A project report on

NLP BASED AMAZON PRODUCT REVIEW ANALYSIS USING DEEP LEARNING

Submitted in partial fulfillment for the award of the degree of

B.Tech Computer Science and Engineering

by

ROHIT SUBRAMANIAN ARIVALAGAN (17BCE1291)

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science and Engineering

May, 2021

NLP BASED AMAZON PRODUCT REVIEW ANALYSIS USING DEEP LEARNING

Submitted in partial fulfillment for the award of the degree of

B.Tech Computer Science and Engineering

by

ROHIT SUBRAMANIAN ARIVALAGAN (17BCE1291)

School of Computer Science and Engineering

May, 2021

DECLARATION

I here by declare that the thesis entitled " NLP based Amazon Product Review Analysis using Deep Learning " submitted by me, for the award of the degree of B.Tech Computer Science and Engineering, VIT is a record of bonafide work carried out by me under the supervision of Dr. K.Sathyarajasekaran.

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Chennai

Date: 21/05/2021

Rohit Subramanian Arivalagan

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science and Engineering

CERTIFICATE

This is to certify that the report entitled " NLP based Amazon Product Review Analysis using Deep Learning " is prepared and submitted by **Rohit Subramanian Arivalagan** (**17BCE12921**) to VIT Chennai, in partial fulfullment of the requirement for the award of the degree of **B.Tech. CSE** programme is a bonafide record carried out under my guidance. The project fulfills the requirements as per the regulations of this University and in my opinion meets the necessary standards for submission. The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma and the same is certified.

Signature of the Guide:
Name: Dr. K.Sathyarajasekaran
Date:

Signature of the Internal Examiner
Name:
Date:

Signature of the External Examiner
Name:
Date:

Approved by the Head of Department, **B. Tech CSE**
Name: Dr. Justus S
Date:

(Seal of SCOPE)

ABSTRACT

Over the past few years, it is an irrefutable fact that e-commerce has grown exponentially, especially during the pandemic situation when people prefer not to step outside the comfort and safety of their homes for their purchasing needs. Since the consumer cannot physically analyse the product, they have to resort to the photos and ratings that accompany the reviews of the products to decide if it is worth purchasing or not which makes the reviews an important aspect of e-commerce.

In this paper, various Natural Language Processing along with Deep Learning techniques are used to identify suspicious reviews, summarise the reviews and predict the rating of the product based on a sentiment classification of the reviews. A simple SVM model is proposed for the suspicious review classifier. For the review summariser, an Encoder-Decoder architecture with a 3-stacked LSTM for the encoder model is used and for the sentiment classifier, a Bidirectional RNN with LSTM – GRU is implemented. Although many papers have explored each of the separate modules and proposed various techniques, this paper proposes a system that performs all the 3 operations in a single program.

The concept of multithreading is further used to run the three modules in parallel in an attempt to reduce the run time and the whole project is packaged into an application using Python Tkinter GUI.

ACKNOWLEDGEMENT

It is my pleasure to express with deep sense of gratitude to Dr. K.Sathyarajasekaran, Assistant Professor (Sr.), School of Computer Science and Engineering, Vellore Institute of Technology Chennai, for his constant guidance, continual encouragement, understanding; more than all, he taught me patience in my endeavor. My association with him is not confined to academics only, but it is a great opportunity on my part of work with an intellectual and expert in the field of Natural Language Processing.

I would like to express my gratitude to Dr. G. Vishwanathan, Mr. Sankar Vishwanathan, Dr. Rambabu Kodali, Dr. S.Narayanan for providing with an environment to work in and for his inspiration during the tenure of the course.

In jubilant mood I express ingeniously my whole-hearted thanks to Dr. Justus S, Head of the Department & Co-ordinator, Dr. Karmel A, and Project Coordinator Dr. B V A N S S Prabhakar Rao, B. Tech. Computer Science and Engineering, SCOPE, VIT Chennai Campus, for their valuable support and encouragement to take up and complete the thesis.

Special mention to Dean, Dr. Jagadeesh Kannan R, Associate Dean, Dr. Geetha S, SCOPE, VIT Chennai, for spending their valuable time and efforts in sharing their knowledge and for helping us in every aspect.

All teaching staff and members working as limbs of our university for their not-self-centered enthusiasm coupled with timely encouragements showered on me with zeal, which prompted the acquirement of the requisite knowledge to finalize my course study successfully. I would like to thank my parents for their support.

It is indeed a pleasure to thank my friends who persuaded and encouraged me to take up and complete this task. At last but not least, I express my gratitude and appreciation to all those who have helped me directly or indirectly toward the successful completion of this project.

Place: Chennai

Date: 21/05/2021

Rohit Subramanian

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

Acronym	Full-form
NLP	Natural Language Processing
CNN	Convolutional Neural Networks
RNN	Recurrent Neural Networks
ANN	Artificial Neural Network
ML	Machine Learning
DL	Deep Learning
AI	Artificial Intelligence
SVM	Support Vector Machine
LSTM	Long Short Term Memory
GRU	Gated Recurrent Units
seq2seq	Sequence to Sequence
GloVe	Global Vectors

Chapter 1

Introduction

It is almost impossible to find any service online that doesn't have a review in recent times. In an age of digitalization, when every business is taking up the online route, reviews are a very important part of their core business. It influences both customer decisions and business decisions at the same time. The customers can decide whether to buy a product or not after an analysis of its reviews and a business can analyse the reviews and take decisions as to how to improve or keep up the standards of their products and set up a marketing policy. The drawbacks however for these online reviews do exist. It is not wise to trust a single review whole-heartadly as it could be fake and the legitimacy of a review is always up for questioning. This is where this project comes into use as it uses various Natural Language Processing and Deep Learning techniques to analyze and predict information about the reviews.

Fig 1.1 Amazon Reviews

Natural Language Processing (NLP) is a division of Artificial Intelligence and Computational Linguistics which is concerned with the computers being able to understand and process human language and context. It usually involves processing and analyzing unstructed text data that is messy and hard to interpret. An ideal NLP system should be able to read, decrypt, comprehend, and make sense of the human lingos and process this to convey valuable information that can be used to take various decisions.

There are 5 stages involved in Natural Language Processing :

- Lexical Analysis
- Syntax Analysis
- Semantic Analysis
- Discourse Integration
- Pragmatic Analysis

Syntax and Semantic analysis are most commonly found where NLP uses various algorithms to comprehend the gist and understanding of words and in what way sentences are organized and excerpt meaning related with a sentence and produce valuable information from them when it is provided with a text.

With the exponential increase in the computational power and data accessible in recent times, NLP has been booming in various industries to achieve meaningfull results. There are a lot of use cases of NLP that make it essential even in industries such as healthcare, finance, media, marketing, management etc.

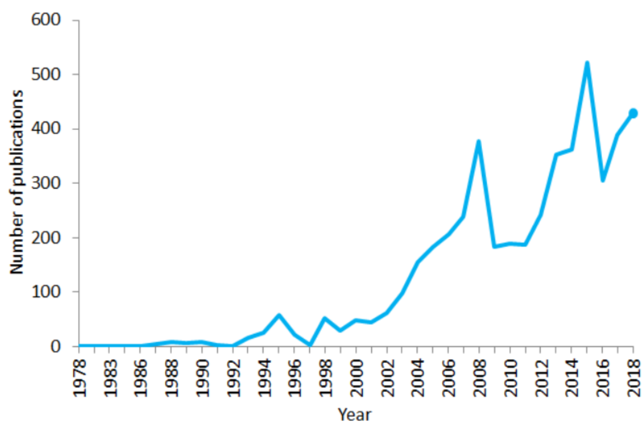


Fig 1.2 Trend of publications involving NLP

This graph shows the rapid increase in the amount of research work involving NLP in the last few decades.

NLP is widely considered as one of the toughest problems to solve for a computer due to the the uncertainty and vague characteristics of human languages that make NLP problematic for machines to implement. If not designed properly, the model may be unsuccessful in understanding the gist of a sentence well and return inaccurate outputs. In this project, NLP is implemented using deep learning techniques.

Deep Learning is a branch of machine learning which works with artificial neural networks that learn and improve on their own by examining their performance. Artificial Neural Networks are layers of nodes that are connected to each other and designed like the human brain. The more the number of layers, the deeper the neural network is said to be. Neural networks are usually very helpful in tasks such as clustering, regression or classification and any problem that involves unstructured data.

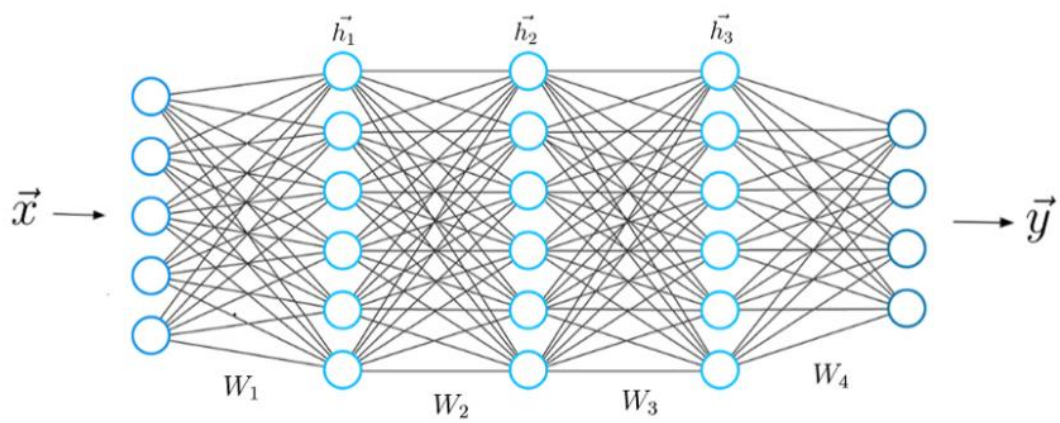


Fig 1.3 Neural Network Architecture

One of the main advantages of deep learning algorithms are the fact that they do not need feature extraction which is often complex and requires acute knowledge of the problem domain. In deep learning, the model recognizes the unique characteristics on its own and eliminates the need of feature extraction. During the learning process of the neural networks, the weights between neurons are constantly changing and adapting for better results. Over its period of training, the program alters the weights and learns and eventually the probability of the correct prediction increases.

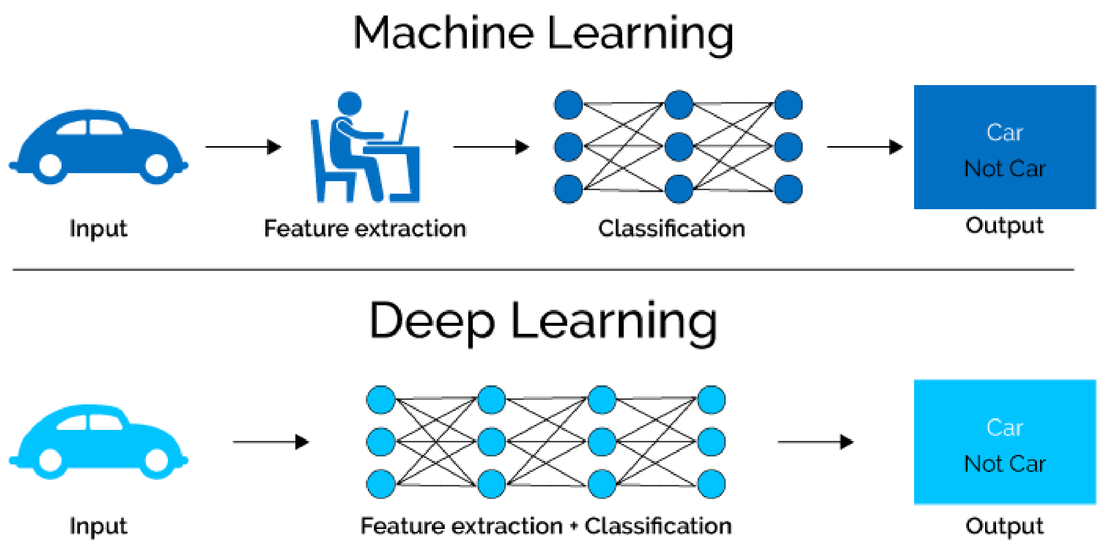


Fig 1.4 Difference between DL and ML algorithms

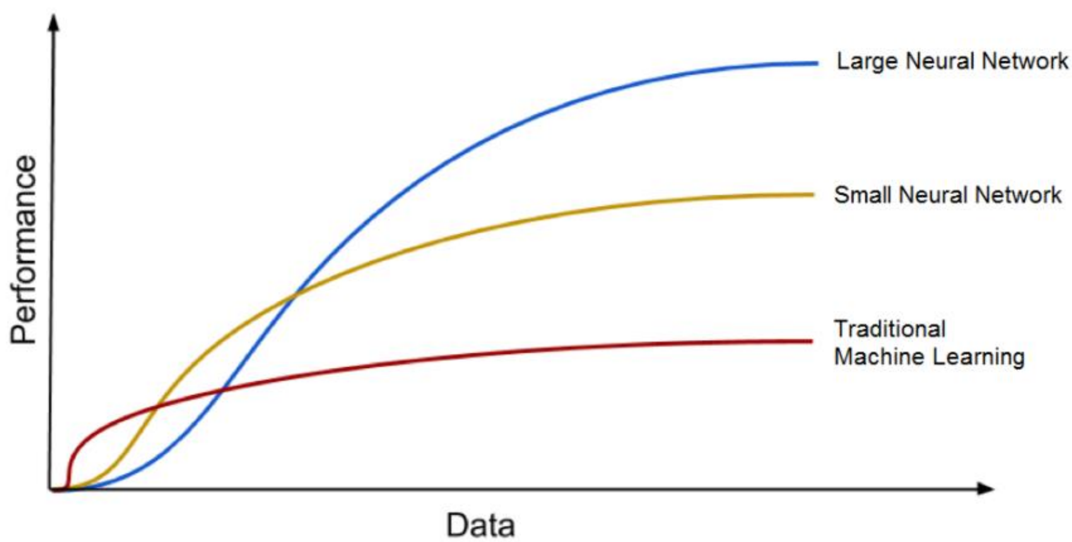


Fig 1.5 DL algorithms performance with amount of data

Deep Learning algorithms and systems perform better with larger amounts of data for accurate results as seen from the graph above and require heavy computational power due to the amount of data and several complex mathematical calculations involved. The exponential increase in computational power and emergence of big data in the recent years has converted deep learning from a theory to a reality.

Some real world applications of Deep Learning are:

- Real-time behaviour analysis
- Autonomous car functions in unstructured conditions
- Pixel restoration
- Automated handwriting recognition
- Demographic prediction
- News aggregation
- Automated machine translation
- Server optimization , Data centre security
- Natural Language Processing / Pattern recognition
- Symptom based disease identification / prediction etc.

Neural networks used in deep learning can be broadly classified into 3 types:

1. Artificial Neural Networks
2. Convolutional Neural Networks
3. Recurrent Neural Networks

An Artificial Neural Network (ANN) follows the multilayer percepetron architecture and feed forward mechanism. It is mainly built of 3 layers- input, output and hidden layer (which acts as a processing layer). The concept of backpropogation is used constantly update the weights of the network. ANN is usually used in problems pertaining to tabular or text data.

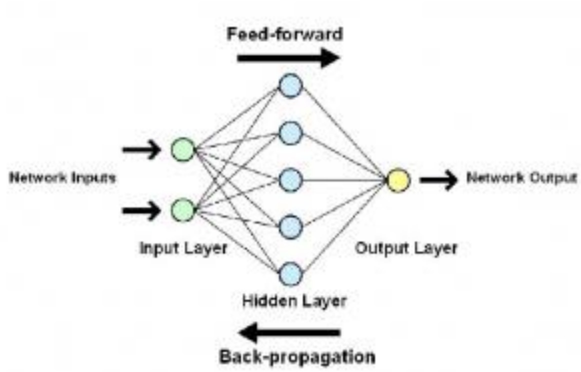


Fig 1.6 ANN architecture

A Convolutional Neural Network (CNN) is also based on the multilayer perceptron architecture but comprises of filters called kernels that use various convolutional operations for feature extraction. CNNs are most widely used for image data as they have the abilty to capture spatial features without explicit coding.

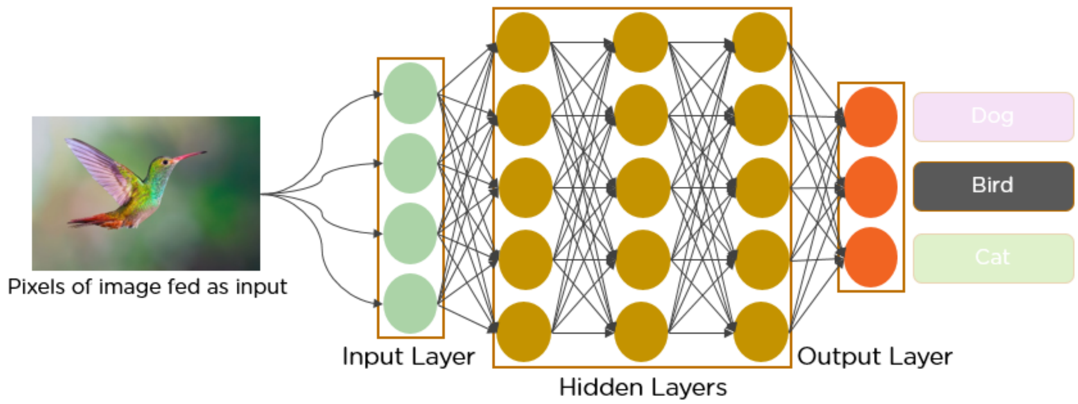


Fig 1.7 CNN Architecture

The CNN consists of the same three layers- input, output and hidden layers as the ANN, however the hidden layers usually comprise of a convolutional and pooling layers that perform the computations on the image input received.

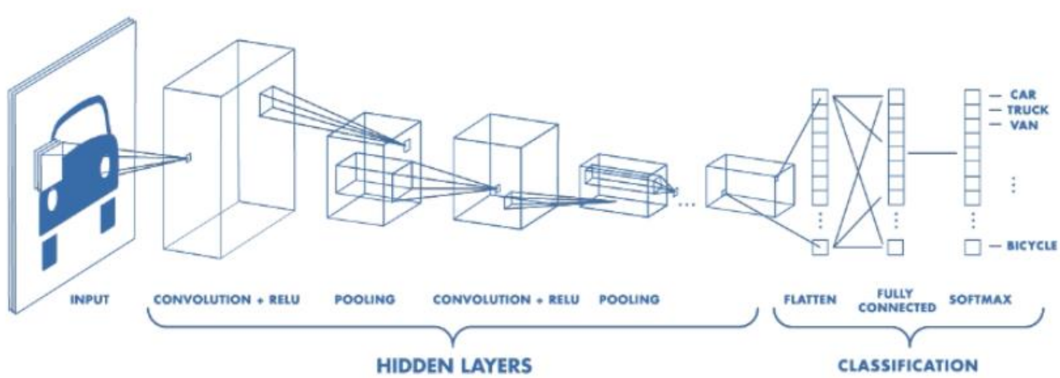


Fig 1.8 CNN Hidden Layers

Recurrent Neural Networks (RNN) have the ability to remember and are hence used for sequential data. Since RNN is widely used in this project, it is discussed in the upcoming sections.

1.1 SUSPICIOUS REVIEW CLASSIFIER

Since the dataset used is a labelled dataset, for the suspicious review filter simple SVM model that classifies the reviews as suspicious or non-suspicious is proposed. The dataset provided is used for classifying fake or real reviews. However, since it is hard for even human beings to be able to identify a review as fake or real, the objective has been pivoted to classify them as suspicious or non-suspicious which seems more appropriate.

A **SVM (Support Vector Machine)** is one of the supervised machine learning algorithms that is generally used for classification problems and is called SVC in such cases. In SVC, the data points are plotted in an n-dimensional space and then a hyperplane (decision boundary) is found which can distinctly classify the data points into different classes. The data points are called support vectors and the line that is used for classification is called as the hyperplane or decision boundary. The main objective is to find the best possible hyperplane for our problem which would be the one which is least biased or has maximum distance from the nearest element of each class.

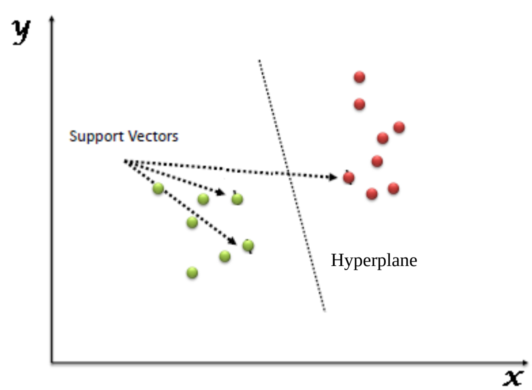


Fig 1.9 Support Vector Machine

The Python NLTK and sklearn libraries have been used to analyse and pre-process the dataset and Jupyter Notebook and other techniques are used to build an SVM classifier that can classify a review as suspicious or not.

1.2 REVIEW SUMMARIZER

Sometimes the customers do not want to read the entire review before deciding to purchase a product. They just want to read a gist of the summary and have a look at the average rating of the product. The deep learning text summariser proposed can understand the context of the review and provide us with a concise summary. Once trained, the model will be able to input a string (review) and summarise it within a phrase which saves the customer a lot of time.

Text summarisation could be of two types:

- Extractive Summarization – The summary is made up of the important sentences extracted from the original text.

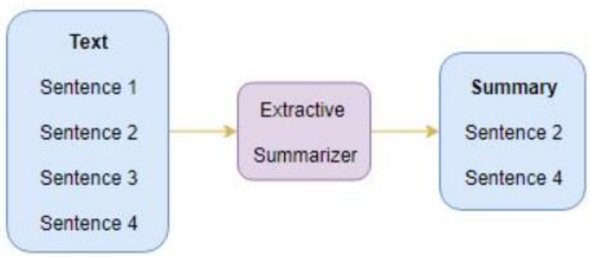


Fig 1.10 Extractive Summarization

- Abstractive Summarization – The summary is made up of new sentences that have been generated from the original text.

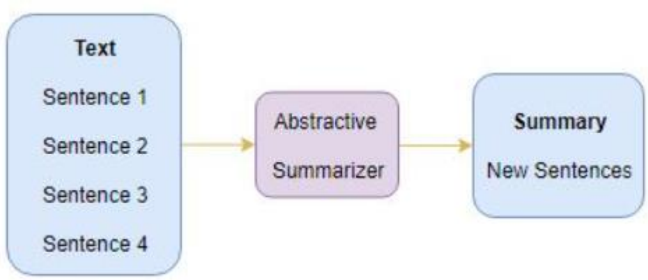


Fig 1.11 Abstractive Summarization

In this paper, an abstractive text summarizer powered by deep learning techniques is proposed. In order to comprehend the context of the complete review and summarise, Seq2Seq (Sequence-to-Sequence) model is used. Seq2Seq is a machine learning approach used to convert a given input sequence from one domain to another. A typical Seq2Seq model has Encoder and Decoder components which are used when the input and output sequences are of different lengths. The Encoder and Decoder components serve as two different neural network models combined into one giant network.

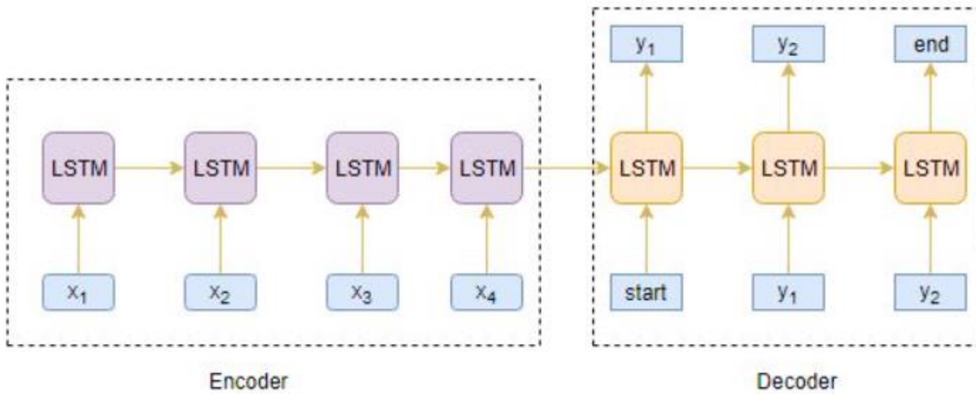


Fig 1.12 Encoder Decoder Architecture

An Encoder inputs a sequence and understands it to generate a reduced dimensional representation of it (vector) which gets passed onto a Decoder that interprets and reverses the process to generate a sequence as output from the vector. Seq2Seq usually uses a **RNN (Recurrent Neural Network)** such as LSTM. A RNN is a neural network architecture that is often used for problems that require dealing with context as the previous step's outputs are fed as the input to the current step. Each RNN has a "memory" component which stores information from previous states and uses them as parameters to produce an output.

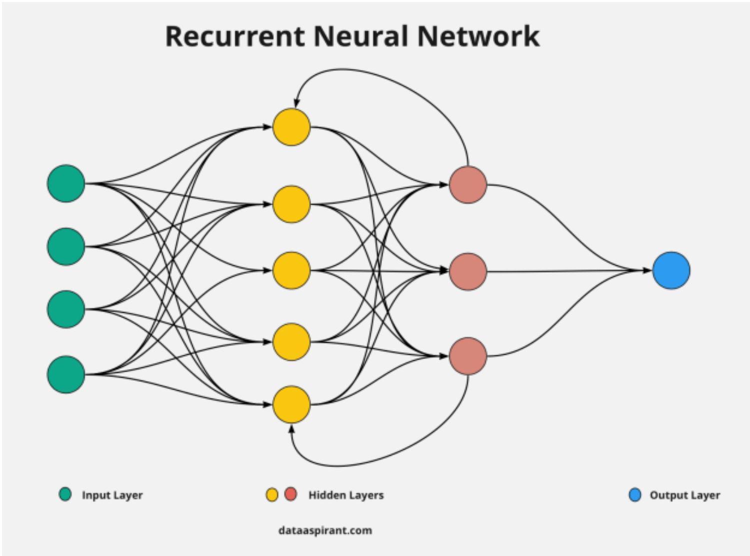


Fig 1.13 RNN Architecture

Due to the shorter memory capabilities of traditional RNNs which causes the problem known as vanishing gradient, **LSTM (Long Short Term Memory)** is used as they can selectively remember sequences for long durations of time. LSTM has 3 gates – input, output and forget gates which are able to control the drift of data in and out of the cell.

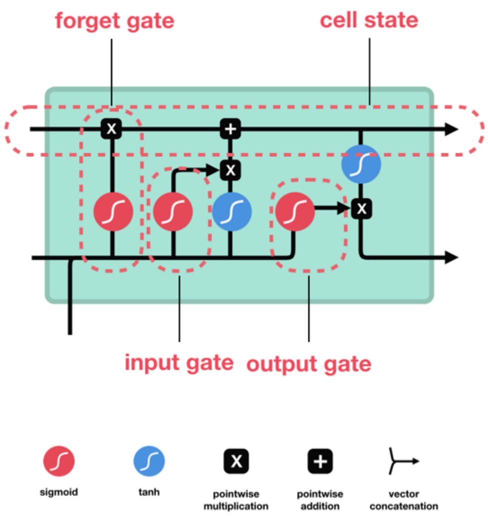


Fig 1.14 LSTM Architecture

One of the main restrictions of the encoder-decoder architecture is that it is only applicable for smaller sequences. The encoder struggles to remember longer sequences into a vector of fixed dimension. In order to solve this problem, attention mechanism is used. In attention mechanism, certain parts of the source sequence is given more importance to derive at a target sequence. Based on the means the context vector is derived, there are 2 kinds of attention mechanisms – Global or Local Attention. In global attention, all the hidden states of the encoder are considered whereas for local attention only a scarce number of hidden states of the encoder are taken into consideration while stemming the context vector. In this paper, global attention mechanism is used.

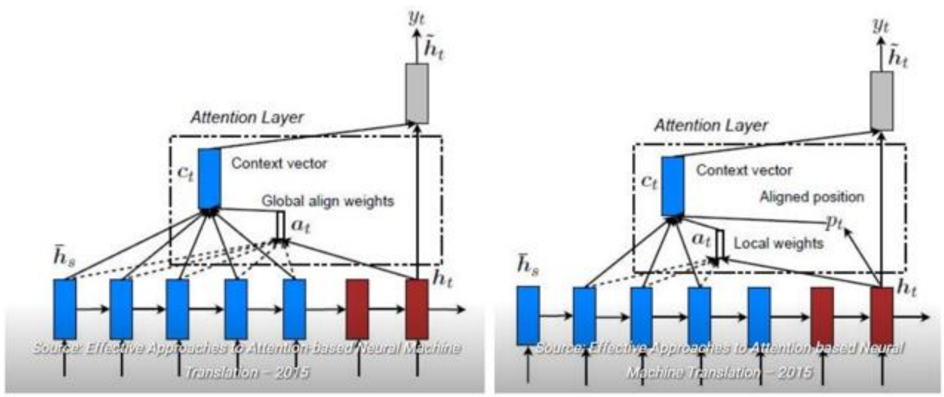


Fig 1.15 Attention Layers (Global and Local)

1.3 SENTIMENT CLASSIFIER

Sentiment analysis involves identifying opinions in a text and classifying them as positive, negative or neutral by using NLP techniques such as tokenization, lemmatization, bag-of-words etc. The main aim is to construct a system that can analyse the semantics of a text and use natural language data to understand the context and emotions behind the text.

There are generally 3 types of sentiment classifiers:

- Rule Based systems- They count the number of positive (lexicon) or negative terms appearing in the text to decide whether it is positive or negative based on which occurs the most.
- Automated systems- These systems are mainly based on machine learning algorithms. They use supervised(training) data to learn to predict sentiment and try to find a pattern that will help classify.
- Hybrid systems- They are a combination of both Rule based and Automated systems. A model is built that learns to predict sentiment and then it is compared with lexicons to further boost the accuracy.

In this project, a deep learning method is applied for Sentiment Classification by means of **bidirectional LSTM-GRU**. As mentioned before, RNN has a "memory" component which stores information from previous states and uses them as parameters to produce an output. This helps the model to understand the context which is very important when trying to deduce the emotion behind the review. In this paper, a combination of bidirectional LSTM-GRU is used. **GRU (Gated Reccurent Unit)** uses the hidden state to transfer information instead of the cell state. GRU has only 2 gates- reset and update gate and is hence faster than LSTM as it involves lesser tensor operations.

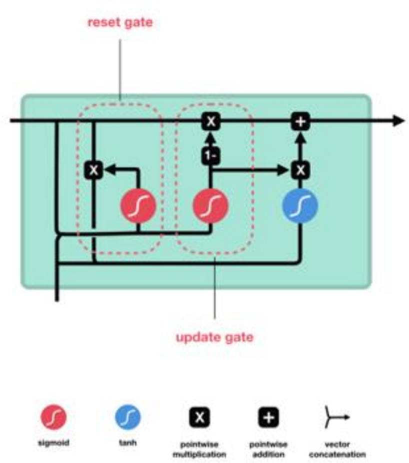


Fig 1.16 GRU Architecture

For word embedding, GloVe embedding is preferred over word2vec in this paper as GloVe focuses on words co-occurrences instead of taking the whole text as training data like word2vec. GloVe embedding is an unsupervised learning algorithm that is based on matrix factorization techniques on the word-content matrix.

Bidirectional LSTMs are used to enhance the performance of models for problems involving sequence classification over old-style LSTMs as dual LSTMs are trained on an input instead of a single one. This structure has both a forward propogation and a backward propogation which gives extra information about the sequence and thus increases context understanding. Context is a very important part of NLP as it can decide whether a sentiment is positive or negative. For example, “not good enough” is a negative review. However a basic system that does not understand the context would classify it as a positive sentiment due to the presence of the word “good”. Bidirectional GRU or any other RNN works in the same way.

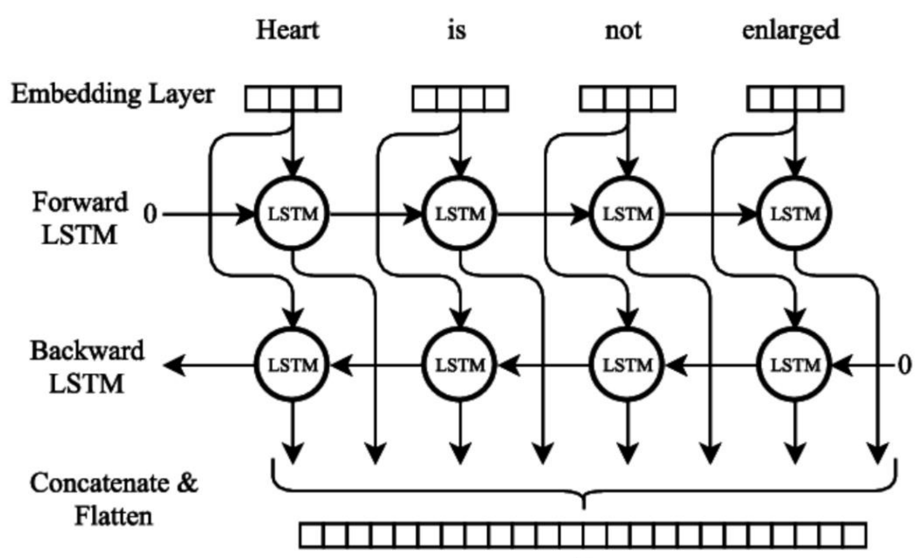
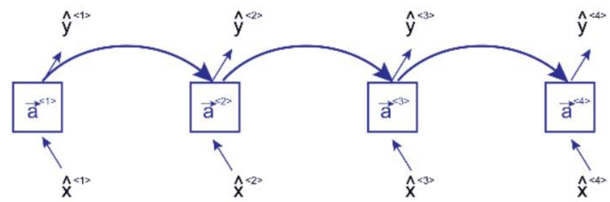


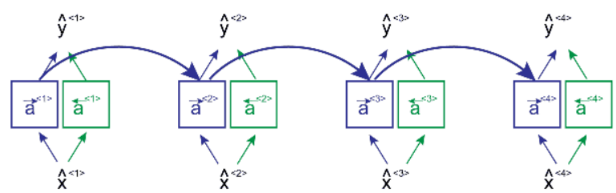
Fig 1.17 Bidirectional LSTM Architecture

The steps involved in a Biderectional RNN are as follows



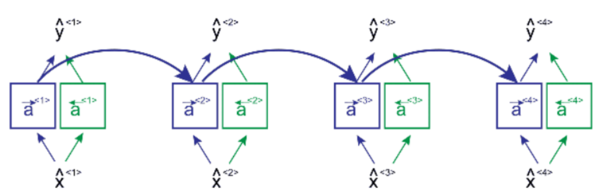
Forward RNN (LSTM or GRU) network

1.



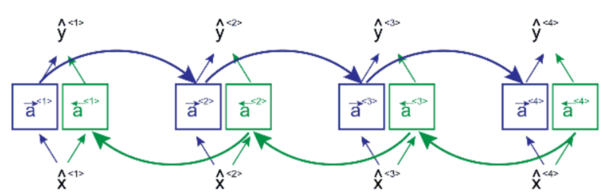
Adding another cells for the other direction

2.



Forward RNN (LSTM or GRU) network

3.



Connecting the backward cells

4.

Fig 1.18 Bidirectional RNN working

1.4 MULTITHREADING

Threads are a lightweight version of a process. Multithreading leads to maximum CPU utilisation as it allows execution of multiple parts of the program concurrently, In python, multithreading is used to execute multiple threads simultaneously. Python has an inbuilt module called multiprocessing that allows for multithreading.

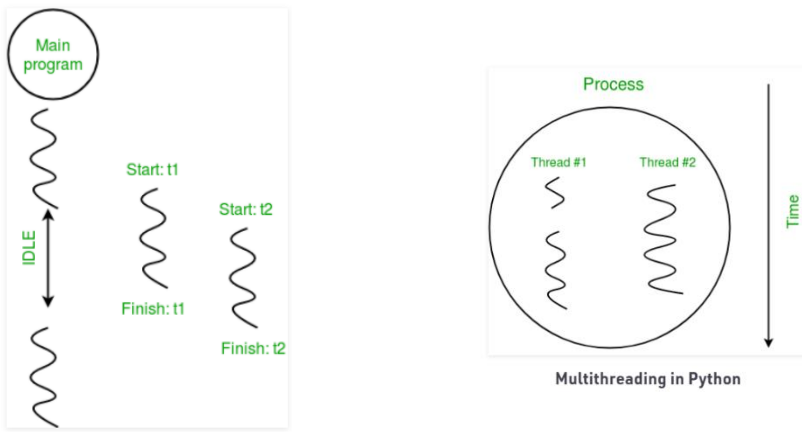


Fig 1.19 Multithreading in Python

1.5 OBJECTIVES

The main objective of this project is to create a system that when provided with a text review:

- Will classify it as suspicious or not
- Will summarize the review using a phrase
- Will provide a (0-5) rating based on sentiment classification of the review.

1.6 SCOPE OF THE PROJECT

The scope of the project is to build a multi-threaded GUI based application that takes a user review as an input and analyzes it to classify it as suspicious or not, summarize it and predict a rating for the review based on sentiment classification with a relatively low runtime and high accuracy using deep learning techniques.

Chapter 2

Experiment Setting

2.1 TECHNOLOGY STACK

This project is completely implemented using Python 3.8.5 and Anaconda Environment. The model defining, training and testing phases are done on Jupyter notebooks and the front end building and integration of the modules is done on Spyder IDE .

```
(base) C:\Users\bioni>python --version
Python 3.8.5
```

Fig 2.1 Python Version

Multiple python libraries are used in this project but some of the main ones are shown in table 2.1:

Table 2.1 Python Libraries used

library	Function
numpy	is the fundamental package for scientific computing in Python.
pandas	is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool
keras	is a powerful and easy-to-use free open source Python library for developing and evaluating deep learning models
sklearn	provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python.
matplotlib	is a comprehensive library for creating static, animated, and interactive visualizations in Python.
pickle	is used for serializing and de-serializing python object structures and saving for future use
nltk	is a platform for building Python programs to work with human language data.
threading	Is used for implementing thread based parallelism

2.2 CHALLENGES

- Integration – Since 3 models are built and trained separately, it is a tedious task to integrate them into a single program and make them work in a pipeline, especially a multithreaded pipeline.

- Dataset differences – a specific labeled dataset that can be used to train all the 3 models was not found. Therefore different datasets are used to train the models using the transfer learning technique.

- Accuracy – since the datasets are not standardised, the accuracy might not be the best possible.

- Computational constraints – The computational powers and resources for training a deep learning model are very high and it is only possible to implement a scaled down version.

Chapter 3

Data Exploration

2.1 AMAZON REVIEWS DATASET

The dataset used is a recently released corpus of Amazon Reviews that is labelled as Fake or Real obtained from Kaggle.

```
df = pd.read_csv("amazon_reviews.txt", delimiter = "\t")
```

```
len(df)
```

```
21000
```

The dataset consists of 21,000 reviews, equally distributed across product categories, which have been identified as 'non-compliant' with respect to Amazon policies. It also has a lot of additional features for each review such as rating, verified purchase, product category, product ID, product title and review title which can help us improve the performance of the SVM.

```
df.columns
```

```
Index(['DOC_ID', 'LABEL', 'RATING', 'VERIFIED_PURCHASE', 'PRODUCT_CATEGORY',
       'PRODUCT_ID', 'PRODUCT_TITLE', 'REVIEW_TITLE', 'REVIEW_TEXT'],
      dtype='object')
```

```
df.dtypes
```

```
DOC_ID                int64
LABEL                object
RATING                int64
VERIFIED_PURCHASE    object
PRODUCT_CATEGORY     object
PRODUCT_ID           object
PRODUCT_TITLE        object
REVIEW_TITLE         object
REVIEW_TEXT          object
dtype: object
```

The reviews are classified as fake or real (in the data frame they're mapped fake (__label1__) or real (__label2__)).

```
df.head()
```

	DOC_ID	LABEL	RATING	VERIFIED_PURCHASE	PRODUCT_CATEGORY	PRODUCT_ID	PRODUCT_TITLE	REVIEW_TITLE	REVIEW_TEXT
0	1	__label1__	4	N	PC	B00008NG7N	Targus PAUK10U Ultra Mini USB Keypad, Black	useful	When least you think so, this product will sav...
1	2	__label1__	4	Y	Wireless	B00LH0Y3NM	Note 3 Battery : Stalion Strength Replacement ...	New era for batteries	Lithium batteries are something new introduced...
2	3	__label1__	3	N	Baby	B000I5UZ1Q	Fisher-Price Papasan Cradle Swing, Starlight	doesn't swing very well.	I purchased this swing for my baby. She is 6 m...
3	4	__label1__	4	N	Office Products	B003822IRA	Casio MS-80B Standard Function Desktop Calculator	Great computing!	I was looking for an inexpensive desk calcolat...
4	5	__label1__	4	N	Beauty	B00PWSAXAM	Shine Whitening - Zero Peroxide Teeth Whitenin...	Only use twice a week	I only use it twice a week and the results are...

A word cloud is plotted to give us an idea as to which are the words that are used most frequently in the reviews.

```
wordcloud = WordCloud(width = 3000, height = 2000, random_state=1, background_color='black',
                      colormap='Set2', stopwords = STOPWORDS).generate(text)
plt.figure()
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show(text)
```

Fig 2.1 Wordcloud

Textstat is a library used to calculate statistics from text that helps determine readability, complexity, and grade level. According to AWAI, FK or Flesch Kincaid is a statistical program that measures the simplicity of writing and can determine how easy or difficult it is to understand the writer.

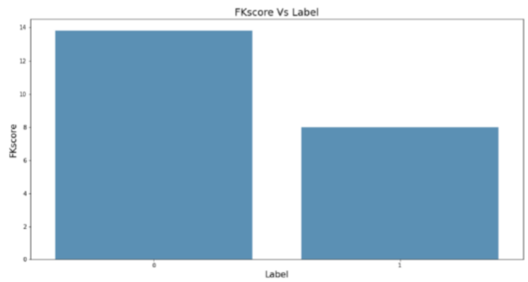


Fig 2.2 FKscore vs Label

Various graphs are drawn to give an idea of the occurances of stopwords, capitals, punctuations, emojis in the reviews dataset for each label.

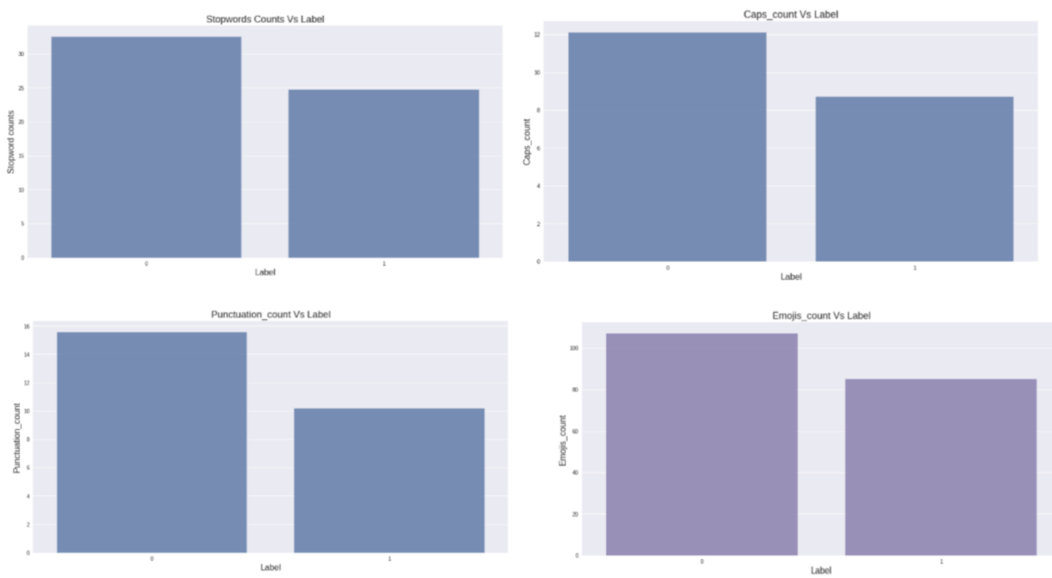


Fig 2.3 Graphs for Label vs Occurances

2.2 AMAZON FINE FOOD REVIEWS DATASET

The dataset used is reviews of Amazon fine foods for a period of 10 years (1999-2012) from Kaggle.

```
data=pd.read_csv("Reviews.csv")
```

```
len(data)
```

```
568454
```

The dataset consists of about 500000 + fine food reviews. It also has a lot of additional features for each review such as helpfulnessNumerator, helpfulnessDenominator, score, time, summary etc.

```
data.columns
```

```
Index(['Id', 'ProductId', 'UserId', 'ProfileName', 'HelpfulnessNumerator',
       'HelpfulnessDenominator', 'Score', 'Time', 'Summary', 'Text'],
      dtype='object')
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 568454 entries, 0 to 568453
Data columns (total 10 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   Id                      568454 non-null  int64
 1   ProductId               568454 non-null  object
 2   UserId                  568454 non-null  object
 3   ProfileName             568438 non-null  object
 4   HelpfulnessNumerator    568454 non-null  int64
 5   HelpfulnessDenominator  568454 non-null  int64
 6   Score                   568454 non-null  int64
 7   Time                    568454 non-null  int64
 8   Summary                 568427 non-null  object
 9   Text                    568454 non-null  object
dtypes: int64(5), object(5)
memory usage: 43.4+ MB
```

The mainly used colums for training and testing the model are the summary and text columns.

```
data.head()
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	5	1303862400	Good Quality Dog Food	I have bought several of the Vitality canned dog food products and have found them all to be of good quality. The product looks more like a stew than a processed meat and it smells better. My Labr...
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	1	1346976000	Not as Advertised	Product arrived labeled as Jumbo Salted Peanuts...the peanuts were actually small sized unsalted. Not sure if this was an error or if the vendor intended to represent the product as "Jumbo".
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	4	1219017600	"Delight" says it all	This is a confection that has been around a few centuries. It is a light, pillowy citrus gelatin with nuts - in this case Filberts. And it is cut into tiny squares and then liberally coated with ...
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	3	2	1307923200	Cough Medicine	If you are looking for the secret ingredient in Robitussin I believe I have found it. I got this in addition to the Root Beer Extract I ordered (which was good) and made some cherry soda. The fl...
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	5	1350777600	Great taffy	Great taffy at a great price. There was a wide assortment of yummy taffy. Delivery was very quick. If your a taffy lover, this is a deal.

It is observed that most of the reviews have received a score of 5 on a scale of 1-5.

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.countplot(data['Score'])
plt.show()
```

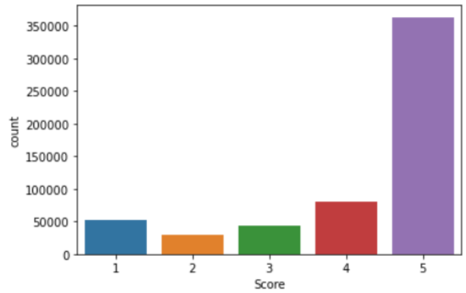


Fig 2.4 Reviews vs Score

A scatterplot matrix is used to determine the correlation between the multiple variables and serves as an estimation of the covariance matrix.

```
from pandas.plotting import scatter_matrix
pd.plotting.scatter_matrix(data, alpha=0.2, figsize=(10, 10))
plt.show()
```

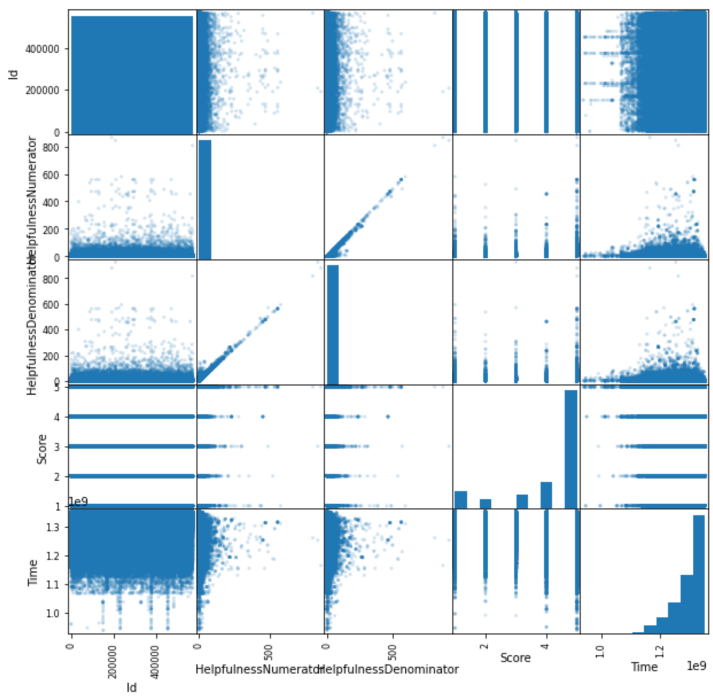


Fig 2.5 ScatterPlot Matrix

2.3 IMDB MOVIE REVIEWS DATASET

The dataset used is IMDB movie reviews hosted by Kaggle for binary sentiment classification. The technique of transfer learning is implemented to train the model on this dataset and then use it for sentiment classification of Amazon reviews.

```
df1=pd.read_csv('IMDB.csv')
```

```
len(df1)
```

```
50000
```

It contains 50,000 movie reviews that are equally distributed between postivie and negative sentiment. Each review only contains the review text and its sentiment.

```
df1.columns
```

```
Index(['review', 'sentiment'], dtype='object')
```

```
df1['sentiment'].value_counts()
```

```
positive    25000
negative    25000
Name: sentiment, dtype: int64
```

```
df1.describe()
```

	review	sentiment
count	50000	50000
unique	49582	2
top	Loved today's show!!! It was a variety and not...	positive
freq	5	25000

There are 49,852 unique reviews that can be used to train and test our sentiment classifier model built using deep learning techniques.

```
df1.head()
```

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive

Chapter 4

Literature Survey

In recent years, the amount of research being done in the field of Natural Language Processing (NLP) and Deep Learning (DL) have gone up drastically. A lot of studies have proposed different models and techniques for tackling the problem of NLP and improving accuracy especially when it comes to RNNs such as LSTM and GRU.

S Feng [1] has put forth a hypothesis that there are natural distributions of opinions in product reviews. A shady company that hires people to write fake reviews will purposely distort its distribution of review scores in order to seem legitimate, thus leaving distributional footprints behind which can be used to identify suspicious reviews.

The paper by Seung Ah Choi [2] is the most recent work and it aims to identify fake product reviews on Amazon.com through semantic analysis and using additional metadata such as words used, time, and the reviewer details. This paper proposes certain features that could be key to identifying fake reviews such as looking for exaggerated or professional words and if the review is a carbon copy of another review which may indicate that the review was copy pasted numerous times and is fake. Some attributes of the review such as length of the review and number of helpful votes are also considered in addition to checking if the review comprises a related photograph of the product or if any term from the product title or product category is stated in the review.

A Review Factor Graph model has been suggested for simultaneously detecting fake reviews and review spammers by Y Lu et al. [3]. The model incorporates all of the features and takes advantage of belief transmission between reviews and reviewers, by describing features to characterise each review and reviewer. This algorithm seems to outperform all other baseline approaches in terms of performance and precision by a large margin.

In collaboration with Jeremy Duns, T Fornaciari and M Poesio [4] developed a set of guidelines to find suspicious reviews and compiled a database of reviews, some of which are unquestionably fraudulent (as the authors admitted), while others may be genuine or misleading. They assigned a class to each analysis in the dataset using Raykar et al [5]'s learning from crowds algorithm. It involves the use of trained models using supervised methods with the aim of identifying fake reviews. Depending on the number of deceptive clues found, the review was classified as fake or not. The features were just of unigrams, bigrams and trigrams of lemmas and part-of-speech (POS), as collected from the reviews through TreeTagger10 (Schmid, 1994). The best results were attained by using Support Vector Machines (SVMs).

Mohammad Ehsan Basiri et al. [6] have projected an Attention-based Bidirectional CNN-RNN Deep Model for sentiment analysis of twitter datasets which can understand context due to the temporal flow of information provided by the forward and backward propogation of sequence in the bidrectional RNNs. An attention mechanism is used to further add emphasis on different words which helped the model reach state of the art results on both long and short reviews. Ruales et al. [7] created a sentiment classification of movie reviews found in the internet in which the model vector representation is used for visualization and word retrieval. However he concluded that using LSTM was not a significant improvement to using regular RNNs.

Trofimovich J. et al [9] in their paper, have compared three different neural network models for Twitter sentiment analysis. They have used SVM classifier, Convolutional Neural Networks(CNN) and Gated Recurrent Neural Networks (RNN- GRU) and word2vec model for word embedding. Based on their test results and evaluations, the GRU model had the best performance. Sharat Sachin et al. [8] have implemented LSTM, GRU and Bi-LSTM and Bi-GRU models for sentiment analysis on an Amazon review dataset.

Mehmet Umut Salur and Ilhan Aydin [10] have put forth an innovative hybrid deep learning model in their paper that syndicates multiple word embedding algorithms (FastText, Word2Vec) with diverse deep learning approaches (CNN, GRU, LSTM, BiLSTM). Their model displays better sentiment classification performance than the basic deep learning models of the past.

Table 4.1 shows examples of a few hybrid models that were proposed for Sentiment Clasification problems.

Table 4.1 Survey of Hybrid Models

Research	Model Used
Luo [11]	LDA representation, CNN, GRU
Jabreel et al. [12]	Target-dependent Bi-GRU, Bi-RNN
Shen et al. [13]	Bi-LSTM, Bi-GRU, Bi-SRU, CNNs, multi-head attention, DiSAN
Majumder et al. [14]	Deep CNN, GRU
Piao et al. [15]	CNNs and RNNs, word embedding
Wang et al. [16]	RNN, LSTM model, GRU, Bi-LSTM, CNN and CNN-tensor
Penghua et al. [17]	GRU, CNN, Bi-GRU, TC-LSTM, ATAE-LSTM, attention-based Bi-GRU
Bjerva et al. [18]	ResNets, GRU
Zhou et al. [19]	Bi-LSTM, Bi-GRU, LSTM and GRU

Ramesh Nallapati et al. [20] have proposed using a Attentional Encoder-Decoder Recurrent Neural Networks on a dataset consisting of multi-sentence summaries for abstractive text summarization and also established performance benchmarks for further works and try to address the critical problems found in text summarization with a RNN solution.

Table 4.2 shows examples of a few bidirectional neural network models that were proposed for Sentiment Clasification problems.

Table 4.2 Survey of Bidirectional NN Models

Research	Model Used
Minh Dang et al. [21]	Two-stream GRU
Wang et al. [22]	Bi-GRU with attention, word embeddings
Zhang et al. [23]	Bi-RNN, Bi-GRU
Huang et al. [24]	Bi-LSTM, sentiment-specific word embeddings
Wu et al. [25]	Bi-GRU, context-aspect hierarchical attention network

In a study on text summarization by N. Moratanch and S. Chitrakala [27], it is stated that a maximum of the abstractive summarization approaches produces coherent, highly amalgamated, fewer redundant summary and information aplenty. Linqing Liu et al. [28] in their paper, have proposed using a Generative Adversarial Network (GAN) for modeling an abstractive text summarizer. simultaneously train a generative model G which uses reinforcement learning and a discriminative model D to provide more abstractive, readable and diverse summaries.

Shengli et al. [29] have put forth a LSTM-CNN based abstractive text summarizer which is capable of constructing novel sentences by discovering additional fine-grained fragments called semantic phrases. ATSDL has 2 critical phases, one extracts expressions from source sentences and the other produces text summaries by means of deep learning algorithms. Investigational results show that this model achieves competitive results.

Tian Shi et al. [30] analyzed various seq2seq models for abstractive text summarization and prepared a literature survey. In this survey, they have provided a brief assessment of several models that have been proposed for language generation and modeling tasks and that have been used in abstractive text summarization. In addition to this survey, they have built an open source library for text summarization called as Neural Abstractive Text Summarizer, abbreviated as the NATS toolkit and use this library for benchmarking.

From the above mentioned literature surveys, it has been pretty prominent that bidirectional LSTM-GRU architecture is the most apt design for the NLP problems involving sequential data.

Chapter 5

Methodology and Implementation

The proposed system has 3 individual modules – Suspicious Review classifier, Review summarizer and Sentiment classifier that are trained and compiled on different datasets. Once the models are trained, they are saved using the functions provided by the keras and pickle modules in Python and imported to the application that runs the three concurrently using multithreading module in Python on a given input review.

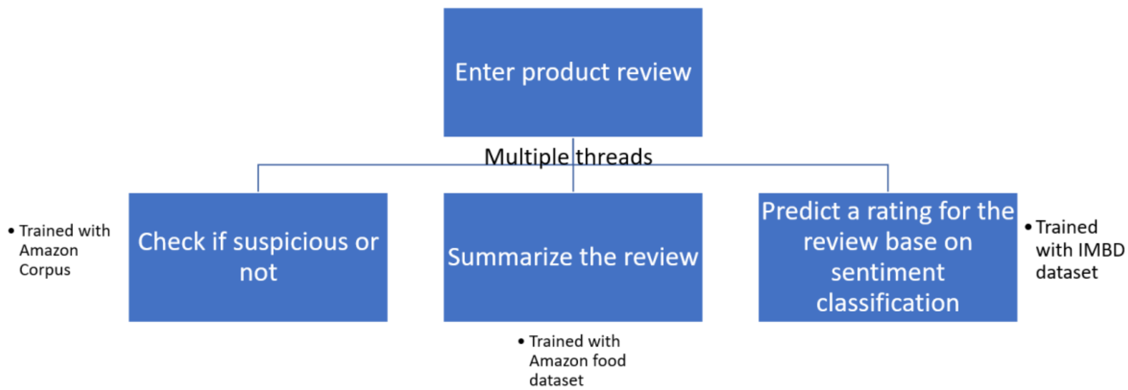


Figure 5.1 System Architecture

Each module works on the basic pipeline that is generally used when it comes to machine learning algorithms. First, the data is inspected to understand and preprocessed to clean and tokenize it based on the model to be used. The model is then built and trained on this data after which testing occurs.

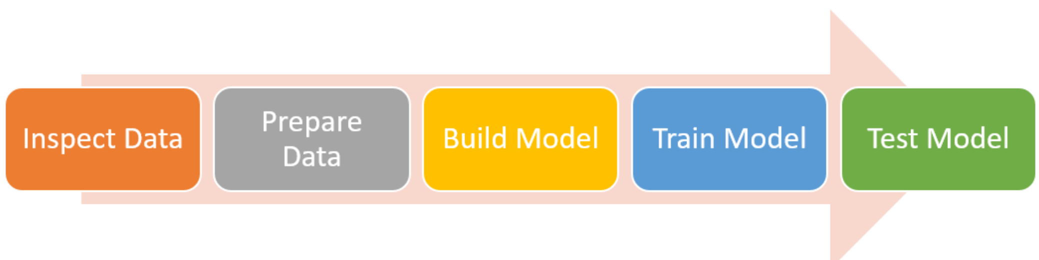


Figure 5.2 Model pipeline

5.1 SUSPICIOUS REVIEW CLASSIFIER

A simple SVM model is proposed for the suspicious review classifier. The workflow of the module is as follows:

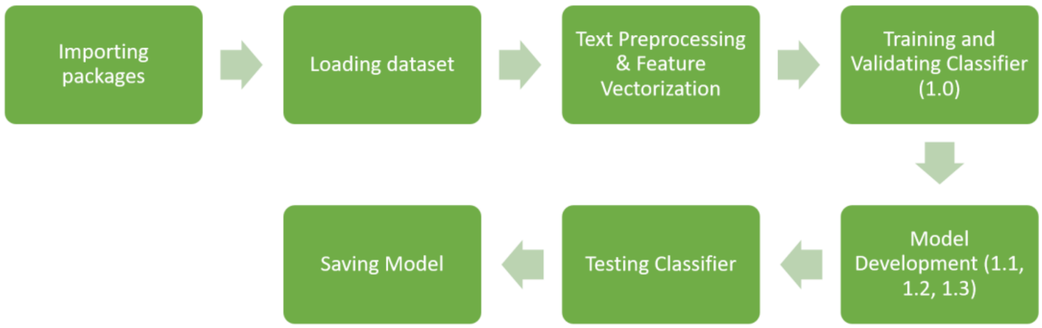


Figure 5.3 Suspicious Review Classifier workflow

The Python NLTK and sklearn libraries have been used to analyse and pre-process the dataset and Jupyter Notebook and other techniques are used to build an SVM classifier that can classify a review as suspicious or not. The Amazon reviews dataset (from Ch. 2.1) is used to train and test this model. The reviews are parsed and only the required columns are extracted after which tokenization is done. The tokens are transformed into a feature dictionary that has as its keys the tokens, and as values the weight of those tokens in the preprocessed reviews. A 80-20 split is used for training and testing data.

During the development of this module, multiple models of SVM were built to improve the accuracy.

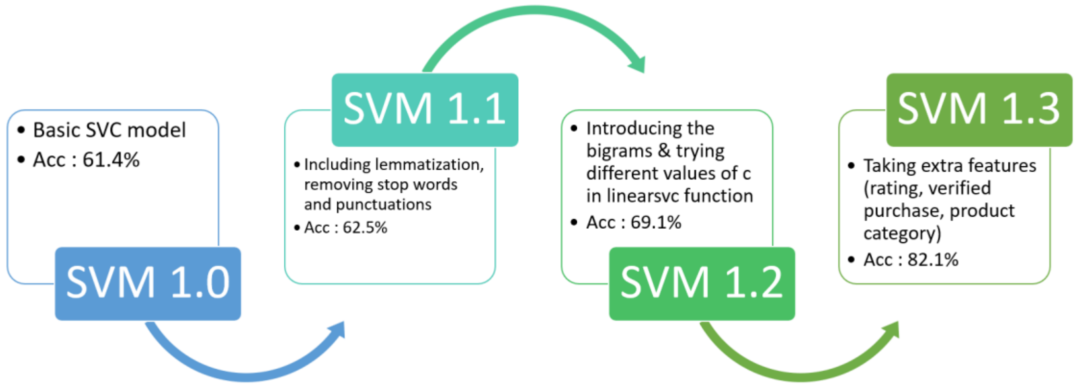


Figure 5.4 SVM development lifecycle

The SVM 1.0 is a basic SVC model from sklearn library of Python. The pre-processing for this model is just an inbuilt function after which feature vectorization is done. Upon training and crossvalidating with a 10-fold cross validation, the module returned a very low accuracy of 61.4%. But this is just a base SVM which was used without any pre-processing or smart weights and is expected to have a low accuracy.

```
# TEXT PREPROCESSING AND FEATURE VECTORIZATION
def preProcess(text):
    return word_tokenize(text)

featureDict = {} # A global dictionary of features
def toFeatureVector(tokens):
    localDict = {}
    for token in tokens:
        if token not in featureDict:
            featureDict[token] = 1
        else:
            featureDict[token] = +1

        if token not in localDict:
            localDict[token] = 1
        else:
            localDict[token] = +1

    return localDict
```

For SVM 1.1, stop words and punctations are removed in preprocessing. Stop words are words that are commonly found in text and are therefore of no significance use as they carry very little useful information and can be removed for increase in performance when it comes to NLP. In addition, Lemmatization is also implemented under preprocessing. Lemmatization is converting the word into a basic form (like stemming) so that different words with same meaning can be grouped and analysed together. It usually involves removing the prefix/suffix/tense etc. The base form of a word is called lemma. Python provides various packages that can be used for these functions. When trained and crossvalidated with 10-folds, it returned an accuracy of 62.5%.

```
# TEXT PREPROCESSING AND FEATURE VECTORIZATION

table = str.maketrans({key: None for key in string.punctuation})

def preProcess(text):
    lemmatizer = WordNetLemmatizer()
    filtered_tokens=[]
    stop_words = set(stopwords.words('english'))
    text = text.translate(table)
    for w in text.split(" "):
        if w not in stop_words:
            filtered_tokens.append(lemmatizer.lemmatize(w.lower()))
    return filtered_tokens
```

In SVM 1.2, bigrams are introduced which are a sequence of 2 words or tokens. A bigram model is an n-gram model that forecasts the probability of a given bigram within any sequence of words in the language. The addition of bigrams makes the language model more sensitive to the input text.

```
# TEXT PREPROCESSING AND FEATURE VECTORIZATION
table = str.maketrans({key: None for key in string.punctuation})
def preProcess(text):
    lemmatizer = WordNetLemmatizer()
    filtered_tokens=[]
    lemmatized_tokens = []
    stop_words = set(stopwords.words('english'))
    text = text.translate(table)
    for w in text.split(" "):
        if w not in stop_words:
            lemmatized_tokens.append(lemmatizer.lemmatize(w.lower()))
        filtered_tokens = [' '.join(l) for l in nltk.bigrams(lemmatized_tokens)] + lemmatized_tokens
    return filtered_tokens
```

Further, the value of the C parameter which indicates the support vector machine optimization how much to evade misclassifying each training instance is changed to 0.01 during training as it gave the best results on experimentation. This model returned an accuracy of 69.1% on crossvalidation.

```
# TRAINING AND VALIDATING OUR CLASSIFIER
def trainClassifier(trainData):
    print("Training Classifier...")
    pipeline =  Pipeline([('svc', LinearSVC(C=0.01))])
    return SklearnClassifier(pipeline).train(trainData)
```

For SVM 1.3, additional features available in the dataset (Rating, Verified Purchase, Product Catergory) are taken into consideration as well. The presence of these additional features helps the model understand the context better and predict with a better accuracy than was possible previously. When trained and crossvalidated on 10-folds it returned an accuracy of 82.1% which is observed to be the highest among all the models.

```
featureDict = {} # A global dictionary of features
def toFeatureVector(Rating, verified_Purchase, product_Category, tokens):
    localDict = {}
#Rating
    featureDict["R"] = 1
    localDict["R"] = Rating
#Verified_Purchase
    featureDict["VP"] = 1

    if verified_Purchase == "N":
        localDict["VP"] = 0
    else:
        localDict["VP"] = 1
#Product_Category
    if product_Category not in featureDict:
        featureDict[product_Category] = 1
    else:
        featureDict[product_Category] = +1

    if product_Category not in localDict:
        localDict[product_Category] = 1
    else:
        localDict[product_Category] = +1
#Text
    for token in tokens:
        if token not in featureDict:
            featureDict[token] = 1
        else:
            featureDict[token] = +1

        if token not in localDict:
            localDict[token] = 1
        else:
            localDict[token] = +1

    return localDict
```

When the model is run on test data, it gives an accuracy of about 80%.

```
# TEST DATA
classifier = trainClassifier(trainData)
predictions = predictLabels(testData, classifier)
true_labels = list(map(lambda d: d[1], testData))
a = accuracy_score(true_labels, predictions)
p, r, f1, _ = precision_recall_fscore_support(true_labels, predictions, average='macro')
print("accuracy: ", a)
print("Precision: ", p)
print("Recall: ", a)
print("f1-score: ", f1)
```

```
Training Classifier...
accuracy:  0.8042857142857143
Precision:  0.8080454049606811
Recall:  0.8042857142857143
f1-score:  0.8036867139280308
```

5.2 REVIEW SUMMARIZER

For the abstractive review summarizer, an Encoder-Decoder architecture is implemented using RNN and LSTM. The workflow of the module is as follows:

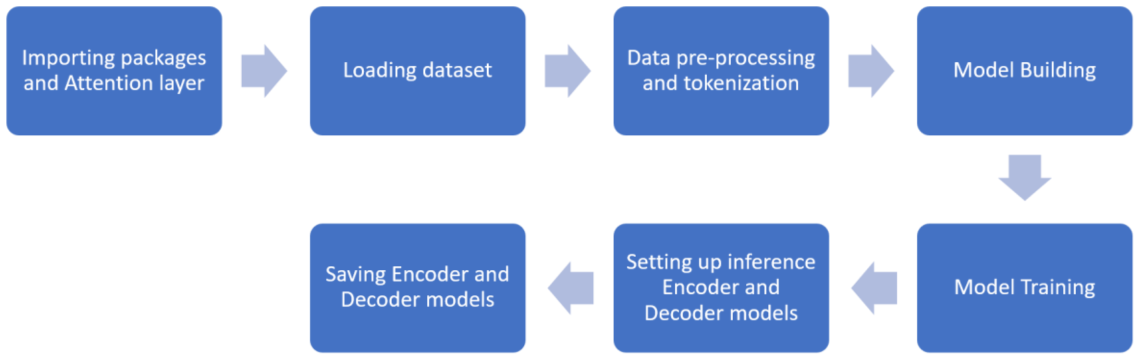


Figure 5.5 Text Summarizer workflow

There is no official support for the attention layer provided by Keras yet. So, a third-party application of the attention layer is imported from Github and saved in a file **attention.py**.

After importing the attention layer and the packages required, the dataset is loaded using the Pandas dataframe. The next key step is to clean the data which takes place in several steps. First, duplicate and empty/null reviews out of the 500,000 reviews are dropped from the dataframe. Then a function called text_cleaner is defined which carries out the following tasks:

- Convertion to lower case
- Removal of html tags
- Contraction mapping
- Removal of (‘s)
- Removal of text inside parethesis ()
- Removal of punctuations and special characters
- Stopwords removal
- Shortwords removal

In an attempt to reduce the otherwise very lengthy training time, a sample of 100,000 reviews is used instead of the complete dataset.

```
import re
import nltk
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))

def text_cleaner(text,num):
    newString = text.lower()
    newString = BeautifulSoup(newString, "lxml").text
    newString = re.sub(r'\([^)]*\)', '', newString)
    newString = re.sub('"','', newString)
    newString = ' '.join([contraction_mapping[t] if t in contraction_mapping else t for t in newString.split(" ")])
    newString = re.sub(r"'s\b","",newString)
    newString = re.sub("[^a-zA-Z]", " ", newString)
    newString = re.sub('[m]{2,}', 'mm', newString)
    if(num==0):
        tokens = [w for w in newString.split() if not w in stop_words]
    else:
        tokens=newString.split()
    long_words=[]
    for i in tokens:
        if len(i)>1:                                                #removing short word
            long_words.append(i)
    return (" ".join(long_words)).strip()
```

On analyzing the cleaned sentences, it is seen that the majority of the reviews (94%) have a maximum text length of 8 and max summary length of the reviews is always under 30, so these values are fixed as the maximum lengths of the sequence.

```
import matplotlib.pyplot as plt
text_word_count = []
summary_word_count = []
for i in data['cleaned_text']:
      text_word_count.append(len(i.split()))
for i in data['cleaned_summary']:
      summary_word_count.append(len(i.split()))
length_df = pd.DataFrame({'text':text_word_count, 'summary':summary_word_count})
length_df.hist(bins = 30)
plt.show()
```

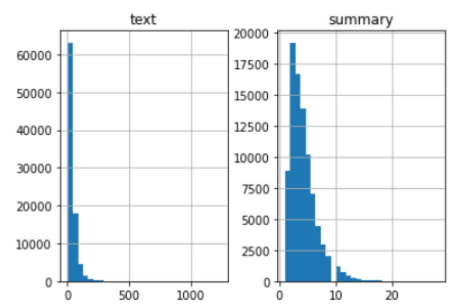


Fig 5.6 Length of texts and sequences

Special tokens are added in the beginning and end of a review. sostok is used as start token and eostok is used as end token. The dataset is split into 90-10 for training and testing. 2 tokenizers are built that convert the word sequence into integer sequence.

x_tokenizer is the review tokenizer and y_tokenizer is the summary tokenizer.

```
#prepare a tokenizer for reviews on training data
x_tokenizer = Tokenizer(num_words=tot_cnt-cnt)
x_tokenizer.fit_on_texts(list(x_tr))

#convert text sequences into integer sequences
x_tr_seq    =   x_tokenizer.texts_to_sequences(x_tr)
x_val_seq   =   x_tokenizer.texts_to_sequences(x_val)

#padding zero upto maximum length
x_tr    =   pad_sequences(x_tr_seq,  maxlen=max_text_len, padding='post')
x_val   =   pad_sequences(x_val_seq, maxlen=max_text_len, padding='post')

#size of vocabulary ( +1 for padding token)
x_voc   =   x_tokenizer.num_words + 1
```

```
#prepare a tokenizer for reviews on training data
y_tokenizer = Tokenizer(num_words=tot_cnt-cnt)
y_tokenizer.fit_on_texts(list(y_tr))

#convert text sequences into integer sequences
y_tr_seq    =   y_tokenizer.texts_to_sequences(y_tr)
y_val_seq   =   y_tokenizer.texts_to_sequences(y_val)

#padding zero upto maximum length
y_tr    =   pad_sequences(y_tr_seq, maxlen=max_summary_len, padding='post')
y_val   =   pad_sequences(y_val_seq, maxlen=max_summary_len, padding='post')

#size of vocabulary
y_voc  =   y_tokenizer.num_words +1
```

After tokenization, the model is built. A 3 stacked LSTM is built for the encoder as the multiple layers built on top of one another lead to a better depiction of the sequence. The model consists of an Input layer, 2 embedding layers, 3 stacked lstm layers, attention layer and concatenation layer (Refer to Appendix 1). latent_dim is taken as 300 and the embedding_dim is taken as 100.

```
# Encoder
encoder_inputs = Input(shape=(max_text_len,))

#embedding layer
enc_emb =  Embedding(x_voc, embedding_dim,trainable=True)(encoder_inputs)

#encoder lstm 1
encoder_lstm1 = LSTM(latent_dim,return_sequences=True,return_state=True,dropout=0.4,recurrent_dropout=0.4)
encoder_output1, state_h1, state_c1 = encoder_lstm1(enc_emb)

#encoder lstm 2
encoder_lstm2 = LSTM(latent_dim,return_sequences=True,return_state=True,dropout=0.4,recurrent_dropout=0.4)
encoder_output2, state_h2, state_c2 = encoder_lstm2(encoder_output1)

#encoder lstm 3
encoder_lstm3=LSTM(latent_dim, return_state=True, return_sequences=True,dropout=0.4,recurrent_dropout=0.4)
encoder_outputs, state_h, state_c= encoder_lstm3(encoder_output2)

# Set up the decoder, using `encoder_states` as initial state.
decoder_inputs = Input(shape=(None,))

#embedding layer
dec_emb_layer = Embedding(y_voc, embedding_dim,trainable=True)
dec_emb = dec_emb_layer(decoder_inputs)

decoder_lstm = LSTM(latent_dim, return_sequences=True, return_state=True,dropout=0.4,recurrent_dropout=0.2)
decoder_outputs,decoder_fwd_state, decoder_back_state = decoder_lstm(dec_emb,initial_state=[state_h, state_c])

# Attention layer
attn_layer = AttentionLayer(name='attention_layer')
attn_out, attn_states = attn_layer([encoder_outputs, decoder_outputs])

# Concat attention input and decoder LSTM output
decoder_concat_input = Concatenate(axis=-1, name='concat_layer')([decoder_outputs, attn_out])

#dense layer
decoder_dense =  TimeDistributed(Dense(y_voc, activation='softmax'))
decoder_outputs = decoder_dense(decoder_concat_input)
```

```
model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 30)]         0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 30, 100)      844000      input_1[0][0]
__________________________________________________________________________________________________
lstm (LSTM)                     [(None, 30, 300), (N 481200      embedding[0][0]
__________________________________________________________________________________________________
input_2 (InputLayer)            [(None, None)]       0
__________________________________________________________________________________________________
lstm_1 (LSTM)                   [(None, 30, 300), (N 721200      lstm[0][0]
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, None, 100)    198900      input_2[0][0]
__________________________________________________________________________________________________
lstm_2 (LSTM)                   [(None, 30, 300), (N 721200      lstm_1[0][0]
__________________________________________________________________________________________________
lstm_3 (LSTM)                   [(None, None, 300),  481200      embedding_1[0][0]
                                                                 lstm_2[0][1]
                                                                 lstm_2[0][2]
__________________________________________________________________________________________________
attention_layer (AttentionLayer ((None, None, 300),  180300      lstm_2[0][0]
                                                                 lstm_3[0][0]
__________________________________________________________________________________________________
concat_layer (Concatenate)      (None, None, 600)    0           lstm_3[0][0]
                                                                 attention_layer[0][0]
__________________________________________________________________________________________________
time_distributed (TimeDistribut (None, None, 1989)   1195389     concat_layer[0][0]
==================================================================================================
Total params: 4,823,389
Trainable params: 4,823,389
Non-trainable params: 0
__________________________________________________________________________________________________
```

Sparse categorical cross-entropy is used as the loss function to overcome memory issues and a concept called early stopping is implemented during the training phase of the neural network to stop training when the validation loss seems to be on the upward curve or increases. The model is then trained on a batch size of 128 and 50 epochs with early stopping and validated on the holdout dataset.

```
from matplotlib import pyplot
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()
```

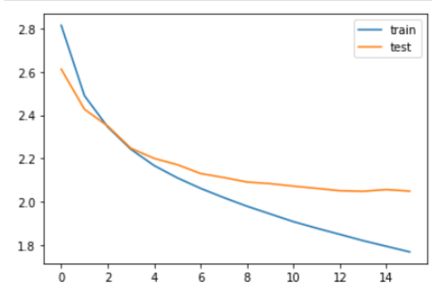


Fig 5.7 Validation loss plot

From the plot, it is visible that validation loss has increased after epoch 13 for 2 successive epochs and hence, due to early stopping mechanism implemented, training is stopped at epoch 16.

A dictionary is built to transform the index to word for the source and target vocabulary and saved using the pickle module.

For the inference, encoder and decoder models are set up and then saved using the keras function for future use.

```
# Encode the input sequence to get the feature vector
encoder_model = Model(inputs=encoder_inputs,outputs=[encoder_outputs, state_h, state_c])

# Decoder setup
# Below tensors will hold the states of the previous time step
decoder_state_input_h = Input(shape=(latent_dim,))
decoder_state_input_c = Input(shape=(latent_dim,))
decoder_hidden_state_input = Input(shape=(max_text_len,latent_dim))

# Get the embeddings of the decoder sequence
dec_emb2= dec_emb_layer(decoder_inputs)
# To predict the next word in the sequence, set the initial states to the states from the previous time step
decoder_outputs2, state_h2, state_c2 = decoder_lstm(dec_emb2, initial_state=[decoder_state_input_h, decoder_state_input_c])

#attention inference
attn_out_inf, attn_states_inf = attn_layer([decoder_hidden_state_input, decoder_outputs2])
decoder_inf_concat = Concatenate(axis=-1, name='concat')([decoder_outputs2, attn_out_inf])

# A dense softmax layer to generate prob dist. over the target vocabulary
decoder_outputs2 = decoder_dense(decoder_inf_concat)

# Final decoder model
decoder_model = Model(
    [decoder_inputs] + [decoder_hidden_state_input,decoder_state_input_h, decoder_state_input_c],
    [decoder_outputs2] + [state_h2, state_c2])
```

The decode_sequence function is defined to get the predicted summary given an input sequence.

```
def decode_sequence(input_seq):
    # Encode the input as state vectors.
    e_out, e_h, e_c = encoder_model.predict(input_seq)

    # Generate empty target sequence of length 1.
    target_seq = np.zeros((1,1))

    # Populate the first word of target sequence with the start word.
    target_seq[0, 0] = target_word_index['sostok']

    stop_condition = False
    decoded_sentence = ''
    while not stop_condition:

        output_tokens, h, c = decoder_model.predict([target_seq] + [e_out, e_h, e_c])

        # Sample a token
        sampled_token_index = np.argmax(output_tokens[0, -1, :])
        sampled_token = reverse_target_word_index[sampled_token_index]

        if(sampled_token!='eostok'):
            decoded_sentence += ' '+sampled_token

        # Exit condition: either hit max length or find stop word.
        if (sampled_token == 'eostok'  or len(decoded_sentence.split()) >= (max_summary_len-1)):
            stop_condition = True

        # Update the target sequence (of length 1).
        target_seq = np.zeros((1,1))
        target_seq[0, 0] = sampled_token_index

        # Update internal states
        e_h, e_c = h, c

    return decoded_sentence
```

Here are a few reviews from the dataset along with their original summary and the summary predicted by the model.

```
Review: great toy dogs chew everything else little literally eats toys one toys yet destroy loves carries around everywhere got
rex cutest thing
Original summary: good for chewers
Predicted summary:  dog loves it


Review: really search good deals tea tea great price tea amazon almost cup price cup coffee herbal varieties low caffine good o
ption wife used dinner coffe
Original summary: great price for great tea
Predicted summary:  tea


Review: pricey essentially small bag hard crumbs maybe dog spoiled treats like third class treats definitely bottom doggie trea
t often simply walk away glad people like buying
Original summary: waste of money
Predicted summary:  dog treats


Review: little pricey consider sugar low cal caffine really rich flavor best chai ever found
Original summary: fabulous product
Predicted summary:  good stuff



Review: absolutely delicious satisfy something sweet really filling great early morning time make breakfast great afternoon sna
ck work feeling sluggish
Original summary: protein bar
Predicted summary:  love these


Review: aware decaf coffee although showed search decaf cups intended purchase gift kept recipient drink caffeine favorite mean
s
Original summary: not decaf
Predicted summary:  not what expected


Review: wonderful wrote perfect iced cookie one pen writing cookies names happy ca
Original summary: cookie
Predicted summary:  delicious


Review: truffle oil quite good prefer brand france urbani italy expensive oh delicious tried black white good black bit stronge
r pungent event healthy alternative butter enjoy
Original summary: delicious but not the best
Predicted summary:  best olive oil ever


Review: enjoy coffee office split right middle loving think worth try order regularly
Original summary: hit or miss
Predicted summary:  good coffee


Review: husband gluten free food several years tried several different bread mixes first actually enjoys buying amazon saves lo
af
Original summary: really good gluten free bread
Predicted summary:  great gluten free bread
```

5.3 SENTIMENT CLASSIFIER

For the sentiment classifier, a RNN LSTM model is built for analysis of the text reviews. This model is then trained on the IMDB movie reviews dataset and transfer learning technique is implemented.

A standard sentiment classifier only classifies the sentiment as positive, neutral or negative. This project however proposes a model that will predict the probability of the sentiment being positive (on a scale of 0-1) which will then be converted into a rating for the review. The workflow of the module is as follows:

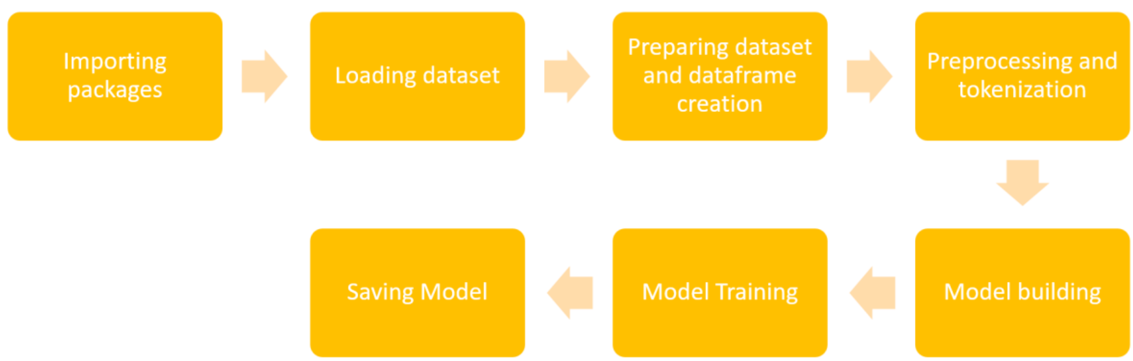


Figure 5.8 Sentiment Classifier workflow

Once the dataset is loaded, the pre-processing code performs the various processes that are required to format the reviews. These include:

- Removing punctuations
- Removing stopwords
- Removing stemwords/ Lemmatization
- Tockenization
- Word embedding (GloVe)

```
punctuations = string.punctuation

def punct_remover(my_str):
    my_str = my_str.lower()
    no_punct = ""
    for char in my_str:
        if char not in punctuations:
            no_punct = no_punct + char
    return no_punct

punctuations


tqdm.pandas()
reviews_train = train["text"].progress_apply(punct_remover)
reviews_test = test['text'].progress_apply(punct_remover)

def create_tokenizer(lines):
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(lines)
    return tokenizer


def max_length(lines):
    return max([len(s.split()) for s in lines])


def encode_text(tokenizer, lines, length):
    encoded = tokenizer.texts_to_sequences(lines)
    padded = pad_sequences(encoded, maxlen=length, padding='post')
    return padded
```

```
100%|██████████| 25000/25000 [00:09<00:00, 2551.28it/s]
100%|██████████| 10931/10931 [00:04<00:00, 2474.27it/s]
```

A 100 dimensional glove model trained on Wikipedia data is used to extract word embeddings for the reviews. An embedding matrix is built with dimensions as 121891 (vocabulary_size) * 300. Then it is tokenized using the python tokenizer.

```
#glove embedding

embeddings_index = dict()
f = open('Data/glove.840B.300d/glove.840B.300d.txt',encoding='utf8')
for line in f:
    values = line.split(" ")
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()
```

```
embed_token = create_tokenizer(reviews_train)
vocabulary_size = 121891

embedding_matrix = np.zeros((vocabulary_size, 300))
for word, index in embed_token.word_index.items():
    if index > vocabulary_size - 1:
        break
    else:
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            embedding_matrix[index] = embedding_vector
```

```
# create tokenizer
tokenizer = create_tokenizer(reviews_train)
length = max_length(reviews_train)
vocab_size = len(tokenizer.word_index) + 1
print('Max document length: %d' % length)
print('Vocabulary size: %d' % vocab_size)
length=800
trainX = encode_text(tokenizer, reviews_train, length)
testX = encode_text(tokenizer, reviews_test, length)
print(trainX.shape, testX.shape)

with open('Saved_models/tokenizer.pickle', 'wb') as handle:
    pickle.dump(tokenizer, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

The model architecture is defined with an input layer, convolutional layer, dropout layer, bidirectional LSTM and bidirectional GRU layers, maxpool layer, flatten layer and 2 dense layers with relu and sigmoid activations (refer to Appendix 1). Bidirectional LSTM-GRU is used as it increases the amount of imformation available to the neural network due to multiple propogations and hence improves the context available to the algorithm which makes sentiment prediction more accurate.

```
def new_mod(length, vocab_size):
    inputs1=Input(shape=(length,))
    embedding1 = Embedding(vocab_size,300,weights=[embedding_matrix])(inputs1)
    conv1 = Conv1D(filters=16, kernel_size=4, activation='relu')(embedding1)
    drop1 = Dropout(0.5)(conv1)
    lstm1 = Bidirectional(LSTM(20, return_sequences = True))(drop1)
    gru1 = Bidirectional(GRU(20, return_sequences = True))(lstm1)
    pool1 = MaxPooling1D(pool_size=2)(gru1)
    flat1 = Flatten()(pool1)
    dense1 = Dense(30, activation='relu')(flat1)
    outputs = Dense(1, activation='sigmoid')(dense1)
    model = Model(inputs=inputs1, outputs=outputs)
    model.compile(loss='binary_crossentropy', optimizer='nadam', metrics=['accuracy'])
    return model
```

Binary cross entropy is used as the loss function to adjust model weights during training phase and hence minimize the loss. Nadam optimizer (combination of NAG and Adam) is used as it is superior to the vanilla Adam optimizer when it comes to NLP using RNNs.

```
model = new_mod(length, vocab_size)
model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 800)]             0
_________________________________________________________________
embedding (Embedding)        (None, 800, 300)          36567300
_________________________________________________________________
conv1d (Conv1D)              (None, 797, 16)           19216
_________________________________________________________________
dropout (Dropout)            (None, 797, 16)           0
_________________________________________________________________
bidirectional (Bidirectional (None, 797, 40)           5920
_________________________________________________________________
bidirectional_1 (Bidirection (None, 797, 40)           7440
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 398, 40)           0
_________________________________________________________________
flatten (Flatten)            (None, 15920)             0
_________________________________________________________________
dense (Dense)                (None, 30)                477630
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 31
=================================================================
Total params: 37,077,537
Trainable params: 37,077,537
Non-trainable params: 0
_________________________________________________________________
```

The model is trained on a batch-size of 256 for 5 epochs and gives an accuracy of 96.3% after which it is saved in a .h5 format using the keras module for future use.

```
model.fit(trainX, pd.get_dummies(train['label'])['Postitive'].values, epochs=5, batch_size=256)
```

```
Epoch 1/5
98/98 [==============================] - 2209s 22s/step - loss: 0.6236 - accuracy: 0.6261
Epoch 2/5
98/98 [==============================] - 2345s 24s/step - loss: 0.3658 - accuracy: 0.8534
Epoch 3/5
98/98 [==============================] - 2432s 25s/step - loss: 0.1978 - accuracy: 0.9222
Epoch 4/5
98/98 [==============================] - 2709s 28s/step - loss: 0.1440 - accuracy: 0.9475
Epoch 5/5
98/98 [==============================] - 2970s 30s/step - loss: 0.1018 - accuracy: 0.9639
```

When a review is passed into the model, it predicts the probability of the review being a positive sentiment. When a negative review is encountered, the result is towards 0 and when a positive review is encountered, the result is towards 1.

```
model.predict(encode_text(tokenizer,["This product is a dissapointment. Please dont waste money to buy it."],length))
```

```
array([[0.02853474]], dtype=float32)
```

The following classification ranges are adopted for rating prediction.

(0.0-0.2) – 1 star

(0.2-0.4) – 2 star

(0.4-0.6) – 3 star

(0.6-0.8) – 4 star

(0.8-1.0) – 5 star

```
estimate=model.predict
rate=estimate[0]*50
print (rate[0])
```

Code snippet for calculation of rating

5.4 BACK-END OF APPLICATION

Once the three models are done training, they are saved in .h5 format along with any weights or tokenizers that are required to parse the review into the model for prediction. The prediction functions for each module is defined as separate functions in a python program.

f1(review) – Suspicious review classifier

f2(review) – Review summarizer

f3(review) – Sentiment classifier/ rating predictor

The concept of multithreading is used to run these 3 functions concurrently in parallel using 3 threads instead of one after another. Hence, 3 threads are initialized and each thread is assigned to a function.

```
start= time.time()

t1 = threading.Thread(target=f1(review))
t2 = threading.Thread(target=f2(review))
t3 = threading.Thread(target=f3(review))

t1.start()
t2.start()
t3.start()

t1.join()
t2.join()
t3.join()

end=time.time()
print("Execution time:",end-start)
```

Code snippet for multithreading

Each computer has different number of cores and threads available and therefore the multithreading performance will vary based on the system. However, the general hypothesis is that the multi-threaded program should usually have a lesser execution time as compared to the serial program.

5.5 FRONT-END OF APPLICATION

Python tkinter GUI is used to built the front-end of the application. A simple window with an input box for the review is designed along with buttons that let the user choose which function to perform and the results are displayed using messagebox widget of tkinter as a pop-up.

Fig 5.9 Front end GUI of application

While the three buttons perform the individual functions, the GO button performs all three together and provides an output

Chapter 6

Results

Table 6.1 shows the evaluation metrics of the four SVM models built for Suspicious Review Classifier. The SVM which was trained with additional features (SVM 1.3) such as Rating, Verified Purchase, Product Catergory etc. showed the best accuracy of about 82%.

Table 6.1 SVM models summary

SVM Version	Accuracy	Precision	Recall	Fscore
SVM 1.0	0.614	0.614	0.614	0.613
SVM 1.1	0.625	0.626	0.625	0.625
SVM 1.2	0.691	0.692	0.691	0.690
SVM 1.3	0.821	0.822	0.821	0.820

Figure 6.1 shows the graph of training accuracy vs epoch for the sentiment classifier model. It can be seen that the accuracy reaches about 96% at the 5th epoch and is almost saturated so the training is stopped.

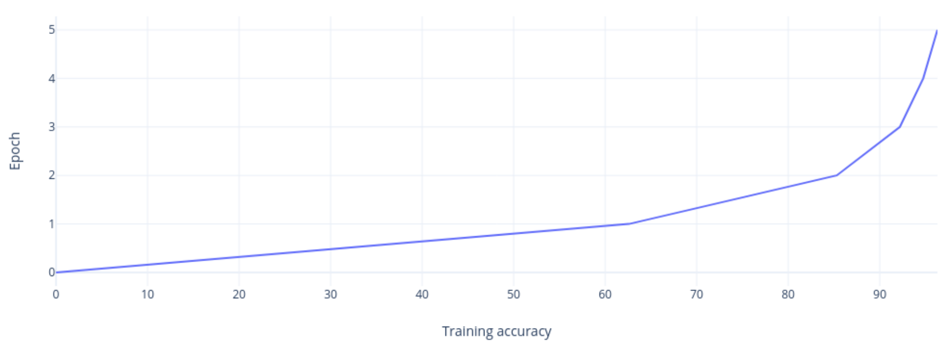


Fig 6.1 Training accuracy vs epoch

Figure 6.2 shows the graph of training loss and validation loss vs epoch for the abstractive text summarizer model. It can be seen that the validation loss increases consecutively after the 13th epoch and hence due to the early stopping mechanism, training stops at epoch 17.

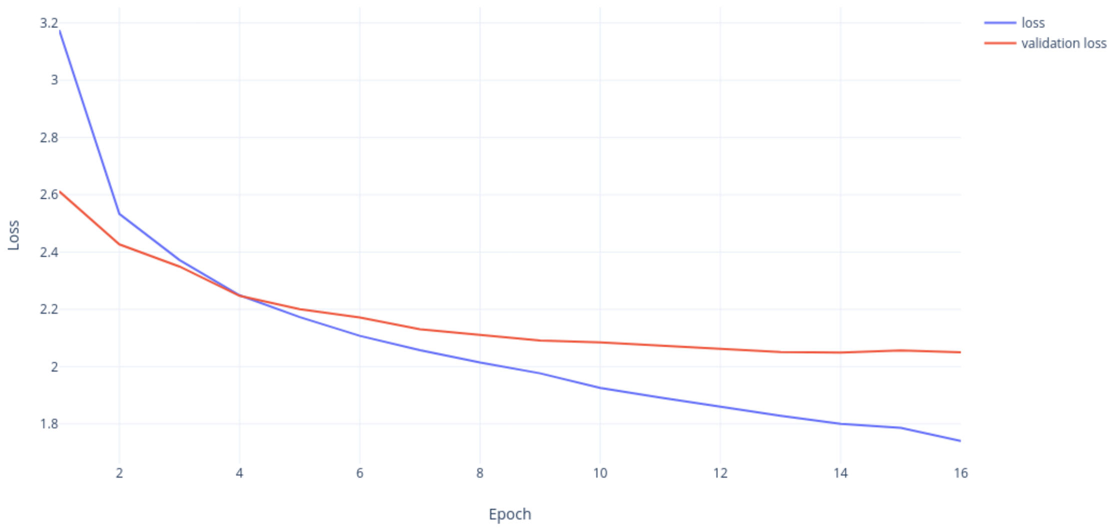


Fig 6.2 Training and validation loss vs epoch

Table 6.2 shows the average execution time for the standard and multithreaded programs for 10 executions. It is seen that the multithreaded program is marginally faster than the serial or standard version of the program.

Table 6.2 Average execution time

Method	Execution time
Standard	6.40 seconds
Multithreaded	5.77 seconds

Table 6.3 summarizes the 3 modules built and trained in this project along with their execution time and accuracy.

Table 6.3 Summary of all models built

Module	Model	Dataset	Epochs	Training Time	Accuracy
Suspicious Review Classifier	SVM	Amazon reviews	N/A	98 sec	80%
Review Summarizer	Encoder-Decoder w RNN- LSTM	Amazon fine food reviews	16 (early stopping)	230 min	N/A
Sentiment Classifier	Bidirectional LSTM-GRU	IMDB movie reviews	5	211 min	96.3%

Fig 6.3 Application Window Screenshot

- **Review :** always perfect snack dog loves knows exactly starts ask time evening gets greenie snack thank excellent product fast delivery.

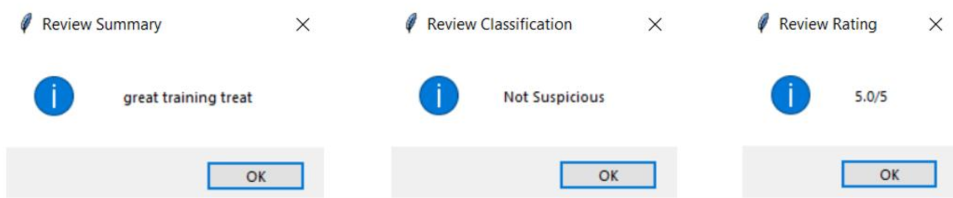


Fig 6.4 Output Screenshot 1

- **Review :** new price attractive however tastes horrible.

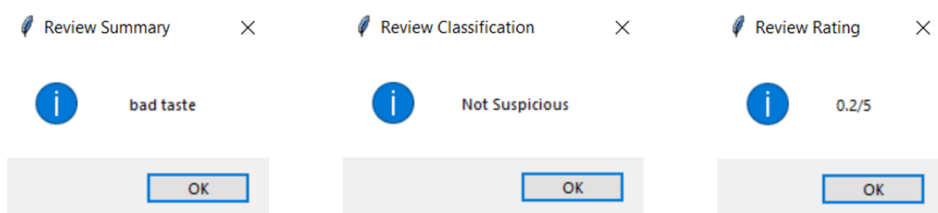


Fig 6.5 Output Screenshot 2

- **Review :** aware decaf coffee although showed search decaf cups intended purchase gift kept recipient drink caffeine.

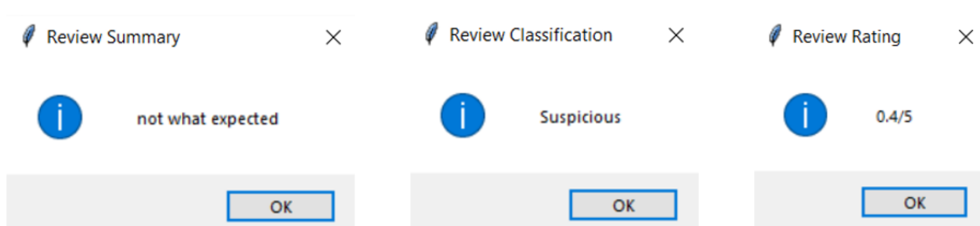


Fig 6.6 Output Screenshot 3

- **Review :** This is the best controller I have ever used. Thanks to Amazon India for providing this product in India.

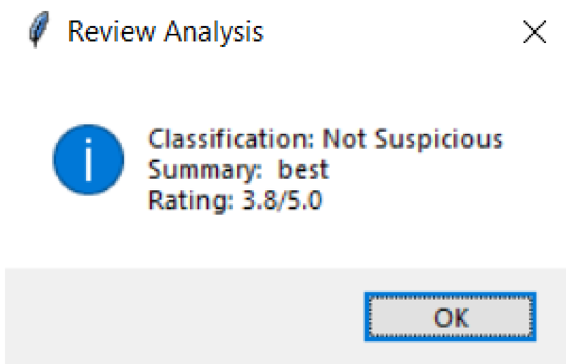


Fig 6.7 Output Screenshot 4

- **Review :** It's very bright and colorful and offers many different effect. All in all this light is good value for money compared to prices I have seen for similar lights in local stores.

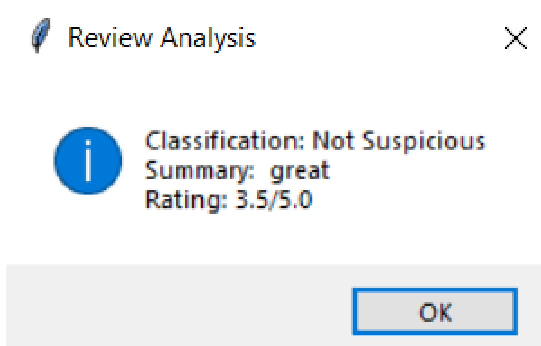


Fig 6.8 Output Screenshot 5

Chapter 7

Conclusion & Future Works

The hypothesis that multithreaded program must have a lesser execution time is proved right although by a very minimal margin. This could be because of various reasons such as lack of multithreading ability of the computer, size of the program being small to exhibit a significant difference or the code not being parallelizable to full extent.

The SVM model worked as expected and showed decent accuracy of 80% + for a classification that is vague even to human abilities. With a more appropriate dataset, the same model can be used to achieve better results in future.

Only a subset 100,000 reviews were used from the 500,000 Amazon fine food reviews dataset due to the time constraints and computing capabilities. For best results, all 500,000 reviews should be taken into training and testing the deep learning model as they show better accuracies with increase in amount of data.

The abstractive text summarizer designed using encoder decoder architecture showed promising results. Its performance can be improved by implementing a bidirectional LSTM to capture more information and context from both the directions, by increasing the training dataset size, by using beam search strategy for the decoding process and implementing various pointer-generator networks and coverage mechanisms. The performance of the model could further be evaluated by using the BLEU score.

The sentiment classifier shows an accuracy of 96% on 5 epochs. More epochs could be tried for better results along with increase in dataset size. More research could be done for better classification ranges of the sentiment score for rating prediction. A bidirectional LSTM-GRU model was used, however a more complex multi-channel neural network model (refer to Appendix 2) can be used for far better results given the computational resources required for processing and training such a complex model.

One thing that was noticed was the high training times of the deep learning LSTM-GRU models (200+ mins). This time could be reduced by using a more powerful machine to train and also by making use of the Graphical Processing Unit (GPU)'s ability to train models at a faster pace.

Further, the application which was built using python tkinter GUI, could be instead built on a flask server and using the TF2.0 serving API to serve the application directly as an API.

Appendices

Appendix 1

Neural Network Layers

Fig A.1 depicts an example of a CNN architecture and its key layers that are involved in its 2 main functions – Feature extraction and classification.

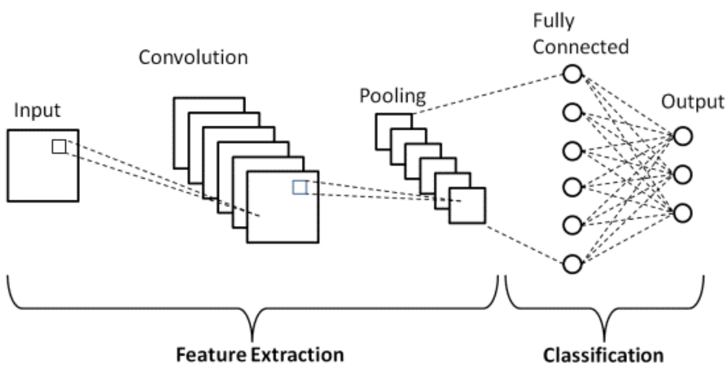


Fig A.1 CNN Architecture

Table A.1 shows the various convolutional neural network layers used in the project and their basic functions.

Table A.1 CNN Layers

Layer	Function
pooling	reduces the number of parameters to learn and the amount of computation performed in the network by reducing the dimensions of the feature maps.
dropout	helps prevent overfitting by randomly setting input units to 0 with a frequency of rate at each step during training time.
flatten	converts the pooled feature map to a single column that is passed to the fully connected layer.
convolutional	summarize the presence of features in an input image in a CNN
dense	adds the fully connected layer to the neural network.
concatenate	concatenates a list of inputs.

Appendix 2

Multi-Channel Neural Network

Multi-Channel Neural networks can be trained on multiple types of inputs and are hence very effective in problems such as emotion recognition or sentiment classification.

For this project, a 3 channel MCNN could be used with each channel consisting of a bidirectional LSTM- GRU layers combined with pooling and flatten layers. However, due to time and computational power constraints this model was dropped and a single channel network was used for the project.

```
def define_model(length, vocab_size):
    # channel 1
    inputs1 = Input(shape=(length,))
    embedding1 = Embedding(vocabulary_size, 300, weights=[embedding_matrix])(inputs1)
    conv1 = Conv1D(filters=16, kernel_size=4, activation='relu')(embedding1)
    drop1 = Dropout(0.5)(conv1)
    lstm1 = Bidirectional(LSTM(10, return_sequences = True))(drop1)
    gru1 = Bidirectional(GRU(10, return_sequences = True))(lstm1)
    pool1 = MaxPooling1D(pool_size=2)(gru1)
    flat1 = Flatten()(pool1)
    # channel 2
    inputs2 = Input(shape=(length,))
    embedding2 = Embedding(vocabulary_size, 300, weights=[embedding_matrix])(inputs2)
    conv2 = Conv1D(filters=16, kernel_size=6, activation='relu')(embedding2)
    drop2 = Dropout(0.5)(conv2)
    lstm2 = Bidirectional(LSTM(10, return_sequences = True))(drop2)
    gru2 = Bidirectional(LSTM(10, return_sequences = True))(lstm2)
    pool2 = MaxPooling1D(pool_size=2)(gru2)
    flat2 = Flatten()(pool2)
    # channel 3
    inputs3 = Input(shape=(length,))
    embedding3 = Embedding(vocabulary_size, 300, weights=[embedding_matrix])(inputs3)
    conv3 = Conv1D(filters=16, kernel_size=8, activation='relu')(embedding3)
    drop3 = Dropout(0.5)(conv3)
    lstm3 = Bidirectional(LSTM(10, return_sequences = True))(drop3)
    gru3 = Bidirectional(GRU(10, return_sequences = True))(lstm3)
    pool3 = MaxPooling1D(pool_size=2)(gru3)
    flat3 = Flatten()(pool3)
    merged = concatenate([flat1, flat2, flat3])
    # interpretation
    dense1 = Dense(10, activation='relu')(merged)
    outputs = Dense(1, activation='sigmoid')(dense1)
    modelx = Model(inputs=[inputs1, inputs2, inputs3], outputs=outputs)
    # compile
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    # summarize
    print(model.summary())
    #plot_model(model, show_shapes=True, to_file='multichannel.png')
    return model
```

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 800)]        0
__________________________________________________________________________________________________
input_2 (InputLayer)            [(None, 800)]        0
__________________________________________________________________________________________________
input_3 (InputLayer)            [(None, 800)]        0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 800, 300)     36567300    input_1[0][0]
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, 800, 300)     36567300    input_2[0][0]
__________________________________________________________________________________________________
embedding_2 (Embedding)         (None, 800, 300)     36567300    input_3[0][0]
__________________________________________________________________________________________________
conv1d (Conv1D)                 (None, 797, 16)      19216       embedding[0][0]
__________________________________________________________________________________________________
conv1d_1 (Conv1D)               (None, 795, 16)      28816       embedding_1[0][0]
__________________________________________________________________________________________________
conv1d_2 (Conv1D)               (None, 793, 16)      38416       embedding_2[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 797, 16)      0           conv1d[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 795, 16)      0           conv1d_1[0][0]
__________________________________________________________________________________________________
dropout_2 (Dropout)             (None, 793, 16)      0           conv1d_2[0][0]
__________________________________________________________________________________________________
bidirectional (Bidirectional)   (None, 797, 20)      2160        dropout[0][0]
__________________________________________________________________________________________________
bidirectional_2 (Bidirectional) (None, 795, 20)      2160        dropout_1[0][0]
__________________________________________________________________________________________________
bidirectional_4 (Bidirectional) (None, 793, 20)      2160        dropout_2[0][0]
__________________________________________________________________________________________________
bidirectional_1 (Bidirectional) (None, 797, 20)      1920        bidirectional[0][0]
__________________________________________________________________________________________________
bidirectional_3 (Bidirectional) (None, 795, 20)      2480        bidirectional_2[0][0]
__________________________________________________________________________________________________
bidirectional_5 (Bidirectional) (None, 793, 20)      1920        bidirectional_4[0][0]
__________________________________________________________________________________________________
max_pooling1d (MaxPooling1D)    (None, 398, 20)      0           bidirectional_1[0][0]
__________________________________________________________________________________________________
max_pooling1d_1 (MaxPooling1D)  (None, 397, 20)      0           bidirectional_3[0][0]
__________________________________________________________________________________________________
max_pooling1d_2 (MaxPooling1D)  (None, 396, 20)      0           bidirectional_5[0][0]
__________________________________________________________________________________________________
flatten (Flatten)               (None, 7960)         0           max_pooling1d[0][0]
__________________________________________________________________________________________________
flatten_1 (Flatten)             (None, 7940)         0           max_pooling1d_1[0][0]
__________________________________________________________________________________________________
flatten_2 (Flatten)             (None, 7920)         0           max_pooling1d_2[0][0]
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 23820)        0           flatten[0][0]
                                                                 flatten_1[0][0]
                                                                 flatten_2[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 10)           238210      concatenate[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 1)            11          dense[0][0]
==================================================================================================
Total params: 110,039,369
Trainable params: 110,039,369
Non-trainable params: 0
__________________________________________________________________________________________________
None
```

REFERENCES

1. Feng, S., Xing, L., Gogar, A., & Choi, Y. (2012). Distributional Footprints of Deceptive Product Reviews. Proceedings of the International AAAI Conference on Web and Social Media, 6(1).

2. Choi, S. A. (2020). Amazon Fake Reviews. arXiv preprint arXiv:2009.09102.

3. Lu, Y., Zhang, L., Xiao, Y., & Li, Y. (2013, May). Simultaneously detecting fake reviews and review spammers using factor graph model. In Proceedings of the 5th annual ACM web science conference (pp. 225-233).

4. Fornaciari, T., & Poesio, M. (2014). Identifying fake Amazon reviews as learning from crowds.

5. Raykar, V. C., Yu, S., Zhao, L. H., Valadez, G. H., Florin, C., Bogoni, L., & Moy, L. (2010). Learning from crowds. Journal of Machine Learning Research, 11(4).

6. Basiri, M. E., Ghasem-Aghaee, N., & Naghsh-Nilchi, A. R. (2014). Exploiting reviewers' comment histories for sentiment analysis. Journal of Information Science, 40(3), 313-328.

7. Ruales, J. (2011). Recurrent neural networks for sentiment analysis. IEEE. Colombia: Colombia University.

8. Sachin, S., Tripathi, A., Mahajan, N., Aggarwal, S., & Nagrath, P. (2020). Sentiment analysis using gated recurrent neural networks. SN Computer Science, 1(2), 1-13.

9. Trofimovich, J. (2016). Comparison of neural network architectures for sentiment analysis of russian tweets. In Computational Linguistics and Intellectual Technologies: Proceedings of the International Conference Dialogue (pp. 50-59).

10. Salur, M. U., & Aydin, I. (2020). A novel hybrid deep learning model for sentiment classification. *IEEE Access, 8*, 58080-58093.

11. Luo, L. X. (2019). Network text sentiment analysis method combining LDA text representation and GRU-CNN. Personal and Ubiquitous Computing, 23(3), 405-412.

12. Jabreel, M., Hassan, F., & Moreno, A. (2018). Target-dependent sentiment analysis of tweets using bidirectional gated recurrent neural networks. In Advances in hybridization of intelligent methods (pp. 39-55). Springer, Cham.

13. Shen, T., Zhou, T., Long, G., Jiang, J., & Zhang, C. (2018). Bi-directional block self-attention for fast and memory-efficient sequence modeling. arXiv preprint arXiv:1804.00857.

14. Majumder, N., Hazarika, D., Gelbukh, A., Cambria, E., & Poria, S. (2018). Multimodal sentiment analysis using hierarchical fusion with context modeling. Knowledge-based systems, 161, 124-133.

15. Piao, G., & Breslin, J. G. (2018, April). Financial aspect and sentiment predictions with deep neural networks: an ensemble approach. In Companion Proceedings of the The Web Conference 2018 (pp. 1973-1977).

16. Wang, Y., Sun, A., Han, J., Liu, Y., & Zhu, X. (2018, April). Sentiment analysis by capsules. In Proceedings of the 2018 world wide web conference (pp. 1165-1174).

17. Penghua, Z., & Dingyi, Z. (2019, February). Bidirectional-GRU based on attention mechanism for aspect-level Sentiment Analysis. In Proceedings of the 2019 11th International Conference on Machine Learning and Computing (pp. 86-90).

18. Bjerva, J., Plank, B., & Bos, J. (2016). Semantic tagging with deep residual networks. arXiv preprint arXiv:1609.07053.

19. Zhou, J., Huang, J. X., Chen, Q., Hu, Q. V., Wang, T., & He, L. (2019). Deep learning for aspect-level sentiment classification: Survey, vision, and challenges. IEEE access, 7, 78454-78483.

20. Nallapati, R., Zhou, B., Gulcehre, C., & Xiang, B. (2016). Abstractive text summarization using sequence-to-sequence rnns and beyond. arXiv preprint arXiv:1602.06023.

21. Minh, D. L., Sadeghi-Niaraki, A., Huy, H. D., Min, K., & Moon, H. (2018). Deep learning approach for short-term stock trends prediction based on two-stream gated recurrent unit network. Ieee Access, 6, 55392-55404.

22. Wang, N., Wang, J., & Zhang, X. (2017, December). YNU-HPCC at IJCNLP-2017 Task 4: attention-based Bi-directional GRU model for customer feedback analysis task of English. In Proceedings of the IJCNLP 2017, Shared Tasks (pp. 174-179).

23. Zhang, L., Zhou, Y., Duan, X., & Chen, R. (2018, March). A hierarchical multi-input and output bi-GRU model for sentiment analysis on customer reviews. In IOP conference series: materials science and engineering (Vol. 322, No. 6, p. 062007). IOP Publishing.

24. Huang, Y., Jiang, Y., Hasan, T., Jiang, Q., & Li, C. (2018, March). A topic BiLSTM model for sentiment classification. In Proceedings of the 2nd International Conference on Innovation in Artificial Intelligence (pp. 143-147).

25. Wu, J., Zheng, K., & Sun, J. (2019, June). Text sentiment classification based on layered attention network. In Proceedings of the 2019 3rd High Performance Computing and Cluster Technologies Conference (pp. 162-166).

26. Moratanch, N., & Chitrakala, S. (2016, March). A survey on abstractive text summarization. In 2016 International Conference on Circuit, power and computing technologies (ICCPCT) (pp. 1-7). IEEE.

27. Liu, L., Lu, Y., Yang, M., Qu, Q., Zhu, J., & Li, H. (2018, April). Generative adversarial network for abstractive text summarization. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 32, No. 1).

28. Song, S., Huang, H., & Ruan, T. (2019). Abstractive text summarization using LSTM-CNN based deep learning. Multimedia Tools and Applications, 78(1), 857-875.

29. Shi, T., Keneshloo, Y., Ramakrishnan, N., & Reddy, C. K. (2021). Neural abstractive text summarization with sequence-to-sequence models. ACM Transactions on Data Science, 2(1), 1-37.

30. Maas, A., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., & Potts, C. (2011, June). Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies* (pp. 142-150).

31. Pennington, J., Socher, R., & Manning, C. D. (2014, October). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 1532-1543).