CHAPTER 1
INTRODUCTION

INTRODUCTION

With the elevation in the communication technology i.e. World Wide Web, a huge number of people from all lineages across the world take part in social network. According to the internet statistics 2019, there are 4.1 billion internet users in the world. The online medium has become a significant way for people to express their opinion and with social media, there is an abundance of opinion information available.

Sentiment analysis or opinion mining is the computational study of people's opinions, sentiments, emotions, appraisals, and attributes towards entities such as products, services, organizations, individual's, issues, events, topics, and their attributes. The rapid growth of the field coincides with those of the social media on web. Since early 2000, sentiment analysis has grown to be one of the most active research areas in natural language processing (NLP). Sentiment analysis aims to determine the attitude of a speaker or a writer with respect to some topic or the overall contextual polarity of the document. The attitude may be his or her judgment or the intended emotional communication towards the context. Sentiment analysis is the process of determining the polarity i.e. Neutral, Positive, Negative feeling toward a subject. Humans have the ability to sense the sentiment easily; however, it is time consuming, inconsistent, and costly in a business context. Humans have limited capacity to work. Hence, it's just not realistic to have people individually read millions of user review and score them for sentiment.

For example, if we consider Semantria's cloud-based sentiment analysis software, which extracts the sentiment of a document and its component through the following steps:

- A document is broken down into its basic parts of speech which defines the structural elements of a document, paragraph, or sentence.

- Now some sentiment-bearing phrases such as "**Worst-service**", are identified using specially designed algorithm.
- Each sentiment-bearing phrase is given a score based on a logarithmic scale between -10 and 10.
- Finally, the scores are summed up to calculate the overall sentiment of the document or sentence. The scores of the document range between -2 and 2.

This Semantria;s cloud-based sentiment analysis software delivers you more accurate and consistent results than two humans.

Fig 1.1.1 Tasks in Sentiment Analysis

Researchers have mainly studied sentiment analysis at three levels of granularity:

- Document level
- Sentence level
- Aspect level

Document level sentiment classification classifies an opinionated document as expressing an overall positive, negative or neutral opinion. It considers the whole

document as the basic information unit and assumes that the document is known to be opinionated and contain opinions about a single entity.

Sentence level sentiment classification classifies individual sentences in a document. However, each sentence cannot be assumed to be opinionated. Generally, one must classify the sentence as opinionated or not opinionated, which is called Subjectivity classification. Then the opinionated sentences are expressed as either positive or negative opinions. In this we have three-classification problem, that is, to classify a sentence as positive, negative or neutral.

Over the two, the document and the sentence level sentiment analysis this aspect level sentiment analysis is finely grained. Its core task is to extract and summarize people's opinions on entities and aspects/features of entities called targets. The aspect level analysis involves few steps which are aspect extraction, entity extraction, and aspect sentiment classification. For example, from the sentence, "At Novotel the food quality is good, but the service is worst", entity extraction should identify "Novotel" as the entity and aspect extraction must identify the "food quality" and "the service" as the two aspects. Aspect classification should classify the sentiment expressed on the "food quality" and "the service" as "positive" and "negative".

1.2 OBJECTIVE

Sentiment classification is a way to analyze the subjective information in the text and then mine the opinion. Sentiment analysis is the procedure by which information is extracted from the opinions, appraisals and emotions of people about entities, events and their attributes. In decision making, the opinions of others have a significant effect on customers ease, making choices with regards to online shopping, choosing events, products, entities.

1.3 PROPSED SYSTEM

With the growing development of online e-commerce sites, massive text information that contains people's views, feeling, and comments on products have emerged on network medium. The task of finding good and effective approach of mining and analysis on these data is a very important research in Natural Language Processing.

At present, sentiment analysis based on statistical machine learning methods has yielded good results in various applications. However, the methods are very simple, which lead to their generalization ability to handle with the complex classification problem and the ability to express the complex function to be restricted to a certain extent under the condition of limited samples and computational units.

The advancement of internet technology and machine learning techniques in information retrieval make Sentiment Analysis becomes popular among researchers. Besides, the emergent of social networking and blogs as a communication medium also contributes to the development of research in this area. Sentiment analysis or mining refers to the application of Natural Language Processing, Computational Linguistics, and Text Analytics to identify and extract subjective information in source materials. Machine Learning is commonly used to classify sentiment from text.

The proposed system deals with building a sentiment analysis model on product reviews using Recurrent Neural Network that overcomes the limitation of the simple machine learning methods. The aim of this project is to compare machine learning method such as Random Forest, KNN to the RNN model in terms of accuracy. This proposed system helps the industries to know the opinions of customers on their product and improve the quality of their products.

CHAPTER 2
LITERATURE SURVEY

LITERATURE SURVEY

Paper 1: Yuling Chen and Zhi Zhang. "Research on text sentiment analysis based on CNNs and SVM", 2018 13th IEEE Conference on Industrial Electronics and Applications (ICIEA).

This paper explains about that the traditional text sentiment analysis is based on emotion dictionary. In this paper they proposed a Convolutional Neural Networks (CNNs) model combined with SVM for text sentiment analysis. The results shows that the proposed system improves the accuracy of text sentiment analysis when compared with the traditional CNN, and confirms the effectiveness of sentiment analysis based on CNNs and SVM. This paper combines the advantages of the SVM and CNN together and constructs a text sentiment analysis where CNN is used as an automatic feature and the final text classifier will be SVM. This model have effectively improved performance of the text analysis.

Paper 2: Neha Raghuvanshi and Prof. J.M. Patil, "A Brief Review on Sentiment Analysis", International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT) - 2016

This paper presents a brief review on sentiment analysis. In this they have explained with different types of the sentiment analysis which are Document-level, Sentence – level, and Word-level sentiment analysis. They initially started discussing about the importance and what is sentiment analysis. They also have proposed different methods that are proposed by the researchers. Different methods are used in different researches process. Different types of sentiment analysis model based on word, sentence, feature and document is also discussed and found that document level model is more promising for better results. This paper proposes the naive bayes method to sentiment analyse the text.

Paper 3: Qiongxia Huang, Xianghan Zheng, Riqing Chen, and Zhenxin Dong, "Deep Sentiment Representation Based on CNN and LSTM" , in 2017 International Conference on Green Informatics.

This paper explains us that the traditional methods such as the SVM, random forest and other machine learning methods are applied to text sentiment analysis which have poor generalization ability in terms of complex classification problem. As RNN and CNN are two important models in the document and sentence level sentiment analysis. So, this paper proposes a model of capturing deep sentiment representation based on CNN and long short term memory LSTM. This proposed model in the paper uses the pre-trained word vector such as WordVect. The model employs CNN to gain significant local features of the text, then features are fed to two-layer LSTMs, which can extract context-dependent features and generate sentence representation for sentiment classification. We evaluate the proposed model by conducting a series of experiments on dataset.

Paper 4: Vikas Malik. And Amit Kumar, "Analysis of Twitter Data Using Deep Learning Approach: LSTM," in International Journal on Recent and Innovation Trends in Computing and Communication.

This papers gives the analysis such as the comparison of accuracy of the traditional naïve bayes model with the LSTM method. Initially this discus about the sentiment analysis and then dives into the discussing about the sentiment analysis with LSTM. They have mentioned step by step way to build a proper model using LSTM. They provided the source of the data, discussed about pre-processing the data, and about the modules used such as Tensor flow. Finally they have discussed about the analysis of LSTM that is the problem of long term dependencies, usage of LSTM in RNN, and the metrics to calculate the accuracy. Besides all these this paper also compares

the LSTM model to the Naïve Bayes model and prove that the accuracy, F-Score, precision and the recall of the LSTM is greater than the Naïve Bayes.

Paper 5: Abdullah Aziz Sharfuddin, Md Nafis Tihami and Md Saiful Islam, " A Deep Recurrent Neural Network with BiLSTM model for Sentiment Classification" in International Conference on Bangla Speech and Language Processing

This paper explains initially about the sentiment analysis. Then they have little bit introduced about Recurrent Neural Network and the Long Short Term Memory. In this paper they have discussed about the Bidirectional LSTM. In unidirectional LSTM information flows from the backward to forward. On the contrary in Bi-directional LSTM information not only flows backward to forward but also forward to backward using two hidden states. Hence Bi-LSTMs understand the context better. Finally they have compared the accuracy of the Bi-LSTM to the accuracies of the Decision Tree classifier and the SVM.

CHAPTER 3

SYSTEM ANALYSIS

SYSTEM ANALYSIS

REQUIREMENTS:

3.1 Functional Requirements:

The functional requirement of the system defines a function of software system or its components. A function is described as set of inputs, behaviour of a system and output.

- Fast and efficient
- Simple Computation

3.2 Non-Functional Requirements:

In systems engineering and requirements engineering, a non-functional requirement (NFR) is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviours. They are contrasted with functional requirements that define specific behaviour or functions.

- Response time: Time taken to get the result
- Memory Usage: Total amount of memory used by system.

3.3 Software requirements:

Operating System	: Window, Linux
IDE	: Visual Studio Code
Programming Languages	: Python

3.4 Hardware Requirements:

RAM	: 8 GB
Memory	: 500 GB
Processor	: Intel core i7
GPU	: Nvidia 940M / AMD

3.5 MODULES:

- **Tensorflow**

  Tensorflow is a free and open-source software library for dataflow ad differentiable programming across a range of tasks. It is a symbolic math library and is also used for machine learning application such as neural networks. Tensorflow was developed by the Google Brain team for internal Google use. TensorFlow's high-level Python API for building and training deep learning models. It was built to run on multiple CPUs or GPUs and even mobile operating systems, and it has several wrappers in several languages like Python, C++ or Java. Tensorflow architecture works in three parts: Preprocessing the data, Build the model, Train and estimate the model. This project uses the tensorflow as the backend.

  The tensorflow package can be installed as follows:

```
pip install tensorflow
```

  The basic data flow graph is as follows:

```
Import tensorflow as tf

X_1 = tf.placeholder(tf.float32, name = "X_1")
X_2 = tf.placeholder(tf.float32, name = "X_2")

multiply = tf.multiply(X_1, X_2, name = "multiply")

with tf.Session() as session:
      result = session.run(multiply, feed_dict={X_1:[1,2,3],
      X_2:[4,5,6]})
      print(result)
```

Fig 3.5.1 Basic Dataflow graph using Tensorflow

- **Keras**

  Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the

least possible delay is key to doing good research. It allows easy and faster prototyping. It runs seamlessly on CPU and GPU.

Keras has few methods that help in pre-processing the dataset and preparing it for the model. The pre-processing methods such as Tokenizer, text_to_word_sequnece are present in keras that tokenize, vectorize a text corpus and converts a text to a sequence of words (or tokens).

Keras helps in building the deep learning models simply with the help of keras.models. Sequential and adding keras.layers such as Dense, Embedding..etc to the defined model.

The package is installed by the following command:

```
pip install keras
```

This is the basic Sequential model built using keras.

```
From keras.model import Sequntial
From keras.layers import Dense

model = Sequential()
model.add(Dense(units=64,activation='relu',input_dim = 100))
model.add(Dense(units=10, activation='softmax' ))

model.compile(loss="categorical_crossentropy",optimizer='sgd',
              metrics=['accuracy']))

model.fit(x_train, y_train, epochs=5, batch_size=32)
```

Fig 3.5.2 Basic sequential model using keras

- **Sklearn**

Scikit-learn is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy. Scikit learn contains all the necessary methods to build machine learning tasks.

The package is installed by the following command:

```
pip install sklearn
```

For example, let's implement KNN Algorithm

```
From sklearn import neighbors
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,random_state=0)

knn = neighbors.KNeighborsClassifier(n_neighbors=5)

y_pred = knn.predict_proba(X_test))

knn.score(X_test, y_test)
```

Fig 3.5.3 Basic ML model using Sklearn

- **NumPy**

  NumPy is the fundamental package for scientific computing with Python. It contains among other things:

  - a powerful N-dimensional array object
  - sophisticated (broadcasting) functions
  - tools for integrating C/C++ and Fortran code
  - useful linear algebra, Fourier transform, and random number capabilities

  The NumPy can be installed as follows:

```
pip install numpy
```

- **Pandas**

  pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language. pandas is a NumFOCUS sponsored project. This will help ensure the success of development of pandas as a world-class open-source project and makes it possible to donate to the project. Pandas have data structures such as Dataframes, Series and Panels.

  The pandas can be installed as follows:

```
pip install pandas
```

- **Tkinter**

  Tkinter is a Python binding to the Tk GUI toolkit. It is the standard Python interface to the Tk GUI toolkit and is Python's de facto standard GUI. Tkinter is included with standard Linux, Microsoft Windows and Mac OS X installs of Python. The name Tkinter comes from Tk interface.

Fig 3.5.4 Tkinter GUI Programming

CHAPTER 4
SYSTEM DESIGN

SYSTEM DESIGN

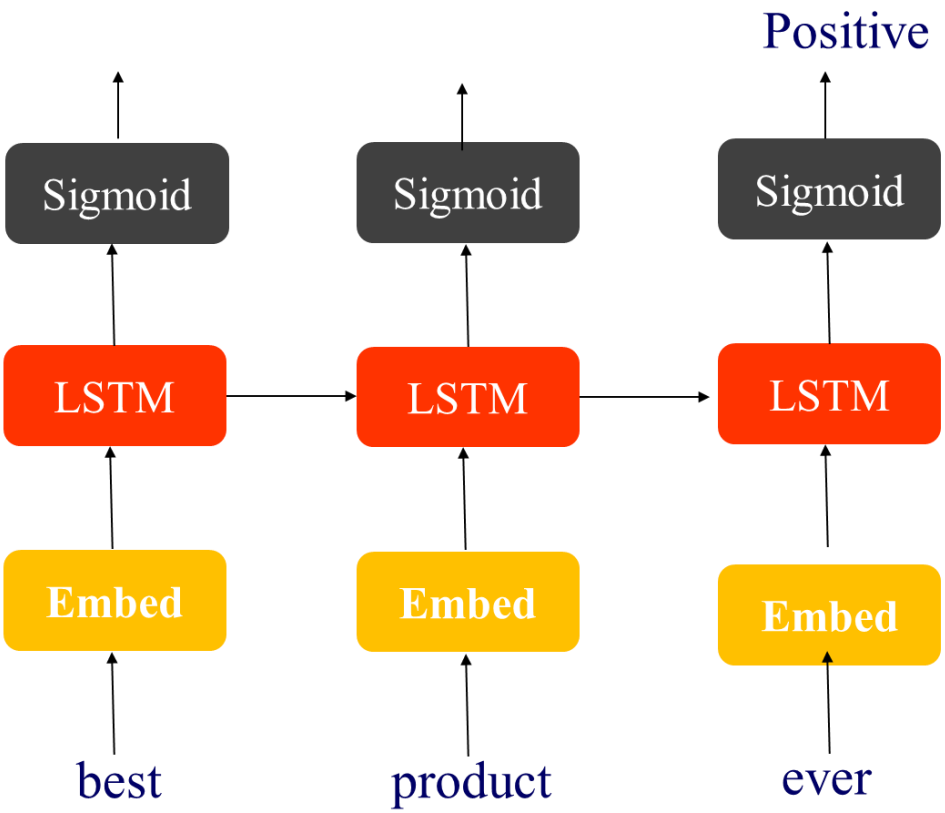
4.1 INTRODUCTION

As this application falls under the category of business, the application must show each minute data required to the customer to analyse and improve the quality of the product. So, the application must be kept simple. It must visualize the data using graph such as bar-graphs, and the accuracy metrics must be shown.

4.2 ARCHITECTURE

The basic architecture of the recurrent neural network has three layers. The three layers in the recurrent neural network are Embedding, Dense, and LSTM layer.

Fig 4.2.1 Architecture of Recurrent Neural Network



4.3 DESIGN

- **User Interface Design**

  The UI part of this application is completely designed using Visual Studio IDE in python. The user interface must be simple to use. The user interface must have all the basic requirements of the client such as the visualization of the data, number of words classified under different classes like positive, negative and neutral.

- **System Design**

  The system must be designed to get more accuracy than the accuracy of the machine learning methods. So, the layer used while designing the Recurrent Neural Network must be taken care.

  Recurrent neural networks (RNN) are a type of network which form memory through recurrent connections. In feed forward networks, inputs are independent of each other. But in RNN, all inputs are connected to each other. This lets the network to exhibit vigorous temporal behavior for a time sequence which makes it favorable for sequential classification like sentiment analysis. As it can be seen in the figure, at first,

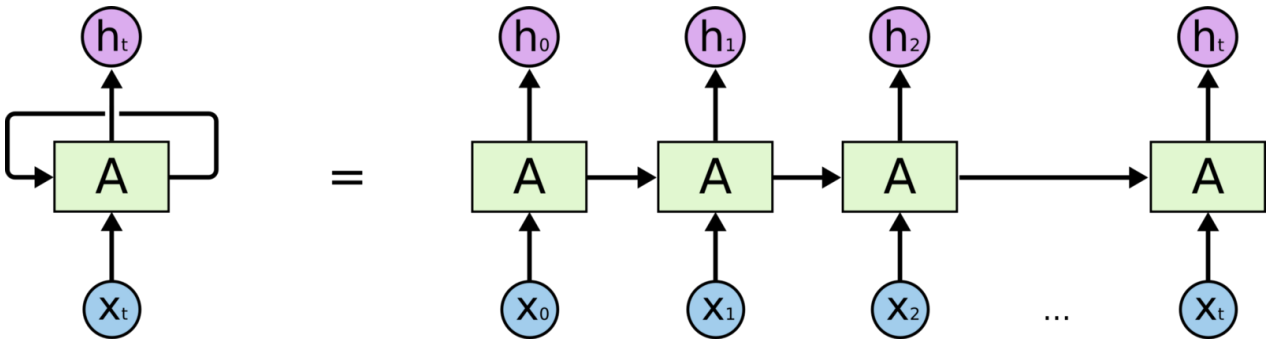


Fig. 4.3.1 RNN loop

It takes the x0 from the sequence of input and then it outputs h0 which together with x1 is the input for the next step. So, the h0 and x1 is the input for the next

step. Similarly, h1 from the next is the input with x2 for the next step and so on. This way, it keeps remembering the context while training.

$$h_t = f(W_{hx}x_t + W_{hh}h_t{-}1 + b)$$

$$y_t = g(W_{yh}h_t + c)$$

- **LSTM**

Long Short-Term Memory Units In 1997, an alteration of RNN with Long Short-Term Memory units, or LSTM units, was proposed by the Sepp Hochreiter and Juergen Schmidhuber. Some errors backpropagate through time in general RNN. These LSTM units help to bypass these errors. While keeping a more consistent error, they let RNNs keep on learning over several time steps. LSTMs consist of information outside the basic flow of the rnn in a valved block. Neural network's notes get triggered by the notes they get. Likewise, LSTM's gates pass on or block the data based on its weight. After that, these signals are grated with their own sets of weights. Subsequently, RNN's learning process modify these weights that control hidden states and input. Ultimately, these cells learn when to let information to get in, get out or be deleted through the consequent steps of taking guesses, back propagating error, and modulating weights via gradient descent.

$$f_t = \sigma_g(W_f x_t + U_f h_t{-}1 + b_f)$$

$$i_t = \sigma_g(W_i x_t + U_i h_t{-}1 + b_i)$$

$$o_t = \sigma_g(W_o x_t + U_o h_t{-}1 + b_o)$$

$$c_t = f_t o c_{t-1} + i_t o \sigma_c(W_c x_t + U_c h_{t-1} + b_c)$$

$$h_t = o_t o \sigma_h(c_t)$$

- **Dense Layer**

A dense layer is just a regular layer of neurons in a neural network. Each neuron recieves input from all the neurons in the previous layer, thus densely

connected. The layer has a weight matrix W, a bias vector b, and the activations of previous layer.

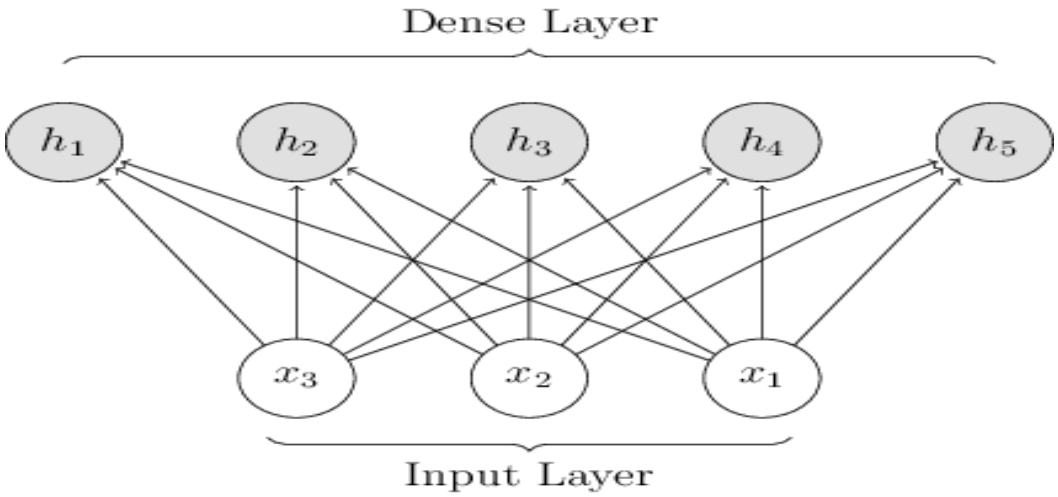


Fig. 4.3.2 Dense Layer

- **Embedding Layer**

The Embedding layer has weights that are learned. If you save your model to file, this will include weights for the Embedding layer. The output of the Embedding layer is a 2D vector with one embedding for each word in the input sequence of words. An embedding is a relatively low-dimensional space into which you can translate high-dimensional vectors. Embedding's make it easier to do machine learning on large inputs like sparse vectors representing words.

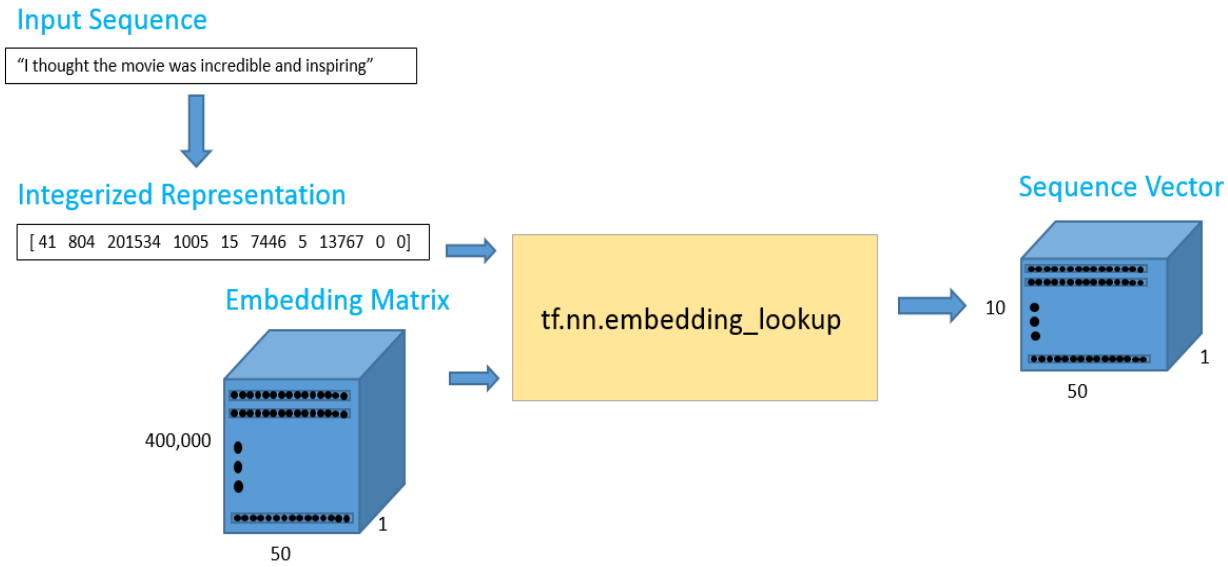


Fig. 4.3.3 Embedding Layer

4.4 UML DIAGRAMS

The Unified Modelling Language is a general-purpose, developmental, modelling language in the field of software engineering that is intended to provide a standard way to visualize the design of a system. It was developed by Grady Booch, Ivar Jacobson and James Rumbaugh at Rational Software in 1994-1995, with further development led by them through 1996.

- **Use-case Diagram:**

  A use case diagrams is a graphic depiction of the interactions among the elements of a system. A use case diagram is one of the diagram that is used for modelling the system. For modelling a system, the most important aspect is to capture the dynamic behaviour as only static behaviour is not enough to model a system. A use case is a methodology used in system analysis to identify, clarify, and organize system requirements. The internal and external agents are called Actors. This diagram is used for easily understanding the functionality of the system.

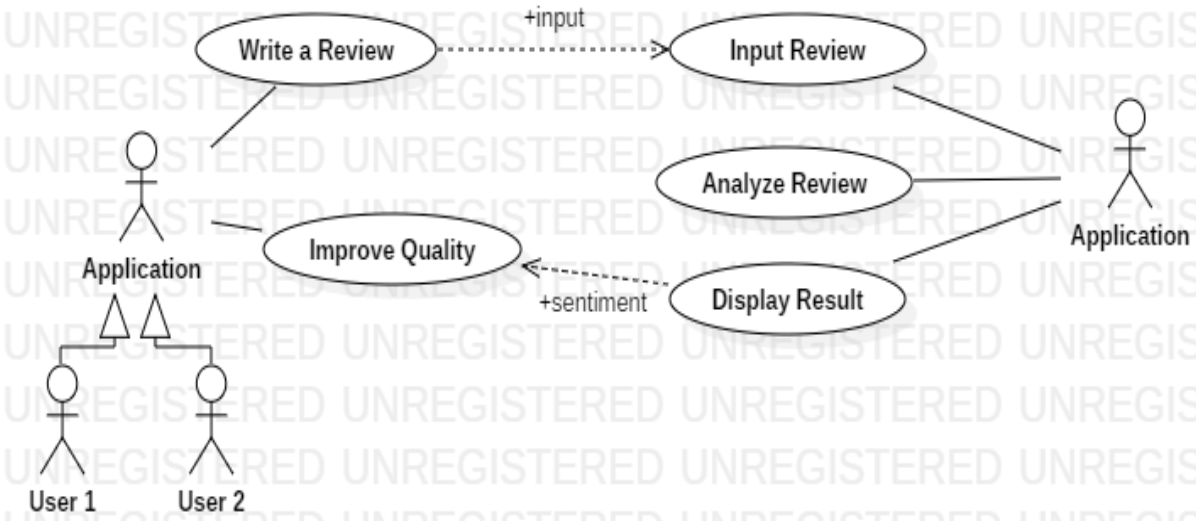


Fig 4.4.1 Use Case diagram

- **Class Diagram:**

In UML, a class diagram is a type of static describes the structure of a system showing the system's classes, their attributes, operations and the relationship among objects. Class diagram is used for general conceptual modelling of the systematic of the application, and for detailed modelling translating the models into programming code. Classes in the class diagrams are represented with boxes that have three compartments:

- The top compartment contains the name of the classes. It is printed in bold and the first letter is capitalized. Here the classes directly represent a real time object.
- The middle compartment contains the attributes of the class. They are left-aligned, and the first letter is lowercase. These attributes represent the characteristics of the real time objects.
- The bottom compartment contains the operations the class can execute. They are also left-aligned, and the first letter is lowercase. The

operations describe the operations a real time objects performs.

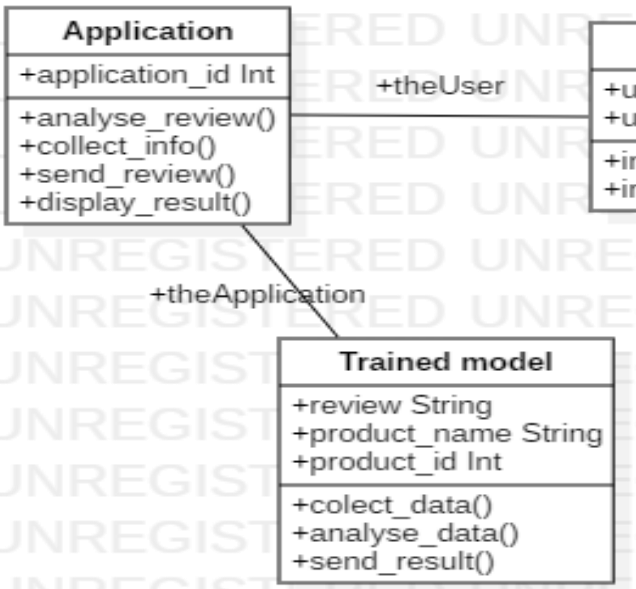


Fig 4.4.2 Class Diagram

- **Sequence Diagram:**

  A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence Diagrams are also called as Event Diagram or Event Scenarios. A sequence diagram shows, as parallel vertical lines, different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

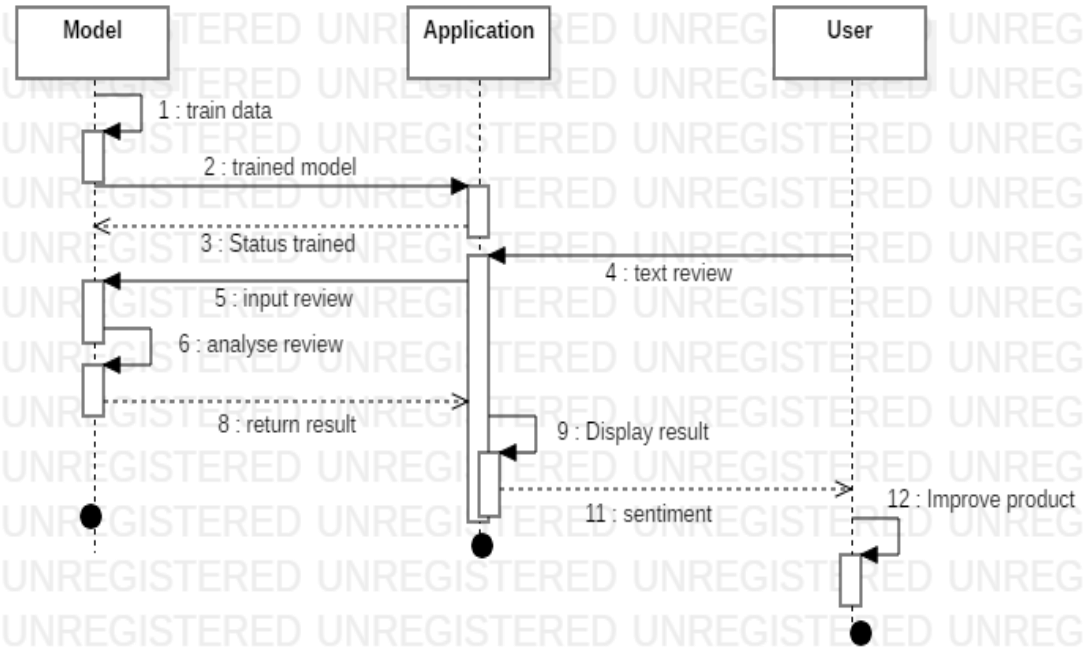


Fig 4.4.3 Sequence Diagram

- **DFD (Data Flow Diagram):**

  A data flow diagram (DFD) illustrates the flow and transformation of data for a business process. It's a visual representation of how data flows through a system. A DFD shows what kind of information will be input to and output from the system, how the data will advance through the system, and where the data will be stored. It does not show information about process timing or whether processes will operate in sequence or in parallel, unlike a traditional structured flowchart, or a UML activity workflow diagram, which presents both control and data flow.

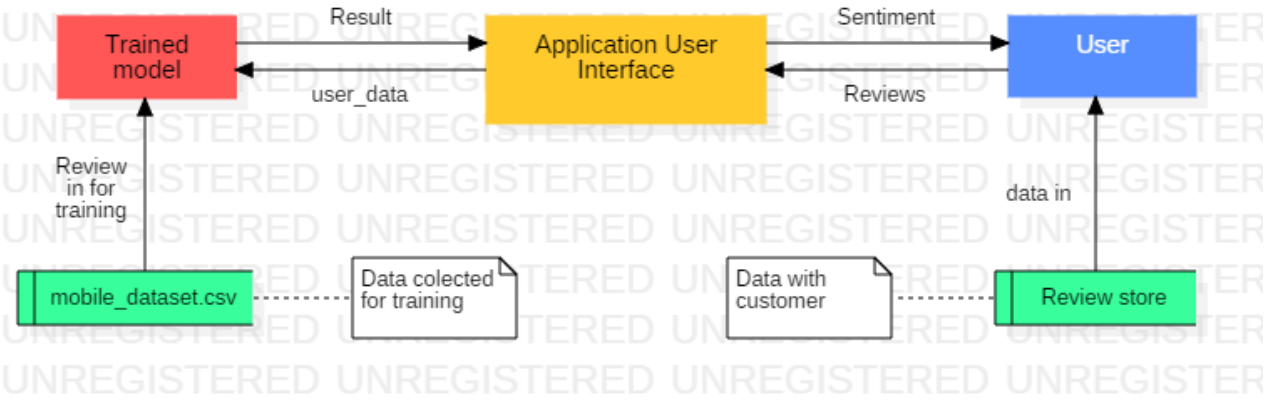


Fig 4.4.4 Data Flow Diagram

The basic parts of DFD's are Process, files or database, input, and flows.

- A process is a business activity or function where the manipulation and transformation of data takes place. It is represented by a circle.
- A Data Store represents the storage of persistent data required and produced by the process.
- An external entity can represent a human, system or subsystem.
- A Data flow represents the flow of information, with its direction represented by an arrow head that shows at the end of flow connector.

CHAPTER 5
IMPLEMENTATION

IMPLEMENTATION

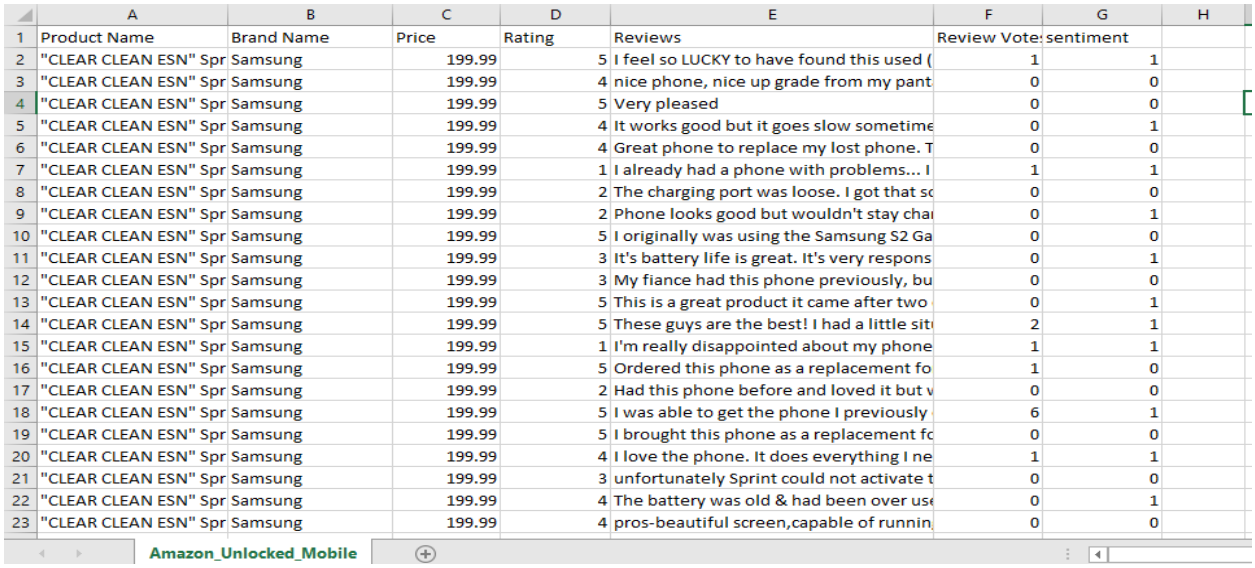
The last phase in software development cycle is the implementation phase. In this the application is implemented and deployed. The main aim of the application is to develop a recurrent neural network that uses the customer product review data and analyze the sentiment. This application is built in python.There are few process involved in implementing such as data collection, data pre-processing and training model.

5.1 Data:

To gather the data many options are possible. We can write a program to collect automatically a corpus of product reviews based on two classes, “positive”, and “negative”. After many searches I found a dataset of 4,32,120 product reviews in English coming from sources Kaggle. It is composed seven columns that are Product Name, Brad Name, Price, Rating, Reviews Review Rating and sentiment. We are only interested by the Sentiment column, product review and the product name. The value of the sentiment column is 0 if the product review is negative and 1 if the review is positive. The product name gives the information about the company of the product. The product is the review we are most concerned about as this is what we use to analyse the sentiment of the review.

There are some particularities and difficulties that we are going to encounter during pre-processing steps.

- The presence of acronyms “bf” or more complicated “APL”. Does it means apple?
- The presence of sequences of repeated characters such as “Juuuuuuuuuuu”, “hmmmm”. In general, we repeat words to emphasize it.
- The presence of emotions “:O”
- Spelling mistakes and urban grammar like “im gonna”



	A	B	C	D	E	F	G	H
1	Product Name	Brand Name	Price	Rating	Reviews	Review Vote:	sentiment	
2	"CLEAR CLEAN ESN" Spr	Samsung	199.99	5	I feel so LUCKY to have found this used (	1	1	
3	"CLEAR CLEAN ESN" Spr	Samsung	199.99	4	nice phone, nice up grade from my pant	0	0	
4	"CLEAR CLEAN ESN" Spr	Samsung	199.99	5	Very pleased	0	0	
5	"CLEAR CLEAN ESN" Spr	Samsung	199.99	4	It works good but it goes slow sometime	0	1	
6	"CLEAR CLEAN ESN" Spr	Samsung	199.99	4	Great phone to replace my lost phone. T	0	0	
7	"CLEAR CLEAN ESN" Spr	Samsung	199.99	1	I already had a phone with problems... I	1	1	
8	"CLEAR CLEAN ESN" Spr	Samsung	199.99	2	The charging port was loose. I got that sc	0	0	
9	"CLEAR CLEAN ESN" Spr	Samsung	199.99	2	Phone looks good but wouldn't stay chai	0	1	
10	"CLEAR CLEAN ESN" Spr	Samsung	199.99	5	I originally was using the Samsung S2 Ga	0	0	
11	"CLEAR CLEAN ESN" Spr	Samsung	199.99	3	It's battery life is great. It's very respons	0	1	
12	"CLEAR CLEAN ESN" Spr	Samsung	199.99	3	My fiance had this phone previously, bu	0	0	
13	"CLEAR CLEAN ESN" Spr	Samsung	199.99	5	This is a great product it came after two	0	1	
14	"CLEAR CLEAN ESN" Spr	Samsung	199.99	5	These guys are the best! I had a little sit	2	1	
15	"CLEAR CLEAN ESN" Spr	Samsung	199.99	1	I'm really disappointed about my phone	1	1	
16	"CLEAR CLEAN ESN" Spr	Samsung	199.99	5	Ordered this phone as a replacement fo	1	0	
17	"CLEAR CLEAN ESN" Spr	Samsung	199.99	2	Had this phone before and loved it but v	0	0	
18	"CLEAR CLEAN ESN" Spr	Samsung	199.99	5	I was able to get the phone I previously	6	1	
19	"CLEAR CLEAN ESN" Spr	Samsung	199.99	5	I brought this phone as a replacement fc	0	0	
20	"CLEAR CLEAN ESN" Spr	Samsung	199.99	4	I love the phone. It does everything I ne	1	1	
21	"CLEAR CLEAN ESN" Spr	Samsung	199.99	3	unfortunately Sprint could not activate t	0	0	
22	"CLEAR CLEAN ESN" Spr	Samsung	199.99	4	The battery was old & had been over use	0	1	
23	"CLEAR CLEAN ESN" Spr	Samsung	199.99	4	pros-beautiful screen,capable of runnin	0	0	

Amazon_Unlocked_Mobile

Fig 5.1.1 Raw Data

5.2 Pre-processing:

Now that we have the corpus of reviews and all the resources that could be useful, we can pre-process the reviews. It is a very important sine all the modifications that we are going to during the process will directly the impat the classifier's performance. The pre-processing includes cleaning, normalization, transformation, feature extraction and selection, etc. the result of pre-processing will be consistent and uniform data that are workable to maximize the classifier's performance. The most important pre-processing is the removal of Unicode, cleaning, stemming, contraction removal, removal of special characters as they had no value that is they are neutral, removal of emotions.

These are the following things in text pre-processing

- **Remove all special characters**
- **Make all words lowercase**
- **Remove punctuation**
- **Tokenize:** divide string into a list of substrings.
- **Remove words not containing letters**
- **Remove words containing numbers**

- **Remove stop words**: stop words are a list of high frequency words like, the, to, and also.
- **Lemmatize**: reduces the dimension of the data by aggregating words that either are the same root or have the same meaning.

For example let us pre-process the below product review

> "*I am ok in this special edition, its not up to the original branded value... also phone calls are unclear... many times I used to avoid headphones for calls... battery performance is good as compared to main edition of 4.5 BT SERIES. For high bass songs sound quality is not good..@ sennheiser 一周有七天* "

Step-1: In the first step unnecessary symbols, Unicode's and special characters are removed as these are add any sentiment to the overall review. So, the review after step-1 will be as below

> "*I am ok in this special edition its not up to the original branded value also phone calls are unclear.. many times I used to avoid headphones for calls... battery performance is good as compared to main edition of 4.5 BT SERIES. For high bass songs sound quality is not good sennheiser*"

Step-2: In the next step we need to make all the words to lower case. The predefined function in python make this task simpler. After the step it would look like this

> "*i am ok in this special edition its not up to the original branded value also phone calls are unclear.. many times i used to avoid headphones for calls... battery performance is good as compared to main edition of 4.5 bt series. for high bass songs sound quality is not good sennheiser*"

Step-3: In this step we have to remove the punctuations such as '"', '.', ',' ..Etc. The review after this step would be

"*i am ok in this special edition its not up to the original branded value also phone calls are unclear many times i used to avoid headphones for calls battery performance is good as compared to main edition of 45 bt series for high bass songs sound quality is not good sennheiser*"

Step-4: Tokenization is a way to split text into tokens. These tokens could be paragraphs, sentences, or individual words. There are five different tokenizer. They are Treebank Word Tokenizer, WordPunct, Punkt Word Tokenizer, Whitespace and pattern Tokenizer. After this step it would look like this

{"i","am","ok","in","this","special","edition","its","not","up","up","to","the","original","branded","value","also","phone","calls","are","unclear","many","time","i","used","to","avoid","headphones","for","calls","battery","performance","is","good","as","compared","to","main","edition","of","bt","series","for","high","bass","songs","sound","quality","is","not","good","sennheiser"}

Step-5: A stop word is a commonly used word (such as "the", "a", "an", "in") that a search engine has been programmed to ignore, both when indexing entries for searching and when retrieving them as the result of a search query. The sample stop words are below

{'ourselves', 'hers', 'between', 'yourself', 'but', 'again', 'there', 'about', 'once', 'during', 'out', 'very', 'having', 'with', 'they', 'own', 'an', 'be', 'some', 'for', 'do', 'its', 'yours', 'such', 'into', 'of', 'most', 'itself', 'other', 'off', 'is', 's', 'am', 'or', 'who', 'as', 'from', 'him', 'each', 'the', 'themselves', 'until', 'below', 'are', 'we', 'these', 'your', 'his', 'through', 'don', 'nor', 'me', 'were', 'her', 'more', 'himself', 'this', 'down', 'should', 'our', 'their', 'while', 'above', 'both', 'up', 'to', 'ours', 'had', 'she', 'all', 'no', 'when', 'at', 'any', 'before', 'them', 'same', 'and', 'been', 'have', 'in', 'will', 'on', 'does', 'yourselves', 'then', 'that', 'because', 'what', 'over', 'why', 'so', 'can', 'did',

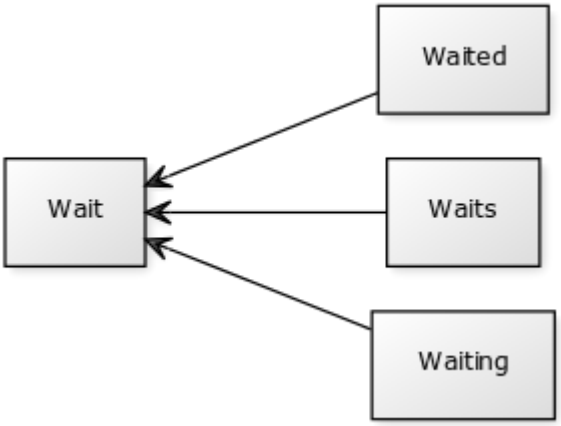
‘not’, ‘now’, ‘under’, ‘he’, ‘you’, ‘herself’, ‘has’, ‘just’, ‘where’, ‘too’, ‘only’, ‘myself’, ‘which’, ‘those’, ‘i’, ‘after’, ‘few’, ‘whom’, ‘t’, ‘being’, ‘if’, ‘theirs’, ‘my’, ‘against’, ‘a’, ‘by’, ‘doing’, ‘it’, ‘how’, ‘further’, ‘was’, ‘here’, ‘than’}

The review after the step-5 is as follows:

{'ok', 'special', 'edition', 'original', 'branded', 'value', 'also', 'phone', 'calls', 'unclear', 'many', 'times', 'used', 'avoid', 'headphones', 'calls', 'battery', 'performance', 'good', 'compared', 'main', 'edition', 'bt', 'series', 'high', 'bass', 'songs', 'sound', 'quality', 'good', 'sennheiser'}

Step-6: A word stem is part of a word. It is sort of a normalization idea, but linguistic. For example, the stem of the word waiting is wait.

Fig 5.2.1 Example of stemming for the word “Wait”



After the step-6 the review be like

['ok', 'special', 'edit', 'origin', 'brand', 'valu', 'also', 'phone', 'call', 'unclear', 'mani', 'time', 'use', 'avoid', 'headphon', 'call', 'batteri', 'perform', 'good', 'compar', 'main', 'edit', 'bt', 'seri', 'high', 'bass', 'song', 'sound', 'qualiti', 'good', 'sennheis']

After all the pre-processing steps above the dataset is ready for training a model. Now let’s dive into the model building, training and analysis. The model is Recurrent Neural Network. The keras and tensor flow are the packages for building the model.

5.3 PRESENT WORK

As the good dataset is what we require for good model we have pre-processed the dataset. Now we have few others steps to do like converting the text review into vectors and allocating each review a vector.

```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences

tk = Tokenizer(lower = True)
tk.fit_on_texts(X)
X_seq = tk.texts_to_sequences(X)
X_pad = pad_sequences(X_seq, maxlen=100, padding='post')
```

Fig 5.3.1 Keras Vectorization

First, the text should be tokenized by fitting Tokenizer class on the data set. As you can see I use "lower = True" argument to convert the text into lowercase to ensure consistency of the data. Afterwards, we should map our list of words (tokens) to a list of unique integers for each unique word using texts_to_sequences class.

```
Original revies:

['          excellent service, great price, and amazing friendly staff both in person and through the customer service hotlin
e (in prague), even though i couldn't speak czech!          '
 '          Comfortable seating with adequate space. Clean and bright.Punctual. Organized storage.I like the electrical and U
SB sockets.There was no wifi however; Maybe there were but I did not know. Better have signs indicating the wifi availability.I
will definitely look for this company in my next travels.       ']

--------------------

Reviews after tokenization and sequencing:

[[197, 30, 103, 188, 3, 671, 172, 146, 314, 8, 323, 3, 178, 1, 57, 30, 837, 8, 326, 62, 261, 4, 954, 222, 1274], [85, 1158, 16,
1275, 518, 105, 3, 2097, 195, 1774, 1159, 4, 108, 1, 2098, 3, 1070, 788, 39, 6, 26, 130, 271, 519, 39, 33, 28, 4, 90, 21, 127,
276, 25, 1392, 1775, 1, 130, 2099, 4, 49, 295, 431, 10, 22, 42, 8, 14, 137, 1276]]
```

Fig 5.3.2: Original reviews vs after tokenization and sequencing

We use pad Sequences class on the list of integers to ensure that all reviews have the same length, which is a very important step for preparing data for RNN model.

Applying this class would either shorten the review to required integers, or pad them with 0's in case they are shorter.

```
Reviews after tokenization and sequencing:

[[197, 30, 103, 188, 3, 671, 172, 146, 314, 8, 323, 3, 178, 1, 57, 30, 837, 8, 326, 62, 261, 4, 954, 222, 1274], [85, 1158, 16,
1275, 518, 105, 3, 2097, 195, 1774, 1159, 4, 108, 1, 2098, 3, 1070, 788, 39, 6, 26, 130, 271, 519, 39, 33, 28, 4, 90, 21, 127,
276, 25, 1392, 1775, 1, 130, 2099, 4, 49, 295, 431, 10, 22, 42, 8, 14, 137, 1276]]
--------------------

Reviews after padding:

[[ 197   30  103  188    3  671  172  146  314    8  323    3  178    1
    57   30  837    8  326   62  261    4  954  222 1274    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0]
 [  85 1158   16 1275  518  105    3 2097  195 1774 1159    4  108    1
  2098    3 1070  788   39    6   26  130  271  519   39   33   28    4
    90   21  127  276   25 1392 1775    1  130 2099    4   49  295  431
    10   22   42    8   14  137 1276    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0]]
```

Fig 5.3.3: Reviews after padding

All the review in the dataset are similarly converted into the vector from which is then used to find the polarity. So the reviews are all set for feeding into the RNN Network. We have converted the text reviews into vector as the model cannot understand text directly. Before building we have to decide the layers that the model must have. In the model we have to map the each review words in the embedding vector space and it must fit all the reviews. So, the first layer in the model is the Embedding Layer followed by LSTM and output layer.

A word embedding is a class of approaches for representing words and documents using a dense vector representation. It is an improvement over more the traditional bag-of-word model encoding schemes where large sparse vectors were used to represent each word or to score each word within a vector to represent an entire

vocabulary. Two popular examples of methods of learning word embeddings from text include:

- **Word2Vec.**
- **GloVe.**

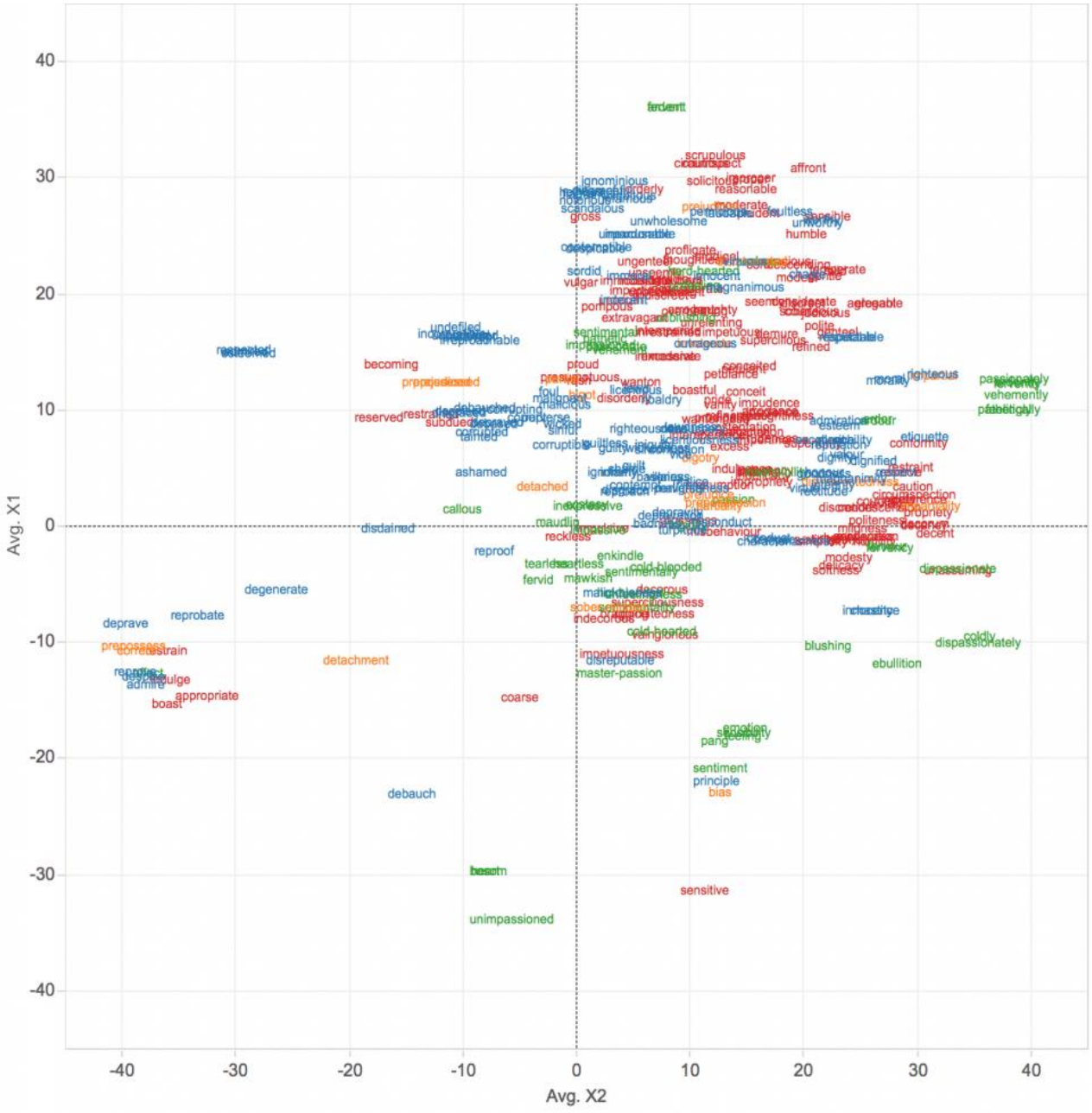


Fig 5.3.4: WordVect mapping of words

Now we have actually split the dataset into training, validation and testing dataset. We can split into training and testing using sklearn train_test_split.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_pad, y, test_size = 0.25, random_state = 1)
```

Fig 5.3.5: train test split

Furthermore the training set can be split into validation and training set. Now let's build the model using the keras package.

```
//import packages

from keras.models import Sequential
from keras.layers import Embedding, LSTM, Dense, Dropout

vocabulary_size = len(tk.word_counts.keys())+1
max_words = 100
embedding_size = 32

//Model
model = Sequential()
model.add(Embedding(vocabulary_size, embedding_size,
input_length=max_words))
model.add(LSTM(200))
model.add(Dense(1, activation='sigmoid'))

//train the model
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
```

Fig 5.3.6: keras model of RNN

As the embedding layer requires the size of the vocabulary and the length of the sequences, we set the size of the vocabulary as the number of words in tokenizer dictionary + 1 and the input length to 100, where the value of the padding must be the same. Embedding size parameters tells us the dimensions that should be used to represent each word. We have LSTM layer with 200 memory cells. Adding more layers and cells can lead to more accuracy and gives good result. Finally we have

the output layer with the sigmoid activation function to predict the probability of review being positive. Now the model is created and set to train. After training the model for 10 epochs, we achieve an accuracy of 98.44% on validation set and 95.93%

```
model.fit( x_train, y_train, validation_data =
(x_valid,y_valid, batch_size = batch_size, epochs = 10)
```

Fig 5.3.7: Training the model

```
Train on 894 samples, validate on 64 samples
Epoch 1/10
894/894 [==============================] - 3s 4ms/step - loss: 0.6309 - acc: 0.6264 - val_loss: 0.5542 - val_acc: 0.7812
Epoch 2/10
894/894 [==============================] - 2s 3ms/step - loss: 0.4602 - acc: 0.8177 - val_loss: 0.4140 - val_acc: 0.8438
Epoch 3/10
894/894 [==============================] - 2s 3ms/step - loss: 0.3744 - acc: 0.8289 - val_loss: 0.3000 - val_acc: 0.9062
Epoch 4/10
894/894 [==============================] - 2s 3ms/step - loss: 0.3332 - acc: 0.8456 - val_loss: 0.2963 - val_acc: 0.8750
Epoch 5/10
894/894 [==============================] - 2s 3ms/step - loss: 0.3030 - acc: 0.8714 - val_loss: 0.3110 - val_acc: 0.9062
Epoch 6/10
894/894 [==============================] - 2s 3ms/step - loss: 0.2405 - acc: 0.8926 - val_loss: 0.2965 - val_acc: 0.9375
Epoch 7/10
894/894 [==============================] - 2s 3ms/step - loss: 0.1871 - acc: 0.9620 - val_loss: 0.1190 - val_acc: 0.9688
Epoch 8/10
894/894 [==============================] - 2s 3ms/step - loss: 0.0925 - acc: 0.9810 - val_loss: 0.0862 - val_acc: 0.9844
Epoch 9/10
894/894 [==============================] - 2s 3ms/step - loss: 0.0381 - acc: 0.9899 - val_loss: 0.0382 - val_acc: 0.9844
Epoch 10/10
894/894 [==============================] - 2s 3ms/step - loss: 0.0095 - acc: 0.9978 - val_loss: 0.0957 - val_acc: 0.9688

<keras.callbacks.History at 0x21c820ea208>
```

Fig 5.3.8: Epoch Output

After the training the model must be evaluated that is the test set that we split is used to evaluate the model.

```
score = model.evaluate(x_test, y_test, verbose = 0)
print("Test accuracy :",score[1]) //Output is 0.9593
```

Fig 5.3.9: Evaluating the model

After evaluating the model the accuracy of the model is **0.9593.** The confusion matrix is drawn for the data.

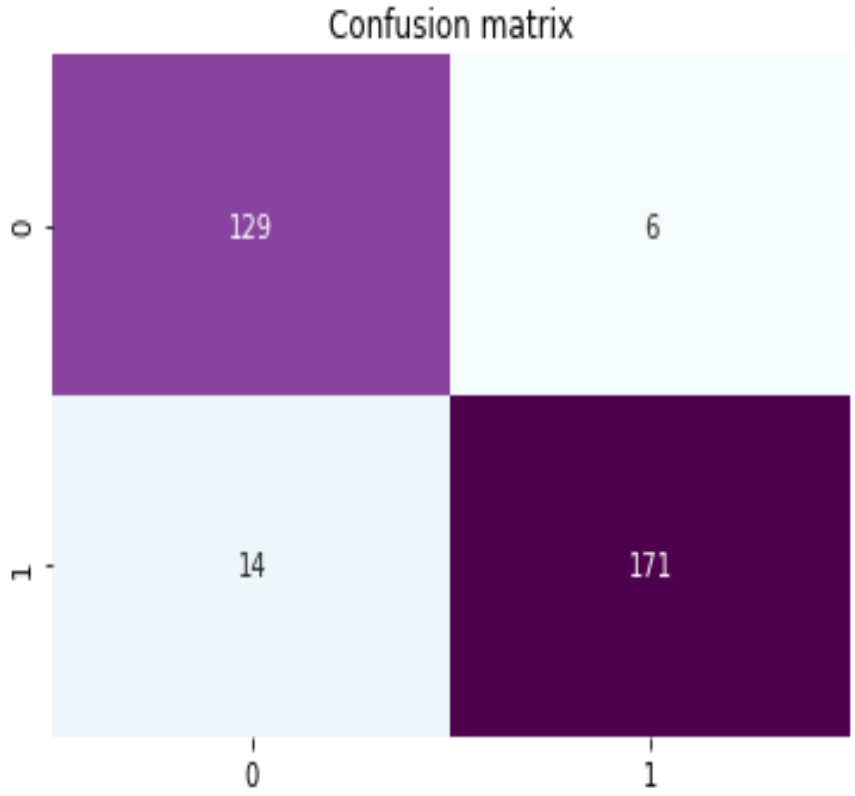


Fig 5.3.10: Confusion matrix

We must do one more validation to confirm the accuracy of the model. We do this by gathering some 100 number of review which are not present in the original dataset. In order to prepare the reviews for prediction, the same pre-processing steps have to be applied on text before passing them into trained model. Now we do the same process we did for the test set instead of test set we just pass the set we have gathered. This would again yield a confusion matrix by which you can come to know the exact accuracy.

The following is the python code for the RNN model and all there models which are Random Forest, Support Vector Machine, and Naïve Bayes Classifier.

5.4 CODE SAMPLES

1. Svm_RndForest_bayes.py

```
import os
import pandas as pd
import numpy as np
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import confusion_matrix
from sklearn.svm import LinearSVC
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier

def import_data():
    train_path = "./data/sent_reviews/train.csv"
    data = pd.read_csv(train_path)
    data = data[data.Is_Response.isnull() == False]
#     data['Is_Response'] = data['Is_Response'].map(int)
    data = data[data['Description'].isnull() == False]
    data.reset_index(inplace=True)
    data.drop('index', axis=1, inplace=True)
    print ('dataset loaded with shape', data.shape)
    data = pd.get_dummies(data, columns=["Is_Response"])
    data.drop(['Browser_Used', 'Device_Used', 'Is_Response_not happy'],
axis=1, inplace=True)
    data.columns = ['User_ID', "Description", "Sentiment"]
    return data

data = import_data()
dfPrint(data.sample(5))

docs = data["Description"]
labels = data["Sentiment"]
docs_train, docs_test, labels_train, labels_test =
train_test_split(docs, labels, test_size=0.2, random_state=69)

//vectorizing data
vectorizer = TfidfVectorizer(min_df=5, max_df = 0.8, sublinear_tf=True,
use_idf=True,stop_words='english')
train_tf_idf = vectorizer.fit_transform(docs_train)
test_tf_idf = vectorizer.transform(docs_test)

model1 = LinearSVC(dual=False)
```

```
model1.fit(train_tf_idf,labels_train)
result1 = model1.predict(test_tf_idf)
score1 = accuracy_score(labels_test, result1, normalize=True,
sample_weight=None)

#print(score1)

model2 = MultinomialNB()
model2.fit(train_tf_idf,labels_train)
result2 = model2.predict(test_tf_idf)
score2 = accuracy_score(labels_test, result2, normalize=True,
sample_weight=None)

#print(score2)

model3 =
RandomForestClassifier(n_estimators=25,max_depth=50,min_samples_leaf=10
, random_state=0)
model3.fit(train_tf_idf,labels_train)
result3 = model3.predict(test_tf_idf)
score3 = accuracy_score(labels_test, result3, normalize=True,
sample_weight=None)

#print(score3)
```

In the above program the model-1 is Support Vector Machines, model-2 is the naïve Bayes classifier and the last model-3 is Random Forest. This programs trains the data, fits to the models and prints the accuracy.

2. Preprocess.py

```
import pandas as pd
import os
import re
from string import punctuation
from textblob import TextBlob
import csv
os.chdir('F:\\Projects\\Main Project\\Code\\main')
data = pd.read_csv('mobile_dataset.csv')
text ="bhargav ihaodihi iugiug 8687 87698_)9*)&**(&56"
def preprocess(text):
    for c in text:
            t =ord(c)
            if not ((t < 123 and t > 96) or (t<91 and t>64) or t==32):
```

```
                text=text.replace(c,'')
    text = re.split(r'\W+',text)
    words = [word.lower() for word in text]
    text = " ".join(word for word in words)
    return text
reviews = data['Reviews']
count = 0
with open('result.csv','w',newline='') as csvfile:
    writer = csv.writer(csvfile,dialect="excel")
    n = int(input(len(reviews)))
    for i in range(n):
        if pd.notnull(reviews[i]):
            temp = preprocess(reviews[i])
            sent = TextBlob(temp)
            if sent.polarity >= 0:
                temp1 = 1
            else:
                temp1 = 0
            count = count +1
            writer.writerow([temp,temp1])
        else:
            print(i)
print(i,count)
```

This preprocess.py converts the raw review into the perfect review and assign the sentiment to the particular review.

3. RNN_Model.py

```
import numpy as np
import pandas as pd
import os
import tensorflow as tf
from sklearn.feature_extraction.text import CountVectorizer
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential,load_model
from keras.layers import Dense,Embedding,LSTM,SpatialDropout1D
from sklearn.model_selection import train_test_split
from keras.utils.np_utils import to_categorical
import os


data = pd.read_csv('result.csv')
data = data[:20000]
print(data[data['sentiment']==1].size)
```

```
print(data[ data['sentiment']==0].size)
data = data[pd.notnull(data['text'])]

max_fatures = 2000
tokenizer = Tokenizer(num_words=max_fatures, split=' ')
tokenizer.fit_on_texts(data['text'].values)
X = tokenizer.texts_to_sequences(data['text'].values)
X = pad_sequences(X)

with tf.device('/device:GPU:0'):
    embed_dim = 128
    lstm_out = 196
    model = Sequential()
    model.add(Embedding(max_fatures, embed_dim,input_length =
X.shape[1]))
    model.add(SpatialDropout1D(0.4))
    model.add(LSTM(lstm_out, dropout=0.2, recurrent_dropout=0.2))
    model.add(Dense(2,activation='softmax'))
    model.compile(loss = 'categorical_crossentropy',
optimizer='adam',metrics = ['accuracy'])
    print(model.summary())

    Y = pd.get_dummies(data['sentiment']).values
    X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size
= 0.33, random_state = 42)
    print(X_train.shape,Y_train.shape)
    print(X_test.shape,Y_test.shape)

    batch_size = 32
    model.fit(X_train, Y_train, epochs = 7, batch_size=batch_size,
verbose = 2)
    validation_size = 1500
    X_validate = X_test[-validation_size:]
    Y_validate = Y_test[-validation_size:]
    X_test = X_test[:-validation_size]
    Y_test = Y_test[:-validation_size]
    score,acc = model.evaluate(X_test, Y_test, verbose = 2, batch_size
= batch_size)
    print("score: %.2f" % (score))
    print("acc: %.2f" % (acc))
model.save('model2.h5')
pos_cnt, neg_cnt, pos_correct, neg_correct = 0, 0, 0, 0
for x in range(len(X_validate)):
    result =
model.predict(X_validate[x].reshape(1,X_test.shape[1]),batch_size=1,ver
bose = 2)[0]
    if np.argmax(result) == np.argmax(Y_validate[x]):
```

```
        if np.argmax(Y_validate[x]) == 0:
            neg_correct += 1
        else:
            pos_correct += 1
    if np.argmax(Y_validate[x]) == 0:
        neg_cnt += 1
    else:
        pos_cnt += 1
print("pos_acc", pos_correct/pos_cnt*100, "%")
print("neg_acc", neg_correct/neg_cnt*100, "%")
```

This **rnn_model.py** trains and build a model called "**Model.h5**" which can be used for testing purpose later on.

4. **test.py**

```
import numpy as np
from keras.models import load_model
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
model = load_model('model2.h5')
max_fatures = 2000
tokenizer = Tokenizer(num_words=max_fatures, split=' ')

def test(twt):
    #vectorizing the tweet by the pre-fitted tokenizer instance
    tokenizer.fit_on_texts(twt)
    twt = tokenizer.texts_to_sequences(twt)
    #padding the tweet to have exactly the same shape as `embedding_2`
input
    twt = pad_sequences(twt, maxlen=1504, dtype='int32', value=0)
    #print(twt)
    sentiment = model.predict(twt,batch_size=1,verbose = 2)[0]

    print(sentiment)
    if(np.argmax(sentiment) == 0):
        print("negative")
    elif (np.argmax(sentiment) == 1):
        print("positive")
review = "i feel so lucky to have found this used phone to us not used
hard at all phone on line from someone who upgraded and sold this one
my son liked his old one that finally fell apart after years "
test(review) #output positive
```

This is the **test.py** we were taking about in the previous paragraph. After the training of the model it saves the model with name "**model.h5**" that is used in the present application to test and generate the result for the review passed through the UI.

CHAPTER 6

TESTING

TESTING

6.1 Overview of Testing:

Software Testing is evaluation of the software against requirements gathered from users and system specifications. Testing is conducted at the phase level in software development life cycle or at module level in program code. Software testing comprises of Validation and Verification.

Software testing is an investigation conducted to provide stakeholders with information about the product or service under test. Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include the process of executing a program or application with the intent of finding (errors or other defects), and verifying that the software product is fit for use.

Software testing involves the execution of a software component or system component to evaluate one or more properties of interest. In general, these properties indicate the extent to which the component or system under test

- meets the requirements that guided its design and development,
- responds correctly to all kinds of inputs,
- performs its functions within an acceptable time,
- is sufficiently usable,
- can be installed and run in its intended environments, and
- Achieves the general result its stakeholder's desire.

As the number of possible tests for even simple software components is practically infinite, all software testing uses some strategy to select tests that are feasible for the available time and resources. As a result, software testing typically (but not exclusively) attempts to execute a program or application with the intent of finding software bugs (errors or other defects). The job of testing is an iterative

process as when one bug is fixed, it can illuminate other, deeper bugs, or can even create new ones.

Software testing can provide objective, independent information about the quality of software and risk of its failure to users or sponsors.[1]

Software testing can be conducted as soon as executable software (even if partially complete) exists. The overall approach to software development often determines when and how testing is conducted. For example, in a phased process, most testing occurs after system requirements have been defined and then implemented in testable programs. In contrast, under an agile approach, requirements, programming, and testing are often done concurrently.

Tests can be conducted based on two approaches –

- Functionality testing
- Implementation testing

Software testing methods are traditionally divided into white- and black-box testing. These two approaches are used to describe the point of view that the tester takes when designing test cases.

Typical Failures in Applications:

- Application Installation Failure
- Application crash during execution
- Scaling / Layout problems
- Application hangs if some resources is not available
- Problem in landscape / Portrait mode

Test Methods:

- **Black Box Testing:**

Black-box testing is a method of software testing that examines the functionality of an application (what the software does) without going inside the internal structure (White-box Testing). We also have something called Gray-box testing which something is in between. You need no knowledge of how the system is created.

• Black-box testing can be done by a person who only know what the software is supposed to do

• Compare to driving a Car – you don't need to know how it is built in order to test it.

- **White Box Testing:**

  You need to have knowledge of how (Design and Implementation) the system is built. In white-box testing, an internal perspective of the system, as well as programming skills, are used to design test cases. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs. While white-box testing can be applied at the unit, integration and system levels of the software testing process, it is usually done at the unit level. It can test paths within a unit, paths between units during integration, and between subsystems during a system–level test. Though this method of test design can uncover many errors or problems, it might not detect unimplemented parts of the specification or missing requirements.

Testing Levels:

Testing itself may be defined at various levels of SDLC. The testing process runs parallel to software development. Before jumping on the next stage, a stage is tested, validated and verified. Software is tested on various levels:

1. Unit testing:

While coding, the programmer performs some tests on that unit of program to know if it is error free. Testing is performed under white-box testing approach.

Unit testing helps developers decide that individual units of the program are working as per requirement and are error free.

2. Integration Testing:

Even if the units of software are working fine individually, there is a need to find out if the units if integrated together would also work without errors. For example, argument passing and data updating etc.

3. System Testing:

System Testing follows Integration Testing.

• It consists of Black-box Tests that validate the entire system against its requirements.

• Checking that a software system meets specifications and that it fulfils its intended purpose.

• Often executed by an independent group (QA group, QA – Quality Assurance).

• Since system tests make sure the requirements are fulfilled, they must systematically validate each requirement in the SRS (Software Requirements Specifications).

4. Acceptance Testing: When the software is ready to hand over to the customer it has to go through last phase of testing where it is tested for user-interaction and response. This is important because even if the software matches all user requirements and if user does not like the way it appears or works, it may be rejected.

5. Regression Testing:

Whenever a software product is updated with new code, feature or functionality, it is tested thoroughly to detect if there is any negative impact of the added code. This is known as regression testing.

CHAPTER 7

RESULTS AND DISCUSSIONS

RESULTS AND DISCUSSIONS

7.1 Results:

In the following section I would like to compare the accuracies of different model's and the promising model that gives much accuracy will be linked to the user interface.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 1504, 128)         256000
_________________________________________________________________
spatial_dropout1d_1 (Spatial (None, 1504, 128)         0
_________________________________________________________________
lstm_1 (LSTM)                (None, 196)               254800
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 394
=================================================================
Total params: 511,194
Trainable params: 511,194
Non-trainable params: 0
```

Fig 7.1.1 Layers of RNN

```
Trainning set shape   : (13375, 1504) (13375, 2)
Testing set shape          : (6589, 1504) (6589, 2)

Epoch 1/7
 - 2421s - loss: 0.3170 - acc: 0.8704
Epoch 2/7
 - 1770s - loss: 0.2092 - acc: 0.9166
Epoch 3/7
 - 14908s - loss: 0.1651 - acc: 0.9387
Epoch 4/7
 - 3534s - loss: 0.1267 - acc: 0.9538
Epoch 5/7
 - 72566s - loss: 0.1014 - acc: 0.9626
Epoch 6/7
 - 2099s - loss: 0.0820 - acc: 0.9701
Epoch 7/7
 - 1893s - loss: 0.0680 - acc: 0.9764

score: 0.17
acc: 0.97
```

Fig 7.1.2 Training RNN Model

The above image presents the training accuracy of the RNN model that is **95%.**

Now we have test the model giving a text as input and check its accuracy.

```
text : "i feel so lucky to have found this used phone to us not used
hard at all phone on line from someone who upgraded and sold this one
my son liked his old one that finally fell apart after years and didnt
want an upgrade thank you seller we really appreciate it your honesty
re said used phonei recommend this seller very highly would but from
them again"
pos_acc 98.34775767112511 %
neg_acc 78.16593886462883 %
[0.98630464 0.01369538]
positive
```

Fig 7.1.3 testing the model

For the **Support Vector Machine Classification model**, the values gamma, C are tuned and comparison of various kernels is done. After tuning, the Poly Kernel seems to be the best fit and the degree used is 3. The above parameters on the SVM Classification model gave an accuracy of **85.54%** on the Training dataset, **85.53%** on the testing dataset.

For the **Naïve Bayes classification model**, the priors are not needed to train the learning model. The Laplacian smoothing of 1.0 is applied to the Training dataset to estimate the value. The Multinomial Naïve Bayes gave a better training accuracy of 87.6% and a testing accuracy of **82.79%.**

For the **Random forest classifier**, the hyper parameters values for number of trees 400, number of features at random 11, depth value was set to unlimited and that provided classification. Though different values of hyper parameters are tried manually for each iteration and based on the accuracy returned in that iteration hyper parameters values will be updated for the next iteration. However we have mainly focused on two hyper parameter that is number of trees and number of features. In which increase in number of trees linearly increases accuracy up to certain values and after that there will not be any drastic change in accuracy results The

Random Forest classifier gave training accuracy of **92.34%** and testing accuracy of **91.2%.**

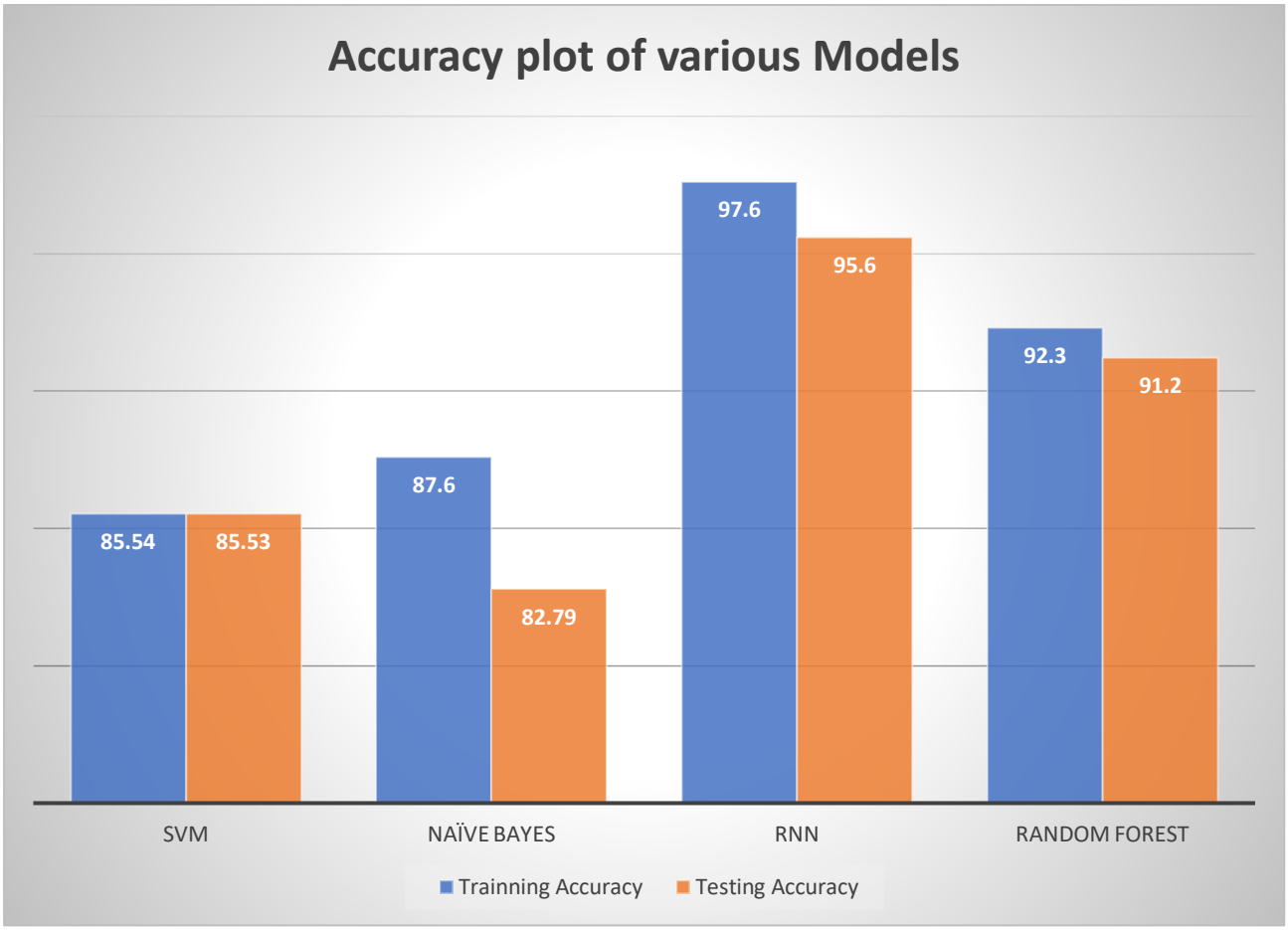


Fig 7.1.4 plot for accuracy comparison of each classification models

As by the above plot we came to know that Recurrent Neural Network provide good accuracy over all the machine learning algorithms such as Random Forests, Support Vector Machines and Naïve Bayes. So, in the main application I would like to implement only the RNN. In this project we use Recurrent Neural Network to analyse the review and give the sentiment of the review.

7.2 Output Screens:

This is the main activity that takes the input's which are the product name and review. After providing inputs Clcik "**Analyse**" button. This application uses the RNN model as this gives more accuracy over others.

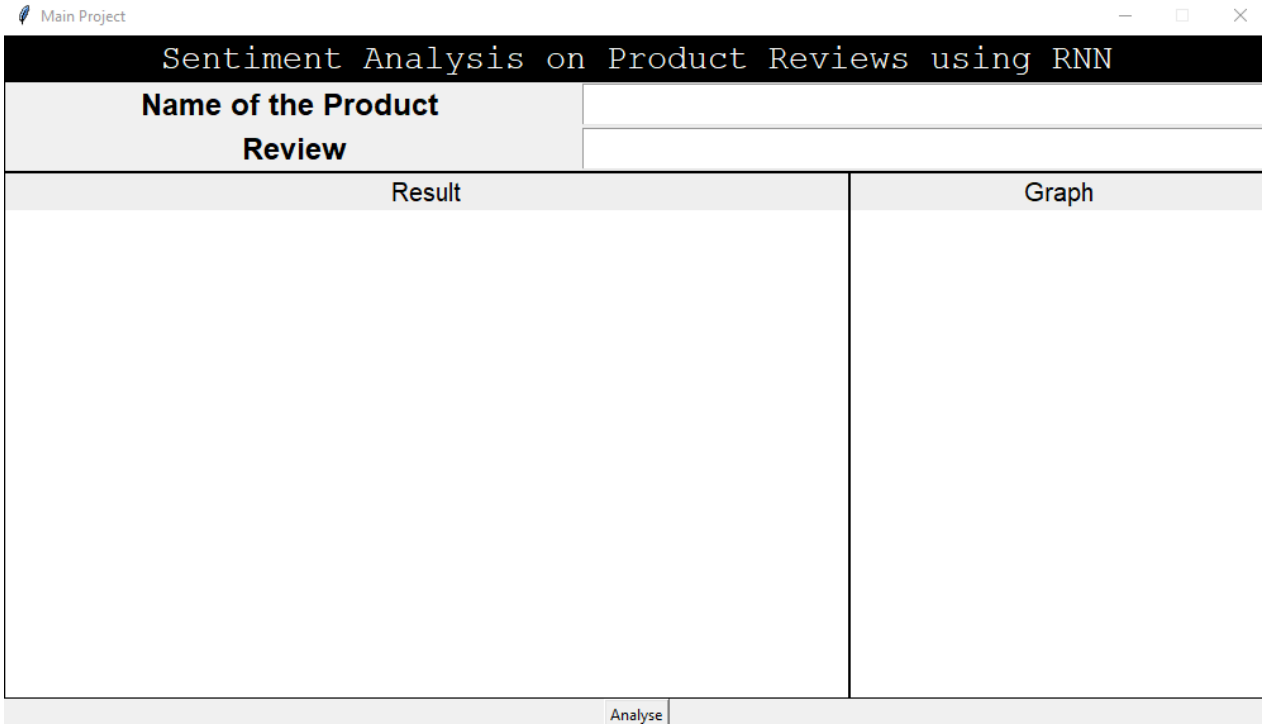


Fig 7.2.1: Main Layout of the Application

After clicking "**analyse**" the review and the product name is sent to the RNN model to analyse the review. The result can be viewed in the result window shown in the UI Layout. This Result window contains all the important data such as the review, Tokenized text, vectorised text, Polarity and the overall sentiment.

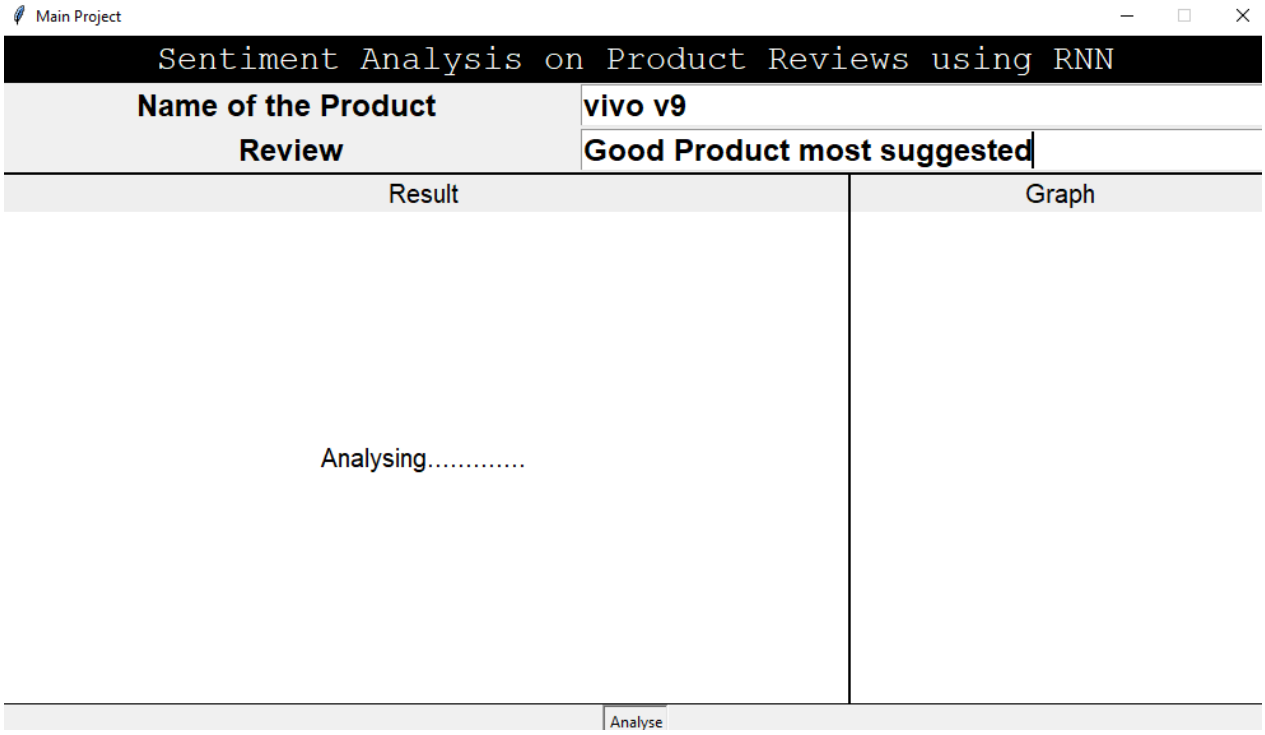


Fig 7.2.2: Analyse Window

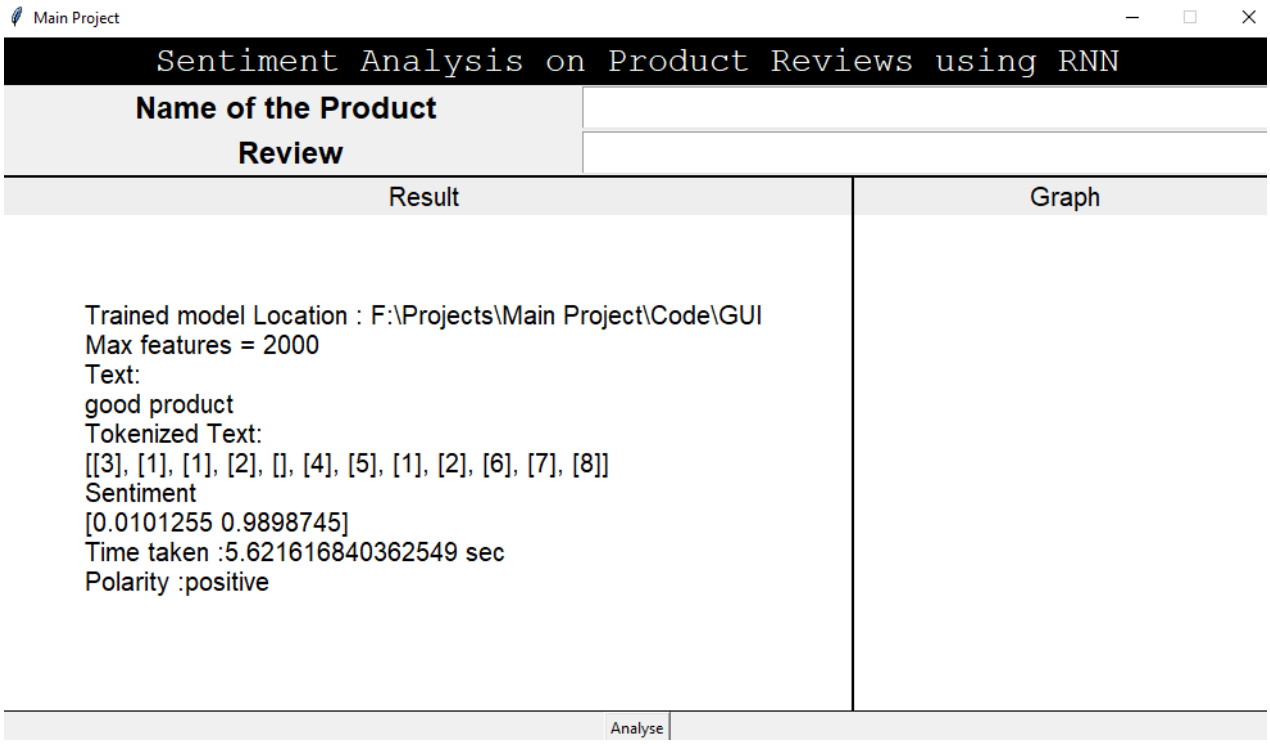


Fig 7.2.3 Result of the review

CHAPTER 7

CONCLUSION

CONCLUSION

Our objective in this project was to apply the advances in deep learning, including more intuitive model architectures to the sentiment classification problem. We performed several experiments with approaches that have traditionally been used for sentiment analysis, like random forest, SVM and Naive Bayes. We also extensively experimented with the proposed architecture.

The first method that we approached for our problem is Naïve Bayes. It is mainly based on the independence assumption .Training is very easy and fast In this approach each attribute in each class is considered separately. Testing is straightforward, calculating the conditional probabilities from the data available. One of the major task is to find the sentiment polarities which is very important in this approach to obtain desired output. In this naïve Bayes approach we only considered the words that are available in our dataset and calculated their conditional probabilities. We have obtained successful results after applying this approach to our problem.

The second method that we applied for our problem is Random Forest. It is an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes.

The third method that we applied for our problem is support vector machine. It reduces the large dimensions into smaller without any loss of data. A window (comprised of five words on either side of the given word) is used to count the number of appearances of each word in our data set, to find the combinations of words .Training is very long compared to naïve Bayes, bag of words .The training of support vector machine is done only with a small dataset. The outcomes of this approach are obtained partially.

However, we were successful at predicting sentiment on topics in mobile reviews on a small scale using three different approaches Naïve Bayes, Random Forest, Support Vector Machine and also gained a lot of information in machine learning.

Finally we have applied Recurrent Neural Network on the same mobile dataset and have had good result. This project compares the accuracies of all the four methods we built and prove that the Recurrent Neural Network model gives much more accuracy than the traditional machine learning algorithm with less loss.

Our present work deals with the sentence level based sentiment analysis on the reviews. Our further work will be on the aspect level based sentiment analysis on the same product review which would be more helpful the companies to improve their product quality. The aspect based will help the companies to understand their product pros and cons very specifically which would really help in the development of the company.

CHAPTER 9
REFERENCES

REFERENCES

1. Yuling Chen and Zhi Zhang. “Research on text sentiment analysis based on CNNs and SVM”, 2018 13th IEEE Conference on Industrial Electronics and Applications (ICIEA).
2. Neha Raghuvanshi and Prof. J.M. Patil, “A Brief Review on Sentiment Analysis”, International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT) - 2016
3. Qiongxia Huang, Xianghan Zheng, Riqing Chen, and Zhenxin Dong, “Deep Sentiment Representation Based on CNN and LSTM”, in 2017 International Conference on Green Informatics.
4. Vikas Malik. And Amit Kumar, “Analysis of Twitter Data Using Deep Learning Approach: LSTM,” in International Journal on Recent and Innovation Trends in Computing and Communication .
5. Abdullah Aziz Sharfuddin, Md Nafis Tihami and Md Saiful Islam, ” A Deep Recurrent Neural Network with BiLSTM model for Sentiment Classification” in International Conference on Bangla Speech and Language Processing.
6. Huma Parveen, and shikha pandey, “Sentiment analysis on Twitter Data-set using Naive Bayes algorithm” in IEEE 2016
7. K. M. Azharul Hasan, Mir Shankar Sabuj, and Zakia Afrin, “Opinion mining using Naïve Bayes” in 2015 IEEE conference.
8. Nurulhuda Zainuddin and Ali Selamat, “Sentiment analysis using Support Vector Machine” in IEEE 2014
9. Xi Ouyang, Pan Zhou, Cheng Hua Li, and Lijun Liu, “Sentiment Analysis Using Convolutional Neural Network” in 2015 IEEE.
10. https://medium.com/@martinpella/naive-bayes-for-sentiment-analysis-49b37db18bf8