm2[0,1])
print('Sensitivity : ', sensitivity1 )
print("")
specificity1 = cm2[1,1]/(cm2[1,0]+cm2[1,1])
print('Specificity : ', specificity1)

```

MODULE 4:

Implementation of LSTM model for training and testing:

```

from tensorflow.keras.models import Sequential,load_model
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.layers import Dropout, Activation, Flatten

```

```

import warnings
warnings.filterwarnings("ignore")
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
data = pd.read_csv('Corona.csv', encoding='latin-1')
data=data.dropna()
data.columns
from sklearn.preprocessing import LabelEncoder
var_mod = ['UserName', 'ScreenName', 'Location', 'TweetAt', 'OriginalTweet',
           'Sentiment']
le = LabelEncoder()
for i in var_mod:
    data[i] = le.fit_transform(data[i]).astype(int)
data.head()
from sklearn.metrics import confusion_matrix, classification_report,
matthews_corrcoef, cohen_kappa_score, accuracy_score,
average_precision_score, roc_auc_score
X = data.drop(labels='Sentiment', axis=1)
#Response variable
y = data.loc[:, 'Sentiment']
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5,
random_state=1, stratify=y)
scaler = MinMaxScaler()
data_training = scaler.fit_transform(X_train)
data_training
scaler = MinMaxScaler()
data_testing = scaler.fit_transform(X_test)
data_testing
data_training[0:10]
data_testing[0:10]
X_train1 = []

```

```

y_train1 = []
for i in range(60, data_training.shape[0]):
    X_train1.append(data_training[i-60:i])
    y_train1.append(data_training[i, 0])
X_train, y_train = np.array(X_train1), np.array(y_train1)
X_train.shape
X_test1 = []
y_test1 = []
for i in range(60, data_training.shape[0]):
    X_train1.append(data_training[i-60:i])
    y_train1.append(data_training[i, 0])
X_train, y_train = np.array(X_train1), np.array(y_train1)
X_train.shape
X_test1 = []
y_test1 = []
for i in range(60, data_testing.shape[0]):
    X_test1.append(data_testing[i-60:i])
    y_test1.append(data_testing[i, 0])
X_test, y_test = np.array(X_test1), np.array(y_test1)
X_test.shape
# Initialising the RNN
model = Sequential()
# Adding the first LSTM layer and some Dropout regularisation
model.add(LSTM(24, return_sequences=True, input_shape=(X_train.shape[1],
X_train.shape[2]))) # returns a sequence of vectors of dimension 64
model.add(Dropout(0.2))

# Adding a second LSTM layer and some Dropout regularisation
model.add(LSTM(units = 50))
#model.add(Dropout(0.2))

model.add(Dense(20, activation='relu'))

```

```

# Adding the output layer
#model.add(Dense(1, activation="linear"))
model.add(Dense(5, activation = 'softmax'))

#model.add(Dense(units = 1))
# Compiling the RNN

model.compile(optimizer = 'adam', loss =
'mean_squared_error',metrics=['accuracy'])

history=model.fit(X_train, y_train,batch_size=32, epochs=10)
# Model summary for number of parameters use in the algorithm
model.summary()
scores = model.predict(X_test)
import math, time
print("")
trainScore = model.evaluate(X_train, y_train, verbose=0)
print('Train Score: %.2f MSE (%.2f RMSE)' % (trainScore[0],
math.sqrt(trainScore[0])))
print("")
testScore = model.evaluate(X_test, y_test, verbose=0)
print('Test Score: %.2f MSE (%.2f RMSE)' % (testScore[0],
math.sqrt(testScore[0])))

```

MODULE 5:

Build a model for LSTM Neural Network:

```

import warnings
warnings.filterwarnings("ignore")
#!pip install keras-metrics
import tqdm

```

```

import numpy as np
import tensorflow
import keras_metrics # for recall and precision metrics
from keras.preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer
from keras.layers import Embedding, LSTM, Dropout, Dense
from keras.models import Sequential
from keras.utils import to_categorical
from keras.callbacks import ModelCheckpoint, TensorBoard
from sklearn.model_selection import train_test_split
import time
import numpy as np
import pickle

from keras.preprocessing.text import Tokenizer
SEQUENCE_LENGTH = 100 # the length of all sequences (number of words per sample)
EMBEDDING_SIZE = 100 # Using 100-Dimensional GloVe embedding vectors
TEST_SIZE = 0.20 # ratio of testing set

BATCH_SIZE = 64# number of epochs

# to convert labels to integers and vice-versa
#label2int = {"truthful": 0, "deceptive": 1} #added
#int2label = {0: "truthful", 1: "deceptive"} #added

label2int = {'Neutral':0, 'Positive':1, 'Negative':2, 'Extremely Positive':3, 'Extremely Negative':4} #added
int2label = {0:'Neutral', 1:'Positive', 2:'Negative', 3:'Extremely Positive',4:'Extremely Negative'} #added
import pandas as pd
data=pd.read_csv("Corona_NLP_train.csv",encoding='latin-1')
data.isnull().sum()
data.shape

```

```

df=data.dropna()
df.shape
df.isnull().sum()
df.Sentiment.unique()
x=df['OriginalTweet']
x
x=list(x)
x
df['Sentiment'] = df['Sentiment'].map({'Neutral':'Neutral', 'Positive':'Positive',
'Negative':'Negative', 'Extremely Positive': 'Positive', 'Extremely Negative':
'Negative'})
df['Sentiment'] = df['Sentiment'].replace({0:'Neutral', 1:'Positive', 2:'Negative'})
y=df['Sentiment']
y.unique()
y=list(y)
y
type(y)
# Text tokenization
# vectorizing text, turning each text into sequence of integers
tokenizer = Tokenizer(num_words=None,char_level=True,oov_token='UNK')
tokenizer.fit_on_texts(x)
# convert to sequence of integers
x = tokenizer.texts_to_sequences(x)
print(x[0])
# convert to numpy arrays
x = np.array(x)
y = np.array(y)
# pad sequences at the beginning of each sequence with 0's
# for example if SEQUENCE_LENGTH=4:
# [[5, 3, 2], [5, 1, 2, 3], [3, 4]]
# will be transformed to:
# [[0, 5, 3, 2], [5, 1, 2, 3], [0, 0, 3, 4]]
x = pad_sequences(x, maxlen=SEQUENCE_LENGTH)
# One Hot encoding labels

```

```

# [spam, ham, spam, ham, ham] will be converted to:
# [1, 0, 1, 0, 1] and then to:
# [[0, 1], [1, 0], [0, 1], [1, 0], [0, 1]]

y = [ label2int[label] for label in y ]
y = to_categorical(y)
print(y[0])
# split and shuffle
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=TEST_SIZE,
random_state=7)
def get_embedding_vectors(tokenizer, dim=100):
    embedding_index = {}
    with open(f'glove.6B.{dim}d.txt', encoding='utf8') as f:
        for line in tqdm.tqdm(f, "Reading GloVe"):
            values = line.split()
            word = values[0]
            vectors = np.asarray(values[1:], dtype='float32')
            embedding_index[word] = vectors

word_index = tokenizer.word_index
embedding_matrix = np.zeros((len(word_index)+1, dim))
for word, i in word_index.items():
    embedding_vector = embedding_index.get(word)
    if embedding_vector is not None:
        # words not found will be 0s
        embedding_matrix[i] = embedding_vector

return embedding_matrix
import tensorflow as tf
def get_model(tokenizer, lstm_units):
    """
    Constructs the model,
    Embedding vectors => LSTM => 2 output Fully-Connected neurons with
    softmax activation

```

```

"""

# get the GloVe embedding vectors
embedding_matrix = get_embedding_vectors(tokenizer)
model = Sequential()
model.add(Embedding(len(tokenizer.word_index)+1,
                    EMBEDDING_SIZE,
                    weights=[embedding_matrix],
                    trainable=False,
                    input_length=SEQUENCE_LENGTH))

model.add(LSTM(lstm_units, recurrent_dropout=0.2))
model.add(Dropout(0.3))
model.add(Dense(3, activation="softmax"))
# compile as rmsprop optimizer
# aswell as with recall metric
model.compile(optimizer="adam", loss="categorical_crossentropy",
              metrics=["accuracy", tf.keras.metrics.Precision(),
                      tf.keras.metrics.Recall()])
model.summary()
return model

# constructs the model with 128 LSTM units
model = get_model(tokenizer=tokenizer, lstm_units=128)
# initialize our ModelCheckpoint and TensorBoard callbacks
# model checkpoint for saving best weights
model_checkpoint = ModelCheckpoint("Covid_19_Sentiment_{val_loss:.2f}",
save_best_only=True, verbose=1)
# for better visualization
tensorboard = TensorBoard(f"Covid_19_Sentiment_{time.time()}")
# print our data shapes
print("X_train.shape:", X_train.shape)
print("X_test.shape:", X_test.shape)
print("y_train.shape:", y_train.shape)
print("y_test.shape:", y_test.shape)
# train the model

```

```

model.fit(X_train, y_train, validation_data=(X_test, y_test),batch_size=32,
epochs=100,verbose=1)
# get the loss and metrics
result = model.evaluate(X_test, y_test)
# extract those
loss = result[0]
accuracy = result[1]
precision = result[2]
recall = result[3]
print(f"[+] Accuracy: {accuracy*100:.2f}%")
print(f"[+] Precision: {precision*100:.2f}%")
print(f"[+] Recall: {recall*100:.2f}%")
def get_predictions(text):
    sequence = tokenizer.texts_to_sequences([text])
    # pad the sequence
    sequence = pad_sequences(sequence, maxlen=SEQUENCE_LENGTH)
    # get the prediction
    prediction = model.predict(sequence)[0]
    # one-hot encoded vector, revert using np.argmax
    return int2label[np.argmax(prediction)]
text=str(input("enter the statement: "))
print(get_predictions(text))

```

OUTPUT SCREENSHOTS:

MODULE 1:

Screen Shot 1:

```
#To describe the dataframe
df.describe()
```

	UserName	ScreenName
count	32567.000000	32567.000000
mean	24418.499309	69370.499309
std	11899.886650	11899.886650
min	3799.000000	48751.000000
25%	14002.500000	58954.500000
50%	24576.000000	69528.000000
75%	34747.500000	79699.500000
max	44955.000000	89907.000000

Screen Shot 2:

```
#Checking datatype and information about dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 32567 entries, 0 to 41156
Data columns (total 6 columns):
#   Column          Non-Null Count  Dtype
---  -
0   UserName        32567 non-null  int64
1   ScreenName      32567 non-null  int64
2   Location        32567 non-null  object
3   TweetAt        32567 non-null  object
4   OriginalTweet   32567 non-null  object
5   Sentiment       32567 non-null  object
dtypes: int64(2), object(4)
memory usage: 1.7+ MB
```

Screen Shot 3:

```
#Checking sum of missing values
df.isnull().sum()
```

```
UserName      0
ScreenName    0
Location      0
TweetAt       0
OriginalTweet  0
Sentiment     0
dtype: int64
```

Screen Shot 4:

	UserName							
	count	mean	std	min	25%	50%	75%	max
Sentiment								
Extremely Negative	4249.0	22964.916686	11968.753390	3819.0	12502.00	22239.0	33171.00	44932.0
Extremely Positive	5273.0	25031.953916	11829.939310	3810.0	15059.00	25198.0	35418.00	44941.0
Negative	7763.0	23905.016360	11873.688714	3808.0	13493.50	23894.0	34062.00	44955.0
Neutral	6172.0	25063.771387	11934.902441	3799.0	14660.25	25667.0	35554.75	44951.0
Positive	9110.0	24741.778705	11832.755683	3800.0	14422.25	25046.0	35069.50	44949.0

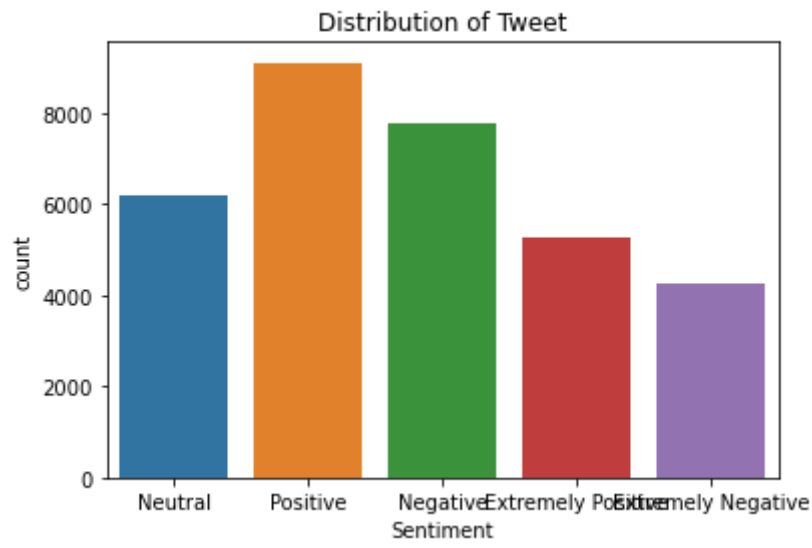
Screen Shot 5:

```
df.head()
```

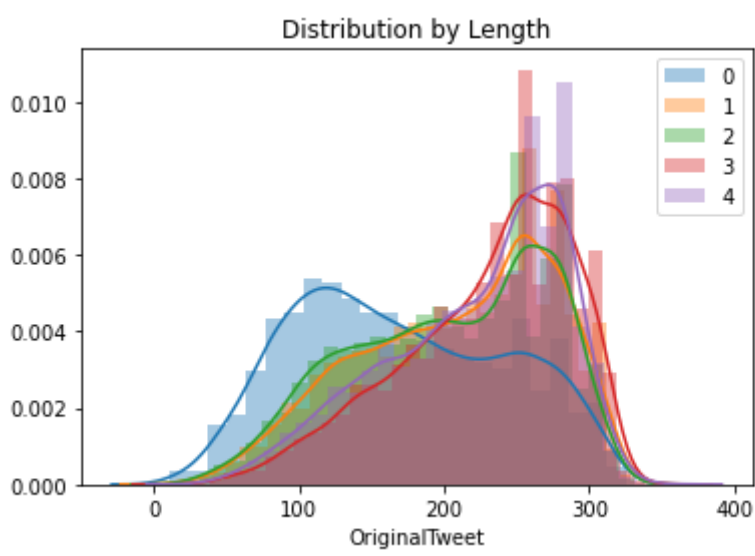
	UserName	ScreenName	Location	TweetAt	OriginalTweet	Sentiment
0	3799	48751	London	16-03-2020	@MeNyrbie @Phil_Gahan @Chrisitv https://t.co/i...	Neutral
1	3800	48752	UK	16-03-2020	advice Talk to your neighbours family to excha...	Positive
2	3801	48753	Vagabonds	16-03-2020	Coronavirus Australia: Woolworths to give elde...	Positive
5	3804	48756	ÃT: 36.319708,-82.363649	16-03-2020	As news of the regionÃs first confirmed COVID...	Positive
6	3805	48757	35.926541,-78.753267	16-03-2020	Cashier at grocery store was sharing his insig...	Positive

MODULE 2:

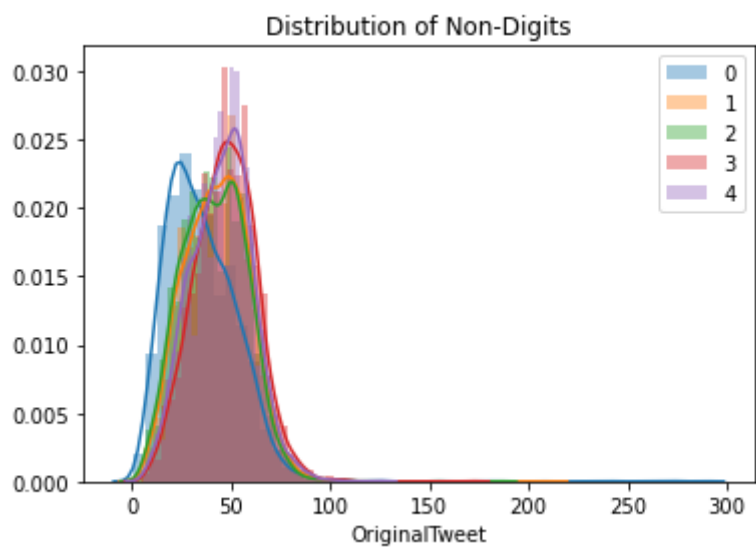
Screen Shot 1:



Screen Shot 2:



Screen Shot 3:



Screen Shot 4:



Screen Shot 5:



MODULE 3:

Screen Shot 1:

Accuracy result of Random Forest: is: 23.784668918227407

Classification report of Random Forest: Results:

	precision	recall	f1-score	support
0	0.16	0.09	0.12	1275
1	0.18	0.13	0.15	1582
2	0.24	0.28	0.26	2329
3	0.21	0.17	0.19	1852
4	0.29	0.38	0.33	2733
accuracy			0.24	9771
macro avg	0.21	0.21	0.21	9771
weighted avg	0.23	0.24	0.23	9771

Confusion Matrix result of Random Forest: is:

```
[[ 119  150  388  189  429]
 [ 117  207  426  218  614]
 [ 189  277  650  357  856]
 [ 138  222  515  316  661]
 [ 192  309  768  432 1032]]
```

Sensitivity : 0.4423791821561338

Specificity : 0.6388888888888888

Screen Shot 2:

Accuracy result of Logistic Regression is: 27.76583768293931

Classification report of Logistic Regression : Results:

	precision	recall	f1-score	support
0	0.00	0.00	0.00	1275
1	0.00	0.00	0.00	1582
2	0.26	0.05	0.08	2329
3	0.00	0.00	0.00	1852
4	0.28	0.95	0.43	2733
accuracy			0.28	9771
macro avg	0.11	0.20	0.10	9771
weighted avg	0.14	0.28	0.14	9771

Confusion Matrix result of Logistic Regression : is:

```
[[ 0  0 59  0 1216]
 [ 0  0 60  0 1522]
 [ 0  0 109  0 2220]
 [ 0  0 66  0 1786]
 [ 0  0 129  0 2604]]
```

MODULE 4:

Screen shot 1:

Model: "sequential_5"

Layer (type)	Output Shape	Param #
lstm_10 (LSTM)	(None, 60, 24)	2880
dropout_5 (Dropout)	(None, 60, 24)	0
lstm_11 (LSTM)	(None, 50)	15000
dense_10 (Dense)	(None, 20)	1020
dense_11 (Dense)	(None, 5)	105

=====
Total params: 19,005
Trainable params: 19,005
Non-trainable params: 0
=====

MODULE 5:

Screen Shot 1:

```
# constructs the model with 128 LSTM units
model = get_model(tokenizer=tokenizer, lstm_units=128)
```

```
Reading GloVe: 400000it [00:13, 29053.31it/s]
```

```
Model: "sequential_2"
```

Layer (type)	Output Shape	Param #
embedding_2 (Embedding)	(None, 100, 100)	11600
lstm_2 (LSTM)	(None, 128)	117248
dropout_2 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 3)	387
Total params: 129,235		
Trainable params: 117,635		
Non-trainable params: 11,600		

Screen shot 2:

```
X_train.shape: (26053, 100)
```

```
X_test.shape: (6514, 100)
```

```
y_train.shape: (26053, 3)
```

```
y_test.shape: (6514, 3)
```

```
Train on 26053 samples, validate on 6514 samples
```

```
Epoch 1/100
```

```
26053/26053 [=====] - 81s 3ms/step - loss: 1.0511 - accuracy: 0.4265 - precision_1: 0.4424 - recall_1: 0.0833 - val_loss: 1.0414 - val_accuracy: 0.4381 - val_precision_1: 0.4427 - val_recall_1: 0.0467
```

```
Epoch 2/100
```

```
26053/26053 [=====] - 78s 3ms/step - loss: 1.0432 - accuracy: 0.4391 - precision_1: 0.4503 - recall_1: 0.0371 - val_loss: 1.0420 - val_accuracy: 0.4461 - val_precision_1: 0.4506 - val_recall_1: 0.0296
```

```
Epoch 3/100
```

```
26053/26053 [=====] - 88s 3ms/step - loss: 1.0424 - accuracy: 0.4398 - precision_1: 0.4527 - recall_1: 0.0261 - val_loss: 1.0383 - val_accuracy: 0.4435 - val_precision_1: 0.4544 - val_recall_1: 0.0236
```

```
Epoch 4/100
```

```
26053/26053 [=====] - 84s 3ms/step - loss: 1.0386 - accuracy: 0.4405 - precision_1: 0.4552 - recall_1: 0.0214 - val_loss: 1.0210 - val_accuracy: 0.4441 - val_precision_1: 0.4568 - val_recall_1: 0.0209
```

Screen Shot 3:

```
text=str(input("enter the statement: "))
```

enter the statement: Feeling that the battles with the coronavirus/covid-19 will take a long time to end. More or less, a year. Anyway, necessary to develop the faster testing apparatus and vaccine.

```
print(get_predictions(text))
```

Negative

REFERENCES:

- [1]. TING DA AND LIANG YANG, " Local COVID-19 Severity and Social Media Responses: Evidence From China", Received October 9, 2020, accepted October 30, 2020, date of publication November 10, 2020, date of current version November 20, 2020.

- [2]. ZIJIAN LONG¹ , RAJWA ALHARTHI¹, " NeedFull – A Tweet Analysis Platform to study Human Needs during the COVID-19 pandemic in New York State".

- [3]. Mabrook S. Al-Rakhami^{1,2}, " Lies Kill, Facts Save: Detecting COVID-19 Misinformation in Twitter".

- [4]. Manoj Sethi* , Sarthak Pandey, " Sentiment Identification in COVID-19 Specific Tweets ".

- [5]. A. Mourad* , A. Srour, " Critical Impact of Social Networks Infodemic on Defeating Coronavirus COVID-19 Pandemic: Twitter-Based Study and Research Directions ".