TEXT MINING AND SENTIMENT ANALYSIS

FINAL Project Report submitted to IcfaiTech (Deemed to be University) as a partial fulfillment of the requirements for the award of the Degree of B.Tech in 7th semester

B. SOWMYA : 18STUCHH010005

Department of Computer Science

IcfaiTech (Deemed to be University)

HYDERABAD DECEMBER, 2021

DECL ARATION

I declare that the work contained in the Project Report is original and it has been done by me under the supervision of Prof.Digvijay Nair . The work has not been submitted to any other University for the award of any degree or diploma.

Date: 11/12/2021

Signature of Student: BURUJU SOWMYA

ACKNOWLEDGMENT

We would like to express our special thanks to all our teammates, whose guidance and encouragement made this possible. This internship opportunity gave us a great chance in learning several things and it helped us in developing my skills. Therefore, we're very lucky to be a part of it.

We would like to express my thankfulness to our director **Mr. M. Srinivasa Reddy**, faculty of Science and technology, IFHE Hyderabad, for giving us an opportunity to excel in our course.

I am very much thankful to our SIP coordinator **Prof. Digvijay Nair** for her constant supervision and co-operation throughout the program.

This Acknowledgement transcends the reality of formality where we would like to express deep gratitude and respect to all those people behind the screen who guided, inspired and helped us throughout the project.

We perceive this opportunity as a big milestone in our career development. We will strive to use gained skills and knowledge in the best possible way and will continue to work on their improvement to attain desired career objectives.

ABSTRACT

The entire world is transforming quickly under the present innovations. The Internet has become a basic requirement for everybody with the Web being utilized in every field. With the rapid increase in social network applications, people are using these platforms to voice them their opinions with regard to daily issues. Gathering and analyzing peoples' reactions toward buying a product, public services, and so on are vital. Sentiment analysis (or opinion mining) is a common dialogue preparing task that aims to discover the sentiments behind opinions in texts on varying subjects. In recent years, researchers in the field of sentiment analysis have been concerned with analyzing opinions on different topics such as movies, commercial products, and daily societal issues. Twitter is an enormously popular microblog on which clients may voice their opinions. Opinion investigation of Twitter data is a field that has been given much attention over the last decade and involves dissecting "tweets" (comments) and the content of these expressions. As such, this paper explores the various sentiment analysis applied to Twitter data and their outcomes.

TABLE OF CONTENTS

1. Introduction…….....12

1.1 Natural language Processing……...……..…12

1.1.1 History of NLP……..14

1.1.2 NLP Phases……16

1.1.2.1 Morpholofical Processing…….16

1.1.2.2 Syntax Analysis……..17

1.1.2.3 Semantic Analysis……..17

1.1.2.4 Pragmatic Analysis……17

1.1.3 NLP Tasks……..17

1.1.4 NLP Tools and Approach…….18

1.1.5 NLP use cases…….19

2.Main Text……21

2.1 Text Preprocessing…….21

2.1.1 Import Libraries…….23

2.1.2 Reading Data……...23

2.1.3 Expand Contractions…….…….23

2.1.4 Lower case……..….24

2.1.5 Remove Punctuations……..…25

2.1.6 Remove words and digits containing digits……..…26

2.1.7 Remove Stopwords……..…26

2.1.8 Stemming……..…27

2.1.9 Tokenize text……...28

2.1.10 Remove Whitespaces……..…….29

2.1.11 Lemmatization……..29

2.2 Exploratory Data Analysis……30

2.2.1 Sentence count……...31

2.2.2 Word count……..…31

2.2.3 Character count…….32

2.2.4 Sentence density…….32

2.2.5 Word density……..32

2.2.6 Punctuation count……..33

2.2.7 Stopwords count…….33

2.2.8 Word frequency analysis……..…..34

2.2.9 Sentence length analysis……35

2.2.10 Average word length analysis……45

2.2.11 BoxPlot……...…36

2.2.12 Scatterplot……..…37

2.2.13 Histogram……..37

2.2.14 Interquartile range(IQR)……38

2.2.15 Plotting dataset for data type integers……38

2.2.16 Describing the time……..…40

2.2.17 Correlation plot……41

2.2.18 Polarity……..41

2.2.19 Stopword count in the text data……..42

2.2.20 Ngrams……..43

2.3 Multi Class Text Classification Using LSTM…….43

2.3.1 Multi Class Classification……43

2.3.2 Recurrent Neural Network……44

2.3.3 Long short term memory(LSTM)……..45

2.3.4 Bidirectional LSTM……46

2.3.5 Data Description……..…47

2.3.6 Coding……...48

2.3.6.1 Importing Libraries…….48

2.3.6.2 Load Data…….……49

2.3.6.3 Data Preprocessing……..49

2.3.6.4 Splitting dataset into training and test sets.....52

2.3.6.5 Defining Hyperparameters……..…52

2.3.6.6 Tokenization……..……...53

2.3.6.7 Padding……..……...54

2.3.6.8 Implementation of Long Short Term Memory (LSTM)……..…….54

2.3.7 Future Work……..…….56

2.4 Multi Label Text Classification with BERT and Pytorch Lightning……57

2.4.1 Why BERT Needed…….……57

2.4.2 How does BERT Work……..……58

2.4.3 Architecture…….……59

2.4.4 Coding…….……59

2.4.5.1 Creating Environment……..….59

2.4.4.2 Importing Libraries……..…….59

2.4.4.3 Load Data……..……60

2.4.4.4 Splitting the data……..……..60

2.4.4.5 Preprocessing……..……61

2.4.4.6 Tokenization……...62

2.4.4.7 Dataset…….……65

2.4.4.8 Model…….…...68

2.4.4.9 Optimizer Scheduler…….…...70

2.4.4.10 Evaluation……...……71

2.4.4.11 ROC Curve…….73

2.5 Ngrams……..74

2.5.1 Importing the libraries……....74

2.5.2 Cleaning the data……75

2.5.3 Bigram……...75

2.5.4 Trigram……75

2.5.5 Ploting most frequent words in bigram.……..76

2.6 Word Embedding……76

2.6.1 Word embedding algorithms……...77

2.6.1.1 Embedding layer……77

2.6.1.2 Word2Vec……77

2.6.2 Reading the data…….....79

2.6.3 Cleaning the data……....79

2.6.4 Tokenizing the sequence and invoking the word2vec…....80

2.6.5 Returing the list of words…….…..80

2.6.6 Checking the vector representation…….81

2.6.7 Checking the list of similar words…….81

2.7 Cosine Similarity…….82

2.7.1 Calculating the cosine similarity between two text data…83

2.7.1.1 Reading the data……83

2.7.1.2 Clear visualization of vectorize data along with tokens……83

2.7.1.3 Cosine Similarity of two text data……84

3.Conclusion……..85

4. Appendices……...86

4.1 Content Extraction From PDF……...86

4.1.1 PyPDF2……..86

4.1.2 Tika……90

4.1.3 Texttract……91

4.1.4 PyMuPDF…….91

4.1.5 PDFtotext……..91

4.1.6 PDFminer……..91

4.1.7 Tabula……92

5.References……..93

1.INTRODUCTION

1.1 NATURAL LANGUAGE PROCESSING

Language is a method of verbal exchange with the help of which we will communicate, read and write. For instance, we think, we make decisions, plans and more in natural language; exactly, in phrases. However, the huge query that confronts us in this AI generation is whether we can speak in a similar way with computer systems. In different words, can people talk with computer systems in their herbal language? It is an assignment for us to expand NLP applications due to the fact that computer systems want established statistics, but human speech is unstructured and regularly ambiguous in nature.

In this sense, we can say that Natural Language Processing (NLP) is the sub-field of Computer Science specially Artificial Intelligence (AI) that is concerned about permitting computers to recognize and technique human language. Technically, the principal assignment of NLP would be to apply computers for reading and processing big amounts of herbal language data.

The essence of Natural Language Processing lies in making computers apprehend the natural language. That’s no longer a clean undertaking though. Computers can understand the dependent form of facts like spreadsheets and the tables in the database, but human languages, texts, and voices form an unstructured class of facts, and it is tough for the computer to understand it, and there arises the need for Natural Language Processing.

There’s a lot of natural language facts obtainable in numerous bureaucracies and it'd get very smooth if computers could recognize and system that information. We can educate the models according to anticipated output in exceptional approaches. Humans have been writing for heaps of years, there are loads of literature portions available, and it might be remarkable if we make computers take that into account. But the assignment is never going to be easy. There are diverse challenges floating accessible like expertise, the best meaning of the sentence, accurate Named-Entity Recognition(NER), accurate prediction of various components of speech, coreference resolution(the most hard factor for my part).

Computers can’t truly recognize the human language. If we feed enough statistics and train a version well, it can distinguish and attempt categorizing diverse parts of speech(noun, verb, adjective, supporter, etc…) based on formerly fed data and stories. If it encounters a brand new word it attempts to make the closest bet which may be embarrassingly wrong in a few instances.

It’s very difficult for a pc to extract the exact meaning from a sentence. For instance – The boy radiated hearth like vibes. The boy had a completely motivating persona or he really radiated hearth? As you spot over right here, parsing English with a pc is going to be complex.

There are diverse levels of worry in education. Solving a complex hassle in Machine Learning by building a pipeline. In simple phrases, it means breaking a complex problem into a number of small problems, making fashions for every of them after which integrating those fashions. A similar component is completed in NLP. We can damage the process of understanding English for a model into a number of small pieces.

It could be truly first rate if a laptop could remember the fact that San Pedro is an island in Belize district in Central America with a population of sixteen, 444 and it's far the second largest city in Belize. But to make the PC understand this, we want to educate laptops on the very primary concepts of written language.

1.1.1 HISTORY OF NLP

We have divided the history of NLP into four phases. The phases have distinctive concerns and styles.

First Phase (Machine Translation Phase) - Late 1940s to late 1960s

The work done in this phase focused mainly on machine translation (MT). This phase was a period of enthusiasm and optimism.

Let us now see all that the first segment had in it −

- The research on NLP commenced in the early Fifties after Booth & Richens' research and Weaver's memorandum on machine translation in 1949.
- 1954 turned into the year while a constrained experiment on automated translation from Russian to English validated within the Georgetown-IBM experiment.
- In the same year, the book of the magazine MT (Machine Translation) started.
- The first international conference on Machine Translation (MT) was held in 1952 and 2nd was held in 1956.
- In 1961, the work offered in the Teddington International Conference on Machine Translation of Languages and Applied Language Evaluation was the high point of this phase.

Second Phase (AI Influenced Phase) – Late 1960s to late Nineteen Seventies

In this section, the paintings done were majorly associated with world knowledge and on its role within the construction and manipulation of that means representations. That is why, this phase is likewise called the AI-flavored section.

The section had in it, the following −

- In early 1961, the paintings commenced at the issues of addressing and constructing records or expertise. This work was stimulated through AI.
- In the identical year, a BASEBALL query-answering device also evolved. The input to this gadget turned into constrained and the language processing concerned changed into a simple one.
- A tons superior device turned into defined in Minsky (1968). This machine, when in comparison to the BASEBALL query-answering system, became recognized and furnished for the want of inference at the information base in deciphering and responding to language enter.

Third Phase (Grammatico-logical Phase) – Late Nineteen Seventies to late Nineteen Eighties

This section may be described because of the grammatico-logical segment. Due to the failure of sensible machine building in the remaining section, the researchers moved in the direction of the use of logic for know-how representation and reasoning in AI.

The 0.33 segment had the following in it −

- The grammatico-logical approach, toward the end of decade, helped us with powerful standard-reason sentence processors like SRI's Core Language Engine and Discourse Representation Theory, which offered a way of tackling extra extended discourse.
- In this section we were given a few realistic assets & gear like parsers, e.G. Alvey Natural Language Tools in conjunction with more operational and industrial systems, e.G. For database questions.
- The paintings on lexicon in Nineteen Eighties also pointed within the course of grammatico-logical method.

Fourth Phase (Lexical & Corpus Phase) – The Nineteen Nineties

We can describe this as a lexical & corpus section. The phrase had a lexicalized technique to grammar that was regarded in the past due Nineteen Eighties and has become an increasing effect. There has been a revolution in herbal language processing in this decade with the creation of gadgets getting to know algorithms for language processing.

1.1.2 NLP Phases

Following diagram shows the phases or logical steps in natural language processing −

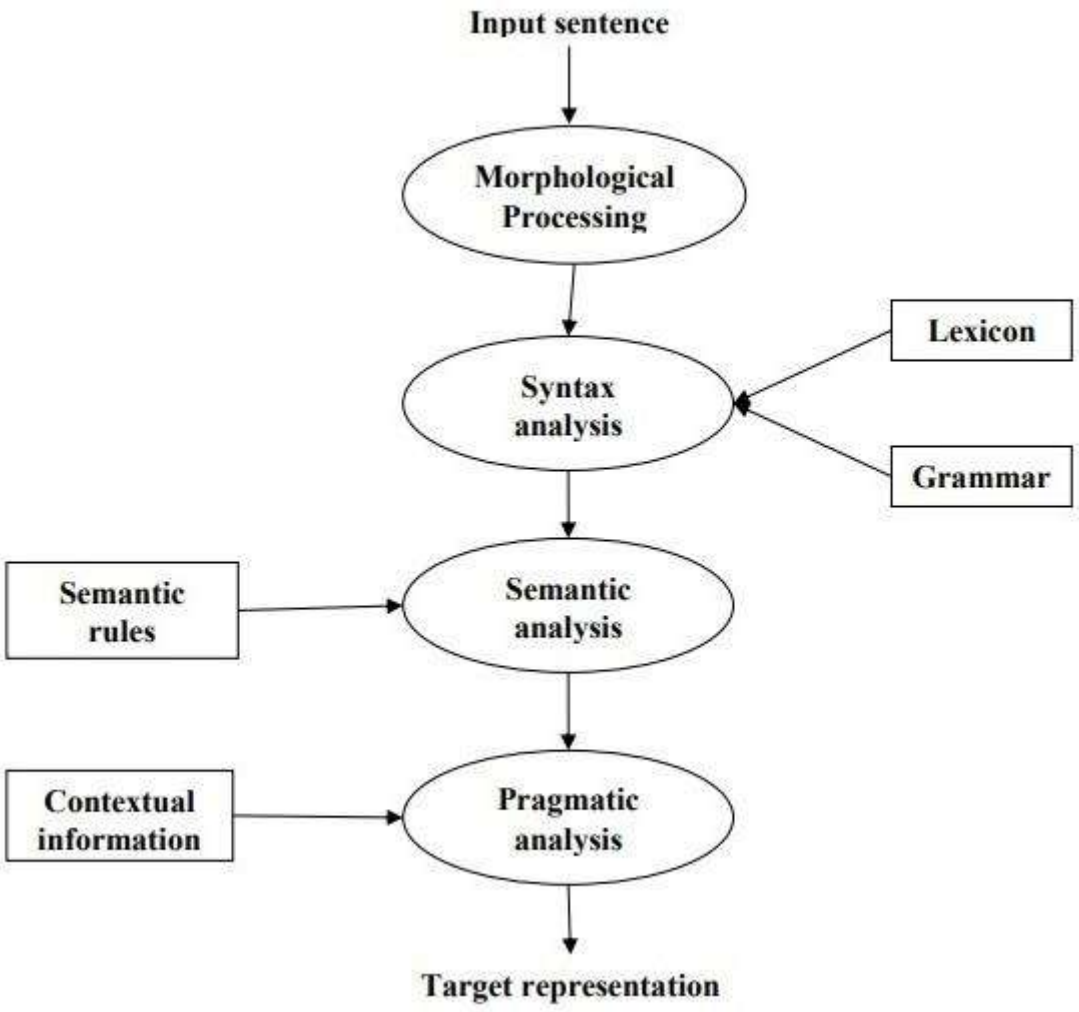

Figure 1 NLP Phases.

1.1.2.1 Morphological Processing

It is the first segment of NLP. The motive of this segment is to interrupt chunks of language input into sets of tokens similar to paragraphs, sentences and phrases. For instance, a phrase like “uneasy” can be damaged into sub-word tokens as “unclean”.

1.1.2.2 Syntax Analysis

It is the second segment of NLP. The reason for this segment is two fold: to check that a sentence is properly fashioned or not and to interrupt it up right into a shape that indicates the syntactic relationships among the unique words. For instance, sentences like “The school goes to the boy” might be rejected via syntax analyzer or parser.

1.1.2.3 Semantic Analysis

It is the 1/3 segment of NLP. The motive of this section is to draw actual meaning, or you could say dictionary which means from the text. The text is checked for meaningfulness. For instance, a semantic analyzer could reject a sentence like “Hot ice-cream”.

1.1.2.4 Pragmatic Analysis

It is the fourth phase of NLP. Pragmatic analysis clearly suits the real gadgets/activities, which exist in a given context with object references acquired over the past section (semantic evaluation). For instance, the sentence “Put the banana within the basket on the shelf” may have two semantic interpretations and a pragmatic analyzer will pick among these possibilities.

1.1.3 NLP TASKS

Human language is packed with ambiguities that make it especially difficult to write software programs that appropriately determine the intended means of textual content or voice records. Homonyms, homophones, sarcasm, idioms, metaphors, grammar and usage exceptions, variations in sentence shape—these only some of the irregularities of human language that take people years to examine, but that programmers need to train natural language-pushed applications to apprehend and apprehend accurately from the begin, if the ones applications are going to be useful.

Several NLP obligations damage down human textual content and voice facts in approaches that help the laptop make experience of what it is consuming. Some of those duties consist of the following:

- Speech recognition, also referred to as speech-to-text, is the project of reliably converting voice data into text information. Speech reputation is required for any application that follows voice instructions or answers spoken questions. What makes speech reputation mainly hard is the way people communicate—fast, slurring words together, with various emphasis and intonation, in distinctive accents, and regularly the use of incorrect grammar.
- Part of speech tagging, additionally referred to as grammatical tagging, is the procedure of figuring out a part of speech of a specific word or piece of text primarily based on its use and context. Part of speech identifies ‘make’ as a verb in ‘I could make a paper plane,’ and as a noun in ‘What make of automobile do you very own?’
- Word sense disambiguation is the selection of the means of a phrase with more than one meaning via a method of semantic analysis that determines the phrase that makes the maximum feel within the given context. For example, phrase sense disambiguation distinguishes the meaning of the verb 'make' in ‘make the grade’ (attain) vs. ‘make a wager’ (place).
- Named entity recognition, or NEM, identifies words or phrases as useful entities. NEM identifies ‘Kentucky’ as a location or ‘Fred’ as a person's call.
- Co-reference decision is the mission of identifying if and when two words talk to the same entity. The maximum commonplace example is figuring out the character or item to which a positive pronoun refers (e.G., ‘she’ = ‘Mary’), but it can also contain identifying a metaphor or an idiom within the textual content (e.G., an example in which 'bear' isn't an animal however a large bushy person).
- Sentiment evaluation attempts to extract subjective characteristics—attitudes, emotions, sarcasm, confusion, suspicion—from textual content.
- Natural language technology is from time to time described as the opposite of speech popularity or speech-to-text; it's the project of setting structured facts into human language.

1.1.4 NLP Tools and Approach

Python and the Natural Language Toolkit (NLTK)

The Python programming language provides a huge variety of tools and libraries for attacking specific NLP duties. Many of these are observed within the Natural Language Toolkit, or NLTK, an open source collection of libraries, packages, and schooling sources for constructing NLP applications.

The NLTK includes libraries for many of the NLP tasks listed above, plus libraries for subtasks, along with sentence parsing, word segmentation, stemming and lemmatization (strategies of trimming phrases down to their roots), and tokenization (for breaking phrases, sentences, paragraphs and passages into tokens that help the pc better apprehend the text). It also includes libraries for enforcing competencies consisting of semantic reasoning, the capacity to attain logical conclusions based on records extracted from textual content.

Statistical NLP, system mastering, and deep mastering

The earliest NLP applications have been hand-coded, regulations-based total systems that would perform certain NLP duties, but couldn't without difficulty scale to accommodate a reputedly infinite circulation of exceptions or the increasing volumes of text and voice records.

Enter statistical NLP, which combines computer algorithms with gadget learning and deep gaining knowledge of models to automatically extract, classify, and label factors of text and voice facts and then assign a statistical probability to each viable meaning of these factors. Today, deep getting to know fashions and learning techniques based totally on convolutional neural networks (CNNs) and recurrent neural networks (RNNs) enable NLP structures that 'analyze' as they work and extract ever more accurate which means from big volumes of raw, unstructured, and unlabeled text and voice statistics units.

1.1.5 NLP use cases

Natural language processing is the driving force behind machine intelligence in lots of contemporary actual-world applications. Here are some examples:

- Spam detection: You may not think about unsolicited mail detection as an NLP answer, but the first-rate spam detection technologies use NLP's textual content category skills to test emails for language that frequently indicates spam or phishing. These indicators can include overuse of monetary phrases, function awful grammar, threatening language, irrelevant urgency, misspelled organization names, and more. Spam detection is one in every of a handful of NLP troubles that professionals remember 'by and large solved' (despite the fact that you may argue that this doesn't shape your email enjoyment).
- Machine translation: Google Translate is an example of broadly available NLP technology at paintings. Truly beneficial gadget translation entails greater than replacing words in a single language with words of every other. Effective translation has to capture as it should be the meaning and tone of the input language and translate it to text with the identical meaning and preferred impact inside the output language. Machine translation equipment is making proper development in phrases of accuracy. An amazing way to test any device translation tool is to translate text to one language after which back to the original. An oft-stated conventional instance: Not lengthy ago, translating “The spirit is inclined however the flesh is susceptible” from English to Russian and back yielded “The vodka is good but the meat is rotten.” Today, the result is “The spirit desires, however the flesh is weak,” which isn’t perfect, however inspires a good deal of extra confidence in the English-to-Russian translation.
- Virtual dealers and chatbots: Virtual dealers together with Apple's Siri and Amazon's Alexa use speech popularity to recognize patterns in voice instructions and herbal language technology to respond with suitable action or beneficial comments. Chatbots perform the same magic in response to typed text entries. The best of these additionally learn how to understand contextual clues approximately human requests and use them to offer even better responses or options over the years. The subsequent enhancement for those programs is query answering, the capacity to reply to our questions—anticipated or not—with applicable and useful solutions in their own phrases.

- Social media sentiment evaluation: NLP has turned out to be an crucial enterprise tool for uncovering hidden statistics insights from social media channels. Sentiment analysis can analyze language used in social media posts, responses, reviews, and greater to extract attitudes and emotions in reaction to products, promotions, and occasions–data groups can be used in product designs, marketing campaigns, and extra.
- Text summarization: Text summarization makes use of NLP strategies to digest large volumes of virtual textual content and create summaries and synopses for indexes, research databases, or busy readers who don't have time to study full text. The pleasant textual content summarization programs use semantic reasoning and natural language generation (NLG) to feature useful context and conclusions to summaries.

2. MAIN TEXT

2.1 TEXT PREPROCESSING

Text preprocessing is a method to clean the text data and make it ready to feed data to the model. Text data contains noise in various forms like emotions, punctuation, text in a different case etc.

Text preprocessing is traditionally an essential step for natural language processing (NLP) obligations. It transforms text right into a greater digestible shape in order that machine mastering algorithms can carry out better.

Data preprocessing is an important step to put together the information to shape a device learning model. There are many crucial steps in statistics preprocessing, which includes statistics cleaning, records transformation, and characteristic selection. Data cleaning and transformation are methods used to dispose of outliers and standardize the information so that they take a shape that may be without difficulty used to create a model.

In NLP, text preprocessing is the first step in the process of building a model.

Text Preprocessing Techniques:

- Expand Contractions

- Lower Case
- Remove Punctuations
- Remove words and digits containing digits
- Remove Stopwords
- Rephrase Text
- Stemming
- Lemmatization
- Normalization
- Remove White spaces

Natural Language ToolKit (NLTK)

NLTK is a main platform for building Python applications to paintings with human language statistics. It presents easy-to-use interfaces to over 50 corpora and lexical sources consisting of WordNet, together with a set of text processing libraries for category, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-energy NLP libraries, and an active dialogue discussion board.

Thanks to a palm-on guide introducing programming basics alongside topics in computational linguistics, plus comprehensive API documentation, NLTK is appropriate for linguists, engineers, students, educators, researchers, and industry customers alike. NLTK is available for Windows, Mac OS X, and Linux. Best of all, NLTK is a free, open supply, network-pushed mission.

NLTK has been called “an excellent tool for teaching, and working in, computational linguistics using Python,” and “an incredible library to play with natural language.”

Natural Language Processing with Python presents a sensible introduction to programming for language processing. Written by the creators of NLTK, it publishes the reader through the basics of writing Python programs, operating with corpora, categorizing text, reading linguistic shape, and extra.

2.1.1 Import Libraries

```
[2]: import pandas as pd
     import numpy as np
     import nltk
     import re
     import string
     from nltk.tokenize import word_tokenize
     from nltk.corpus import stopwords, wordnet
     from nltk.stem import WordNetLemmatizer
```

Figure 2.1 Importing libraries.

2.1.2 .Reading Data

```
[3]: data = pd.read_csv('Data/tain.csv', engine='python', encoding='utf-8', error_bad_lines=False)

/home/bs0c98343/.conda/envs/text_preprocessing/lib/python3.8/site-packages/IPython/core/interactiveshell.py:3417: FutureWarning: The error_bad_li
emoved in a future version.


  exec(code_obj, self.user_global_ns, self.user_ns)
Skipping line 24356: unexpected end of data

[4]: data.head()
```

[4]:

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	0000997932d777bf	Explanation\nWhy the edits made under my usern...	0	0	0	0	0	0
1	000103f0d9cfb60f	D'aww! He matches this background colour I'm s...	0	0	0	0	0	0
2	000113f07ec002fd	Hey man, I'm really not trying to edit war. It...	0	0	0	0	0	0
3	0001b41b1c6bb37e	"\nMore\nI can't make any real suggestions on ...	0	0	0	0	0	0
4	0001d958c54c6e35	You, sir, are my hero. Any chance you remember...	0	0	0	0	0	0

Figure 2.2 Reading Data

2.1.3 .Expand Contractions

Contractions are shortened versions of phrases or syllables. They are created by casting off specific, one or extra letters from words. Often extra than phrases are combined to create a contraction. In writing, an apostrophe is used to signify the area of lacking letters. In English language/textual content, contractions often exist in either written or spoken bureaucracy.

Nowadays, many editors will induce contractions by way of default. For example do not to don't, I could to I'd, you are to you're. Converting each contraction to its improved, authentic form helps with textual content standardization.

```
[9]: import contractions
     data['comment_text'] = data['comment_text'].apply(lambda x: [contractions.fix(word) for word in x.split()])
     data.head()
```

[9]:

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	0000997932d777bf	[explanation, why, the, edits, made, under, my...	0	0	0	0	0	0
1	000103f0d9cfb60f	[daww, he, matches, this, background, colour, ...	0	0	0	0	0	0
2	000113f07ec002fd	[hey, man, I am, really, not, trying, to, edit...	0	0	0	0	0	0
3	0001b41b1c6bb37e	[more, i, cannot, make, any, real, suggestions...	0	0	0	0	0	0
4	0001d958c54c6e35	[you, sir, are, my, hero, any, chance, you, re...	0	0	0	0	0	0

```
[10]: data['comment_text'] = [' '.join(map(str, l)) for l in data['comment_text']]
      data.head()
```

[10]:

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	0000997932d777bf	explanation why the edits made under my userna...	0	0	0	0	0	0
1	000103f0d9cfb60f	daww he matches this background colour I am se...	0	0	0	0	0	0
2	000113f07ec002fd	hey man I am really not trying to edit war its...	0	0	0	0	0	0
3	0001b41b1c6bb37e	more i cannot make any real suggestions on imp...	0	0	0	0	0	0
4	0001d958c54c6e35	you sir are my hero any chance you remember wh...	0	0	0	0	0	0

Figure 2.3 Expand Contractions

2.1.4 Lower case

Lowercasing ALL your textual content records, even though typically unnoticed, is one of the only and only forms of textual content preprocessing. It is relevant to maximum textual content mining and NLP troubles and may assist in instances wherein your dataset is not very large and appreciably allowed with consistency of expected output.

Quite recently, one of my weblog readers educated me on a word embedding model for similarity lookups. He discovered that exclusive variant in enter capitalization (e.G. 'Canada' vs. 'canada') gave him exceptional kinds of output or no output in any respect. This possibly

occurred because the dataset had combined-case occurrences of the word ‘Canada’ and there was inadequate evidence for the neural-network to successfully study the weights for the much less not unusual model. This sort of difficulty is bound to take place while your dataset is reasonably small, and lowercasing is an exquisite way to address sparsity problems.

```
[5]: data["comment_text"][0]
```

```
[5]: "Explanation\nWhy the edits made under my username Hardcore Metallica Fan were reverted? They weren't vandalisms, just closure on some GAs after I vote
n't remove the template from the talk page since I'm retired now.89.205.38.27"
```

```
#lower case
```

```
[6]: data['comment_text'] = data['comment_text'].str.lower()
data.head()
```

[6]:

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	0000997932d777bf	explanation\nwhy the edits made under my usern...	0	0	0	0	0	0
1	000103f0d9cfb60f	d'aww! he matches this background colour i'm s...	0	0	0	0	0	0
2	000113f07ec002fd	hey man, i'm really not trying to edit war. it...	0	0	0	0	0	0
3	0001b41b1c6bb37e	"\nmore\ni can't make any real suggestions on ...	0	0	0	0	0	0
4	0001d958c54c6e35	you, sir, are my hero. any chance you remember...	0	0	0	0	0	0

Figure 2.4 Lower case

2.1.5 Remove Punctuations

This can be clubbed with the step of removing special characters. Removing punctuation within reason is easy. It can be executed with the aid of the usage of string.Punctuation and retaining the entirety which isn't always in this list.

[7]:
```
import re
import string
data['comment_text'] = data['comment_text'].apply(lambda x: re.sub('[%s]' % re.escape(string.punctuation), '' , x))
data.head()
```

[7]:

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	0000997932d777bf	explanation\nwhy the edits made under my usern...	0	0	0	0	0	0
1	000103f0d9cfb60f	daww he matches this background colour im seem...	0	0	0	0	0	0
2	000113f07ec002fd	hey man im really not trying to edit war its j...	0	0	0	0	0	0
3	0001b41b1c6bb37e	\nmore\ni cant make any real suggestions on im...	0	0	0	0	0	0
4	0001d958c54c6e35	you sir are my hero any chance you remember wh...	0	0	0	0	0	0

Figure 2.5 Remove punctuations

2.1.6 Remove words and digits containing digits

```
#Remove words and digits containing digits
```

[11]:
```
data['comment_text'] = data['comment_text'].apply(lambda x: re.sub('W*dw*','',x))
data.head()
```

[11]:

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	0000997932d777bf	explanation why the eits mae uner my username ...	0	0	0	0	0	0
1	000103f0d9cfb60f	aww he matches this backgroun colour I am seem...	0	0	0	0	0	0
2	000113f07ec002fd	hey man I am really not trying to eit war its ...	0	0	0	0	0	0
3	0001b41b1c6bb37e	more i cannot make any real suggestions on imp...	0	0	0	0	0	0
4	0001d958c54c6e35	you sir are my hero any chance you remember wh...	0	0	0	0	0	0

Figure 2.6 Remove words and digits containing digits

2.1.7 Remove Stopwords

Stopwords are often added to sentences to cause them to be grammatically correct, for example, words along with a, is, an, the, and and so forth. These stopwords bring minimum to no significance and are to be had lots of open texts, articles, comments and so on. These ought to be eliminated so machine learning algorithms can better pay attention to phrases which define the

meaning/concept of the text. We are using a list from nltk.Corpus and this listing can in addition be greater by adding or doing away with custom words primarily based on the situation at hand.

```
#Remove Stopwords
```

```
[13]: from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
stop_words.add('subject')
stop_words.add('http')
def remove_stopwords(text):
    return " ".join([word for word in str(text).split() if word not in stop_words])
data['comment_text'] = data['comment_text'].apply(lambda x: remove_stopwords(x))
data.head()
```

[13]:

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	0000997932d777bf	explanation eits mae uner username harcore met..	0	0	0	0	0	0
1	000103f0d9cfb60f	aww matches backgroun colour I seemingly stuck..	0	0	0	0	0	0
2	000113f07ec002fd	hey man I really trying eit war guy constantly..	0	0	0	0	0	0
3	0001b41b1c6bb37e	cannot make real suggestions improvement woner..	0	0	0	0	0	0
4	0001d958c54c6e35	sir hero chance remember page	0	0	0	0	0	0

Figure 2.7 Remove Stopwords

2.1.8 Stemming

Stemming is the manner of lowering inflected/derived words to their phrase stem, base or root shape. The stem now does not equal the original phrase. There are many ways to carry out stemming which includes lookup tables, suffix-stripping algorithms etc. These specially depend on slicing-off ‘s’, ‘es’, ‘ed’, ‘ing’, ‘ly’ and so on from the end of the phrases and once in a while the conversion isn't proper. But stemming enables us to standardize text.

```
#stemming
```

```
[15]: from nltk.stem import PorterStemmer
def stemming(text):
    ps = PorterStemmer()
    return [ps.stem(word) for word in text]
data['comment_text'] = data['comment_text'].apply(lambda wrd: stemming(wrd))
data.head()
```

[15]:

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	0000997932d777bf	[explan, eit, mae, uner, usernam, harcor, meta...	0	0	0	0	0	0
1	000103f0d9cfb60f	[aww, match, backgroun, colour, i, seemingli, ...	0	0	0	0	0	0
2	000113f07ec002fd	[hey, man, i, realli, tri, eit, war, guy, cons...	0	0	0	0	0	0
3	0001b41b1c6bb37e	[cannot, make, real, suggest, improv, woner, s...	0	0	0	0	0	0
4	0001d958c54c6e35	[sir, hero, chanc, rememb, page]	0	0	0	0	0	0

Figure 2.8 Stemming

2.1.9 Tokenize Text

The procedure of splitting an input text into significant chunks is referred to as Tokenization, and that chew is truly known as token OR Given a character sequence and a defined file unit, tokenization is the mission of cutting it up into portions, known as tokens , perhaps on the same time throwing away certain characters, such as punctuation.

```
#tokenize Text
```

```
[14]: def tokenize(text):
          text = re.split('\s+' ,text)
          return [x.lower() for x in text]
      data['comment_text'] = data['comment_text'].apply(lambda msg : tokenize(msg))
      data.head()
```

[14]:

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	0000997932d777bf	[explanation, eits, mae, uner, username, harco...	0	0	0	0	0	0
1	000103f0d9cfb60f	[aww, matches, backgroun, colour, i, seemingly...	0	0	0	0	0	0
2	000113f07ec002fd	[hey, man, i, really, trying, eit, war, guy, c...	0	0	0	0	0	0
3	0001b41b1c6bb37e	[cannot, make, real, suggestions, improvement,...	0	0	0	0	0	0
4	0001d958c54c6e35	[sir, hero, chance, remember, page]	0	0	0	0	0	0

Figure 2.9 Tokenize text

2.1.10 Remove Whitespaces

```
#remove whitespaces
```

```
[17]: data["comment_text"] = data["comment_text"]
      data["comment_text"].str.strip()
      data.head()
```

[17]:

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	0000997932d777bf	[explan, eit, mae, uner, usernam, harcor, meta...	0	0	0	0	0	0
1	000103f0d9cfb60f	[aww, match, backgroun, colour, i, seemingli, ...	0	0	0	0	0	0
2	000113f07ec002fd	[hey, man, i, realli, tri, eit, war, guy, cons...	0	0	0	0	0	0
3	0001b41b1c6bb37e	[cannot, make, real, suggest, improv, woner, s...	0	0	0	0	0	0
4	0001d958c54c6e35	[sir, hero, chanc, rememb, page]	0	0	0	0	0	0

Figure 2.10 Remove Whitespaces

2.1.11 Lemmatization

Though stemming and lemmatization each generate the basis form of inflected/favored phrases, lemmatization is a complicated form of stemming. Stemming might not result in real words,

while lemmatization does conversion nicely with the use of vocabulary, usually aiming to put off inflectional endings best and to go back the bottom or dictionary form of a word, that's known as the lemma.

Before using lemmatization, we have to be conscious that it is extensively slower than stemming, so performance has to be kept in mind earlier than deciding on stemming or lemmatization.

In [54]:
```
#Lemmatization
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
def lemmatize_words(text):
    return " ".join([lemmatizer.lemmatize(word) for word in text.split()])
data["comment_text"] = data["comment_text"].apply(lambda text: lemmatize_words(text))
data.head()
```

Out[54]:

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	0000997932d777bf	Explanation Why the edits made under my userna...	0	0	0	0	0	0
1	000103f0d9cfb60f	D'awww! He match this background colour I'm see...	0	0	0	0	0	0
2	000113f07ec002fd	Hey man, I'm really not trying to edit war. It...	0	0	0	0	0	0
3	0001b41b1c6bb37e	" More I can't make any real suggestion on imp...	0	0	0	0	0	0
4	0001d958c54c6e35	You, sir, are my hero. Any chance you remember...	0	0	0	0	0	0

Figure 2.11 Lemmatization

2.2 EXPLORATORY DATA ANALYSIS

Exploratory Data Analysis is the process of exploring data, generating insights, testing hypotheses, checking assumptions and revealing underlying hidden patterns in the data. Exploratory facts evaluation (EDA) is used by statistics scientists to investigate and look at facts sets and summarize their fundamental characteristics, regularly employing records visualization techniques. It facilitates deciding how first-class to govern information sources to get the answers you want, making it less complicated for fact scientists to discover patterns, spot anomalies, test a hypothesis, or take a look at assumptions.

EDA is mainly used to look at what data can reveal beyond the formal modeling or hypothesis testing project and affords a better know-how of records set variables and the relationships between them. It also can help decide if the statistical techniques you're thinking about for statistics evaluation are appropriate. Originally advanced through American mathematician John

Tukey in the 1970s, EDA strategies continue to be a broadly used approach inside the records discovery system these days.

Few methods for EDA

2.2.1 Sentence Count — Total number of sentences in the text

```
[17]: data['Text'][0]

[17]: 'I have bought several of the Vitality canned dog food products and have found th
at and it smells better. My Labrador is finicky and she appreciates this product

Total number of sentences in the text

[18]: #open a file in read mode
with open('Data/Reviews.csv', 'rb') as f:
      text = str(f.read())
      num_sentences = str(text.count('.'))
      print("Number of sentences found: {}".format(num_sentences))

Number of sentences found: 1040293
```

Figure 3.1 Sentence count

2.2.2 Word Count — Total number of words in the text

Total number of words in the text

```
[19]: count = 0;
#Opens a file in read mode
file = open("Data/Reviews.csv", "r")
#Gets each line till end of file is reached
for line in file:
    #Splits each line into words
    words = line.split(" ");
    #Counts each word
    count = count + len(words);
print("Number of words present in given file: " + str(count));
file.close();

Number of words present in given file: 14585200
```

Figure 3.2 Word count

2.2.3 Character Count — Total number of characters in the text excluding spaces

Total number of characters in the text excluding spaces

```
[20]: #open file in read mode
      file = open("Data/Reviews.csv", "r")

      #read the content of file
      data = file.read()

      #get the length of the data
      number_of_characters = len(data)

      print('Number of characters in text file :', number_of_characters)
```

```
Number of characters in text file : 89128274
```

Figure 3.3 Character count

2.2.4 Sentence density — Number of sentences relative to the number of words

Number of sentences relative to the number of words

```
[21]: def lexical_diversity(data):
          return len(set(data)) / len(data)
      lexical_diversity(data)
```

```
[21]: 1.6268687083517404e-06
```

Figure 3.4 Sentence density

2.2.5 Word Density — Average length of the words

Average length of the words used in the headline

```
[22]: words = data.split()
      average = sum(len(word) for word in words) / len(words)
      average
```

```
[22]: 5.217640258194644
```

Figure 3.5 Word density

2.2.6 Punctuation Count — Total number of punctuations

Total number of punctuations used in the headline

```
[24]: from string import punctuation
      from collections import Counter
      with open('Data/Reviews.csv') as f: # closes the file for you which is important!
          c = Counter(c for line in f for c in line if c in punctuation)
      len(c)

[24]: 32
```

Figure 3.6 Punctuation count

2.2.7 Stopwords Count — Total number of common stopwords in the text

Total number of common stopwords in the text

```
[44]: from nltk.corpus import stopwords
      from nltk.tokenize import word_tokenize
      word_tokens = word_tokenize(str(data)) #splitta i pezzi
      data = [w for w in word_tokens if w in stop_words]
      len(data) / len(word_tokens) * 100

[44]: 23.809523809523807
```

Figure 3.7 Stopwords count

2.2.8 Word frequency analysis

word frequency analysis

[60]:
```
import pandas as pd
import nltk
import seaborn as sns
import matplotlib.pyplot as plt
data = pd.read_csv('Data/Reviews.csv', engine='python', encoding='utf-8', error_bad_lines=False)
top_N = 50
word_dist = nltk.FreqDist(data['Text'])
print('All frequencies')
print('='*60)
rslt=pd.DataFrame(word_dist.most_common(top_N),columns=['Word','Frequency'])
#sns.barplot(ax=axes[0],x='frequency',y='word',data=data2.head(30))
print(rslt)
print ('='*60)
```

```
Skipping line 166304: unexpected end of data
All frequencies
============================================================
                                                 Word  Frequency
0   Diamond Almonds<br />Almonds are a good source...         60
1   This review will make me sound really stupid, ...         59
2   According to the manufacturer's website, this ...         24
3   I'm addicted to salty and tangy flavors, so wh...         18
4   My two traditional striped cats eat mostly dry...         18
5   I have two cats, one 6 and one 2 years old. Bo...         15
6   I'd continue to buy but I'm moving over to mor...         14
7   I've been giving my daughter Gerber fruits, ve...         12
8   I have been buying this food locally but began...         12
9   First Impression: The friendly folks over at "...         11
10  My son loves this food.  He is 16 months now a...         10
11  I understand all the complaints about Science ...         10
12  I feed Hills Science diet to my pet because I ...          9
13  We have five cats - one an elderly cat of 15 y...          9
14  I am on a gluten free diet and find most glute...          9
15  Ok, sounds crazy i know but i heard about this...          9
```

[61]:
```
data['Text'].str.len().hist()
```

[61]: <AxesSubplot:>

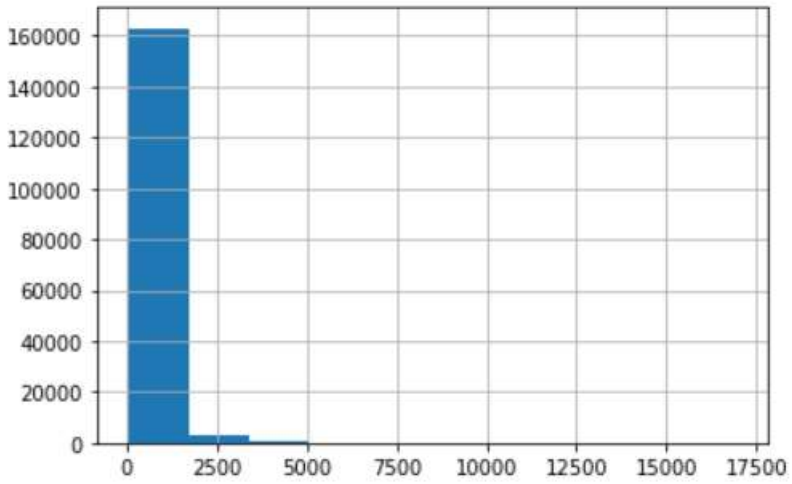

Figure 3.8 Word frequency analysis

2.2.9 Sentence length analysis

```
sentence length analysis
```

[59]:
```
file = open("Data/Reviews.csv", "r")
counter = 0
for c in file:
    counter+=1 #increment the counter
print(counter)
```

```
166304
```

[62]:
```
data['Text'].str.split().\
map(lambda x: len(x)).\
hist()
```

[62]: <AxesSubplot:>

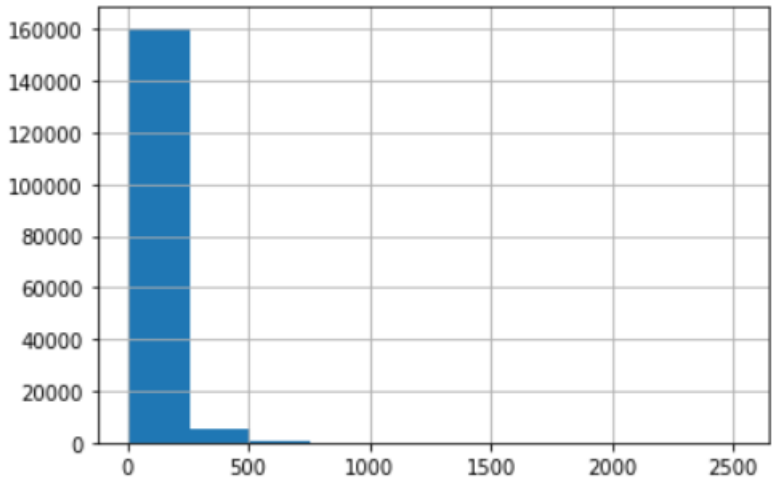

Figure 3.9 Sentence length analysis

2.2.10 Average word length analysis

average word length analysis

[53]:
```
file = open("Data/Reviews.csv", "r")
filtered = ''.join(filter(lambda x: x not in '".,;!-', file))
words = [word for word in filtered.split() if word]
avg = sum(map(len, words))/len(words)
print(avg)
```

```
5.217640258194644
```

Figure 3.10 Average word length

2.2.11.BoxPLot

method for graphically depicting groups of numerical data through their quartiles.

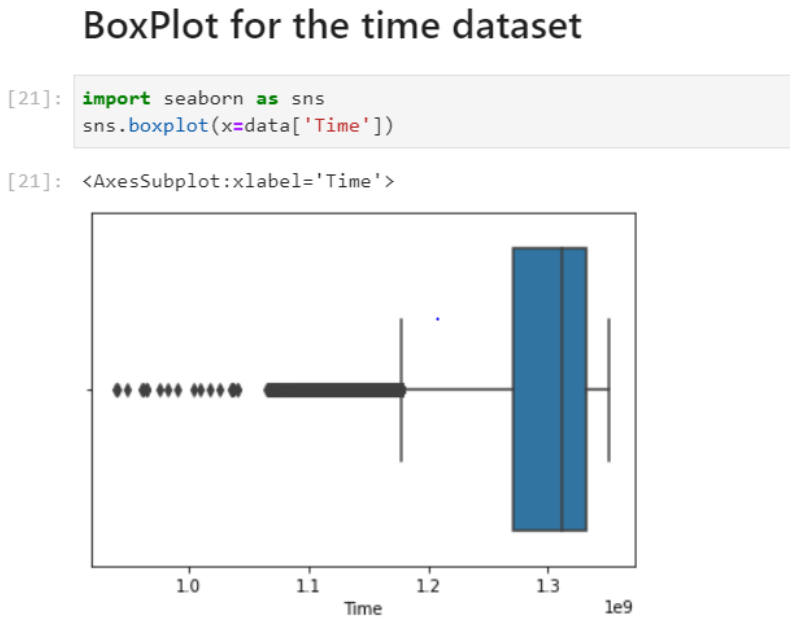

Figure 3.11 Boxplot for time dataset

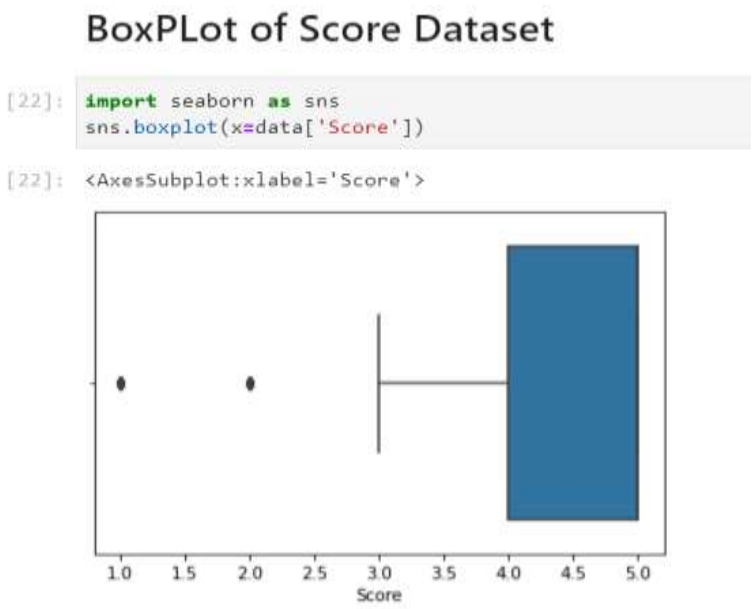

Figure 3.12 Boxplot of score dataset

2.2.12.Scatterplot

observe relationship between variables and uses dots to represent the relationship between them.

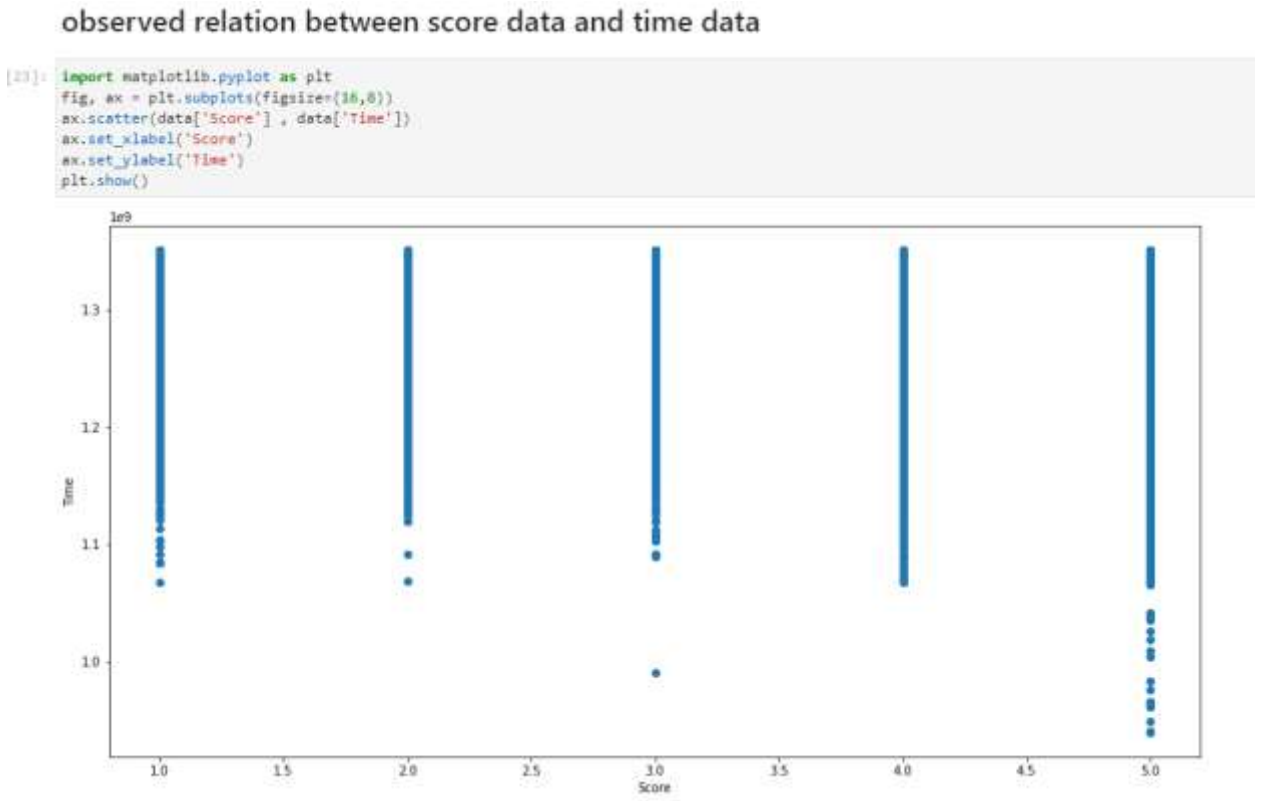

Figure 3.13 Scatterplot

2.2.13.Histogram

great tool for quickly assessing a probability distribution

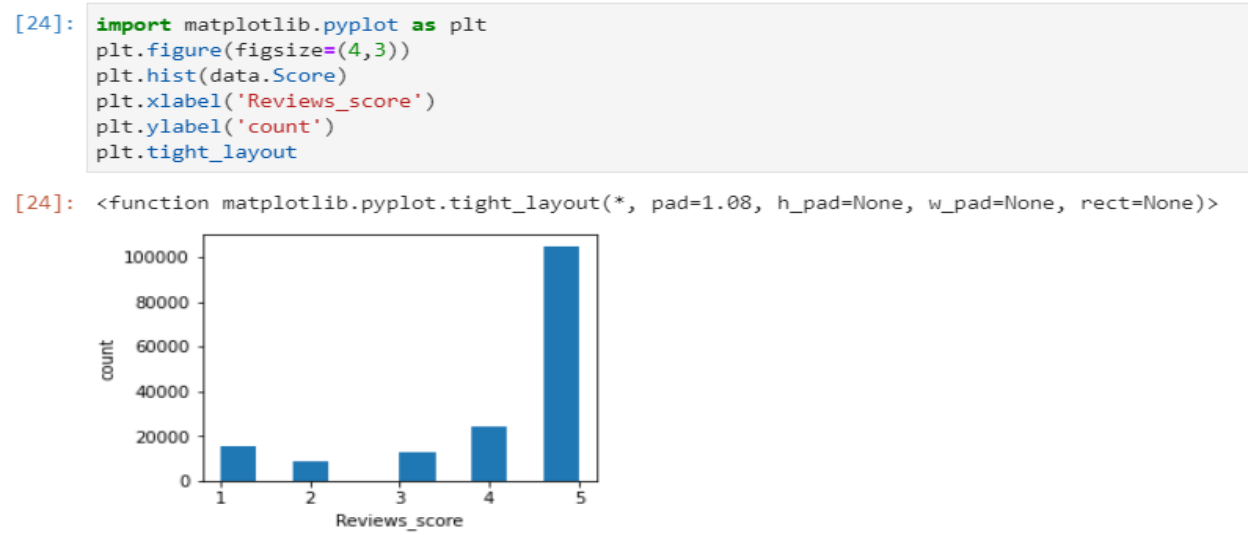

Figure 3.14 Histogram

2.2.14.Interquartile range(IQR)

measure of statistical dispersion,being equal to the difference between 75th and 25th percentiles, or between upper and lower quartiles.

```
[27]: Q1 = data.quantile(0.25)
      Q3 = data.quantile(0.75)
      IQR = Q3 - Q1
      print(IQR)
```

```
Id                          83150.5
HelpfulnessNumerator            2.0
HelpfulnessDenominator          2.0
Score                           1.0
Time                     61948800.0
dtype: float64
```

Figure 3.15 Interquartile range

```
[28]: data_outliner_IQR = data[~((data < (Q1 - 1.5 * IQR)) | (data > (Q3 + 1.5 * IQR))).any(axis=1)]
      data_outliner_IQR.shape
```

```
[28]: (129348, 10)
```

2.2.15.Plotting dataset for data type integers.

```
[29]: list(set(data.dtypes.tolist()))
```

```
[29]: [dtype('O'), dtype('int64')]
```

```
[30]: data_num = data.select_dtypes(include = ['int64'])
      data_num.head()
```

[30]:

	Id	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
0	1	1	1	5	1303862400
1	2	0	0	1	1346976000
2	3	1	1	4	1219017600
3	4	3	3	2	1307923200
4	5	0	0	5	1350777600

Figure 3.16 Plotting dataset

```
[31]: data_num.hist(figsize=(16, 20), bins=50, xlabelsize=8, ylabelsize=8); # ; avoid having the matplotlib verbose informations
```

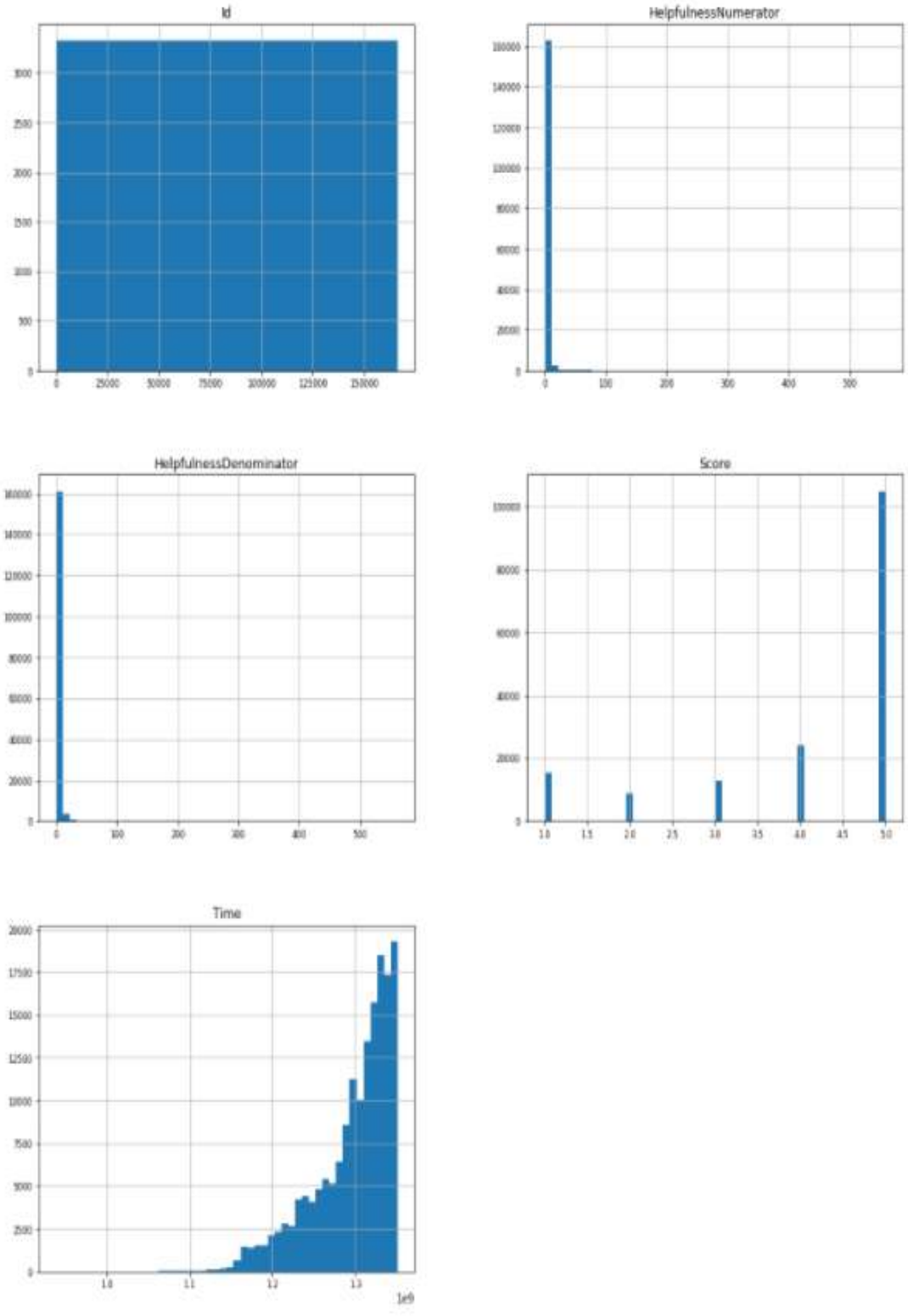

Figure 3.17 Graph of dataset

2.2.16.Describing the time.

```
[32]: print(data['Time'].describe())
      plt.figure(figsize=(9, 8))
      sns.distplot(data['Time'], color='g', bins=100, hist_kws={'alpha': 0.4});
```

```
count    1.663020e+05
mean     1.296082e+09
std      4.758881e+07
min      9.393408e+08
25%      1.270598e+09
50%      1.310947e+09
75%      1.332547e+09
max      1.351210e+09
Name: Time, dtype: float64
/home/bs0c98343/.local/lib/python3.6/site-packages/seaborn/distributions.py:2619: Futu
lot` (a figure-level function with similar flexibility) or `histplot` (an axes-level f
  warnings.warn(msg, FutureWarning)
```

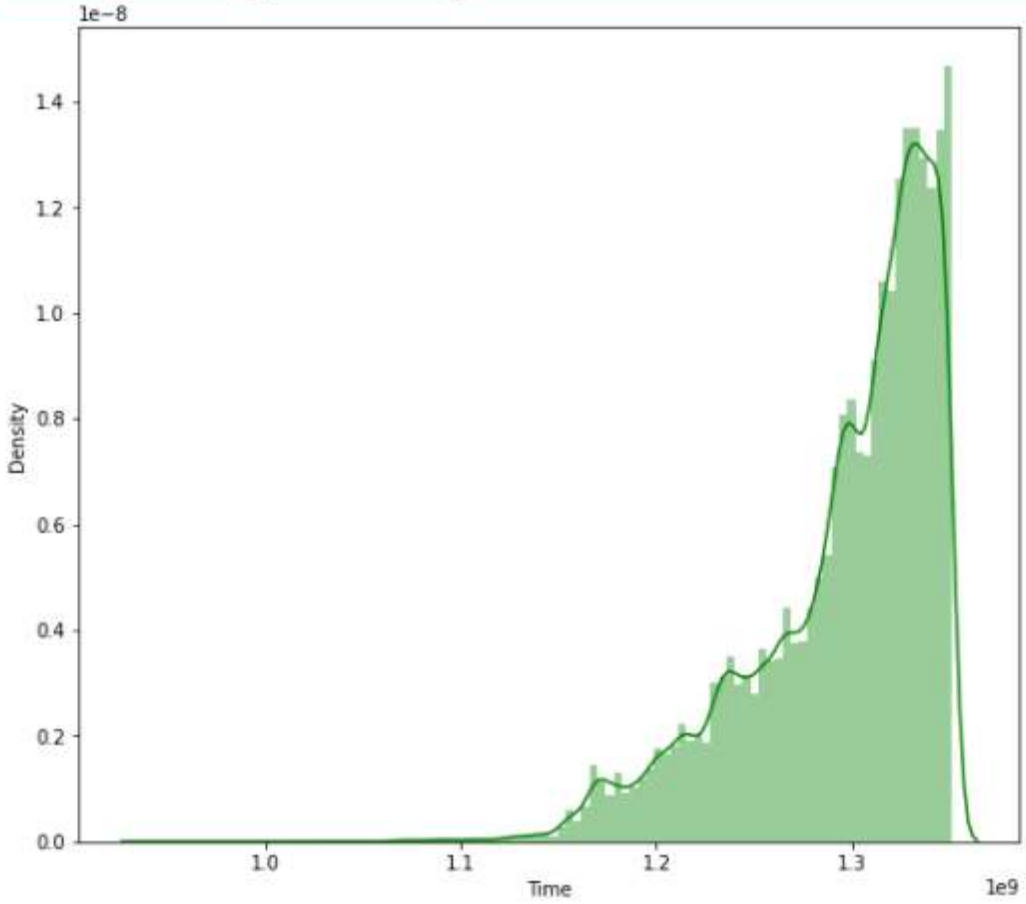

figure 3.18 Describing the time

2.2.17.Correlation plot

sample standard deviation for the Score.

```
[33]: for i in range(0, len(data_num.columns), 5):
          sns.pairplot(data=data_num,
                       x_vars=data_num.columns[i:i+5],
                       y_vars=['Score'])
```

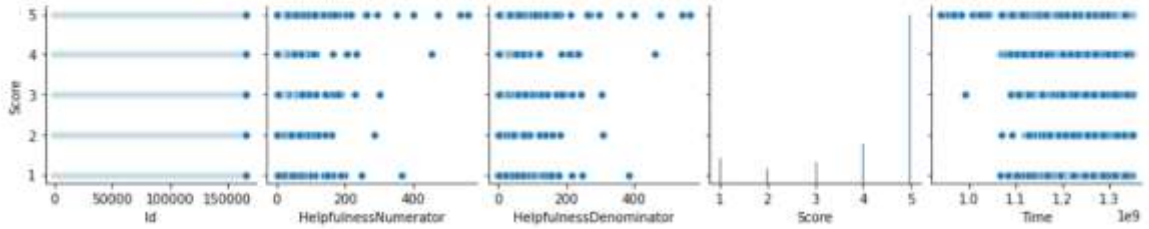

Figure 3.19 Correlation plot

2.2.18.Polarity

Sentiment analysis-the orientation of the expressed sentiment i.e.,it determines if the text expresses the positive, negative or neutral sentiment of the user about the entity in consideration.

```
[34]: from textblob import TextBlob
      def senti(x):
          return TextBlob(x).sentiment
      data['senti_score'] = data['Text'].apply(senti)
      data.senti_score.head()
```

```
[34]: 0                    (0.45, 0.43333333333333335)
      1    (-0.033333333333333326, 0.762962962962963)
      2      (0.1335714285714286, 0.4485714285714285)
      3     (0.16666666666666666, 0.5333333333333333)
      4                  (0.48333333333333334, 0.6375)
      Name: senti_score, dtype: object
```

Figure 3.20 Polarity

2.2.19.Stopword count in the Text data.

[35]:
```
#give the stopwords
stop=set(stopwords.words('english'))
corpus=[]
#defining the text
data= data['Text'].str.split()
data=data.values.tolist()
corpus=[word for i in data for word in i]
# import the default dictionary from collections

from collections import defaultdict
dic=defaultdict(int)
for word in corpus:
    if word in stop:
        dic[word]+=1
counter=Counter(corpus)
most=counter.most_common()

x, y= [], []
for word,count in most[:40]:
    if (word not in stop):
        x.append(word)
        y.append(count)
        plt.xlabel('count')
        plt.ylabel('word')

sns.barplot(x=y,y=x)
```

[35]:
```
<AxesSubplot:xlabel='count', ylabel='word'>
```

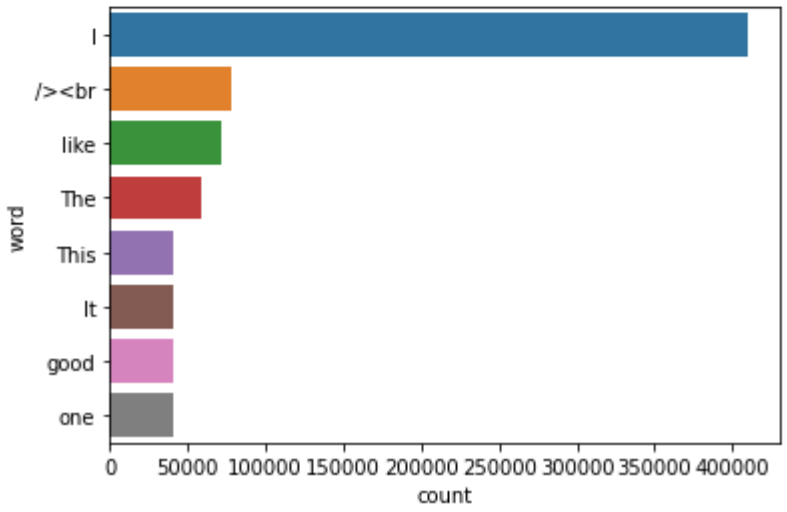

Figure 3.21 Stopword count

2.2.20.NGRAMS

contiguous sequences of n-items in a sentence. n can be 1, 2 or any other positive integers, although usually we do not consider very large N because those n-grams rarely appear in many different places.

```
[*]: from nltk.util import ngrams
with open("Data/Reviews.csv", "r", encoding='latin-1') as file:
    text = file.read()
def ngram_convertor(text,n=3):

    ngram_sentence = ngrams(text.split(), n)
    for item in ngram_sentence:
        print(item)
ngram_convertor(text,3)
```

```
('Id,ProductId,UserId,ProfileName,HelpfulnessNumerator,HelpfulnessDenominator,Score,T
1,1,5,1303862400,Good', 'Quality')
('1,B001E4KFG0,A3SGXH7AUHU8GW,delmartian,1,1,5,1303862400,Good', 'Quality', 'Dog')
('Quality', 'Dog', 'Food,I')
('Dog', 'Food,I', 'have')
('Food,I', 'have', 'bought')
('have', 'bought', 'several')
('bought', 'several', 'of')
('several', 'of', 'the')
('of', 'the', 'Vitality')
('the', 'Vitality', 'canned')
('Vitality', 'canned', 'dog')
('canned', 'dog', 'food')
('dog', 'food', 'products')
('food', 'products', 'and')
('products', 'and', 'have')
('and', 'have', 'found')
('have', 'found', 'them')
```

Figure 3.22 Ngrams

2.3 MULTI CLASS TEXT CLASSIFICATION USING LSTM

2.3.1 MULTI CLASS CLASSIFICATION

In device getting to know, multiclass or multinomial class is the hassle of classifying instances into considered one of 3 or more instructions (classifying instances into one in all two training is called binary classification).

While many category algorithms (extensively multinomial logistic regression) evidently permit the use of extra than instructions, a few are via nature binary algorithms; these can, however, be multinomial classifiers with the aid of a variety of strategies.

Multiclass categories have to not be stressed with multi-label type, in which more than one label is to be expected for every example.

2.3.2 Recurrent Neural Network:

Recurrent Neural Network(RNN) is a type of Neural Network where the output from the preceding step is fed as input to the contemporary step. This is a brief-time period memory to system Sequential statistics(Speech statistics, Music, Sequence of phrases in a text). Here is a pattern structure diagram.

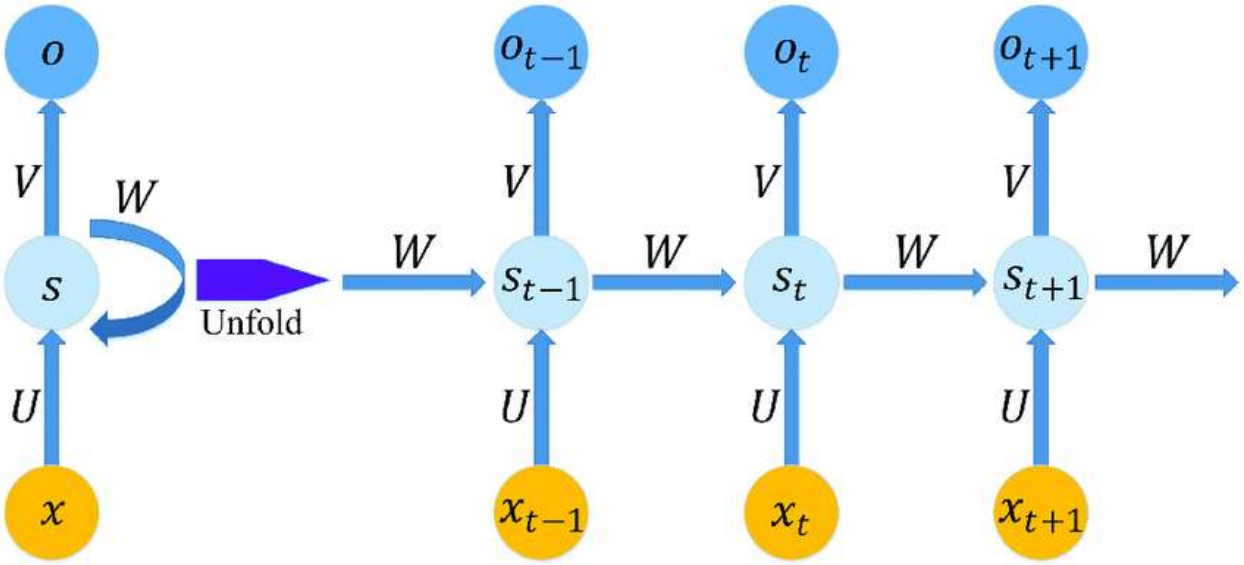

Figure 4 Recurrent neural network

Advantages:

1. RNN has a reminiscence that captures what has been calculated so far.
2. RNNs are perfect for textual content and speech records analysis.

Disadvantages:

1. RNN suffers from exploding and vanishing gradients, which makes the RNN version study slower by propagating a lesser amount of errors backward.

2. This works properly for quick sentences, when we address a protracted article, there can be a long time dependency problem

2.3.3 Long short-term memory (LSTM)

LSTM is a synthetic recurrent neural community (RNN) architecture used in the subject of deep learning. Unlike widespread feedforward neural networks, LSTM has remarkable connections. It can procedure not only single records factors (along with pix), however additionally entire sequences of information (such as speech or video). For example, LSTM is relevant to tasks consisting of unsegmented, connected handwriting recognition, speech popularity and anomaly detection in community traffic or IDSs (intrusion detection systems).

A commonplace LSTM unit is composed of a cellular, an input gate, an output gate and a forget about gate. The mobile recalls values over arbitrary time durations and the three gates alter the drift of statistics into and out of the cell.

LSTM networks are properly-desirable to classifying, processing and making predictions primarily based on time collection statistics, seeing that there may be lags of unknown length among vital occasions in a time series. LSTMs had been advanced to cope with the vanishing gradient problem that can be encountered when training traditional RNNs. Relative insensitivity to hole length is a bonus of LSTM over RNNs, hidden Markov models and different sequence learning methods in severa programs.

Architecture:

LSTM has chains of repeating the LSTM block. It is referred to as an LSTM cell. Each LSTM cell has 4 neural community layers interacting within.

1. Cell State
2. Forget Gate
3. Input Gate
4. Output Gate

Each LSTM cellular gets an enter from an Input collection, preceding mobile country and output from previous LSTM cellular.

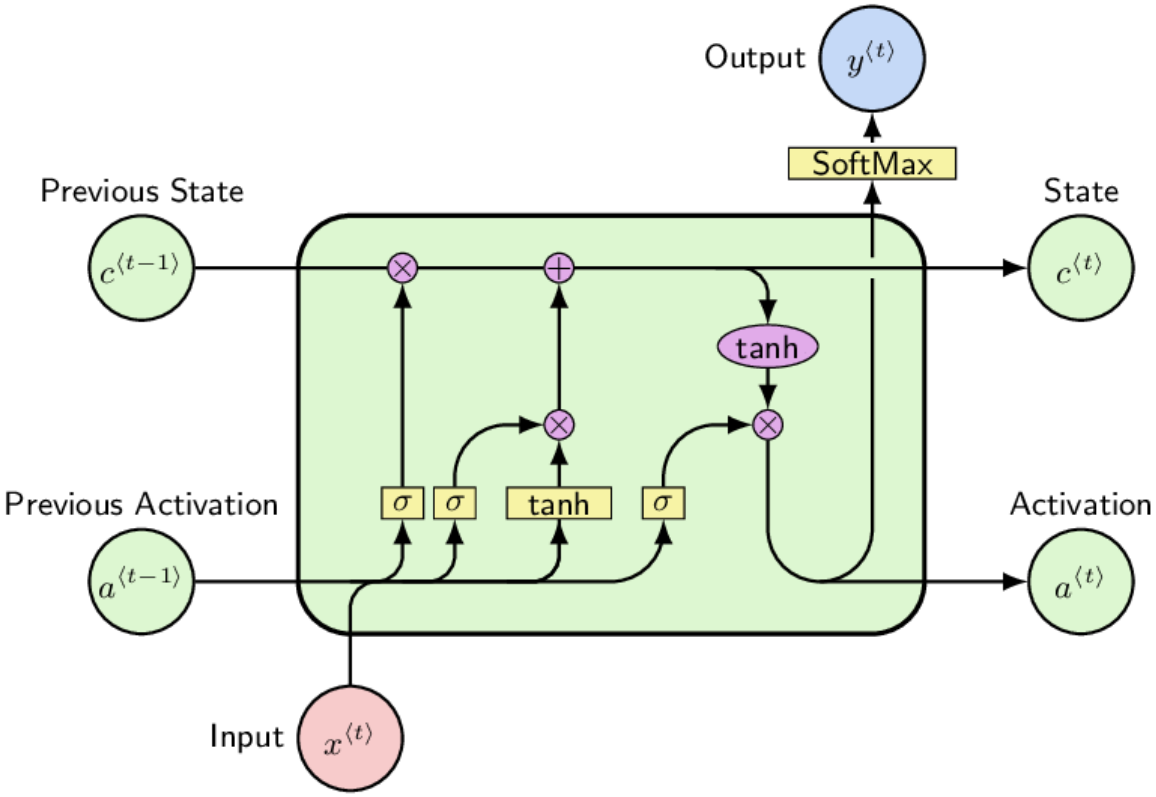

Figure 5 LSTM cellular

2.3.4 Bidirectional LSTM

Bidirectional LSTM trains two layers on the enter collection. One LSTM layer on the input series and 2nd LSTM layer at the reversed copy of the input sequence affords more context for studying sequences

Bidirectional LSTMs are an extension of traditional LSTMs that could enhance model performance on collection classification problems.

In problems where all timesteps of the input sequence are to be had, Bidirectional LSTMs train as opposed to one LSTMs on the enter series. The first on the input sequence as-is and the second one on a reversed copy of the enter series. This can provide additional context to the network and bring about faster or even fuller learning at the problem.

2.3.5 Data Description

- Use case short precis:

Sentiment analysis on film evaluations

- Use case precis:

Reviews of viewers about a film are classified primarily based on the emotions in the textual content of the review.

- Detailed Problem assertion:

This use case is to do sentiment evaluation at the Rotten Tomatoes movie overview dataset. The critiques are on a scale of five values: negative, extremely negative, impartial, relatively advantageous and tremendous. Obstacles like sentence negation, sarcasm, terseness, language ambiguity, and plenty of others make this undertaking very tough. This is a multi-class text classification hassle.

- Prerequisite:

Machine learning

Deep learning

Scientific Python Stack (which include NumPy, Pandas, TensorFlow, NLTK, Matplotlib)

- Data:

The dataset is made from tab-separated files with terms from the Rotten Tomatoes dataset. The train/test cut up has been preserved for the purposes of benchmarking, however the sentences have been shuffled from their original order. Each Sentence has been parsed into many phrases by using the Stanford parser. Each word has a PhraseId. Each sentence has a SentenceId. Phrases which can be repeated (which include short/common phrases) are handiest protected once in the data.Train.Tsv carries the phrases and their related sentiment labels. We have moreover furnished a SentenceId so you can tune which phrases belong to a single sentence.Take a look at.Tsv carries just phrases.

You have to assign a sentiment label to each phrase.

- The sentiment labels are:

0 - negative

1 - somewhat negative

2 - neutral

3 - somewhat positive

4 – positive

- Solution Approach:

Use Data Preprocessing & Feature Engineering like disposing of stop words, punctuations, URLs, lemmatization, stemming. Next step is to outline hyperparameters like embedding dimension, vocabulary size and so on. Then do tokenization and padding for information which could be entirely preprocessing. This information may be similarly used for training of fashions like LSTM, Bi-LSTM.

2.3.6 Coding

2.3.6.1 Importing libraries

```
[1]: import pandas as pd
     import numpy as np
     import re
     import keras
     from nltk.corpus import stopwords
     from nltk.tokenize import word_tokenize
     from nltk.stem import WordNetLemmatizer
     from nltk.stem import PorterStemmer
     from sklearn.model_selection import train_test_split
     from tqdm import tqdm
     from keras.models import Sequential
     from keras.layers.recurrent import LSTM, GRU
     from keras.layers.core import Dense, Activation, Dropout
     from keras.layers.embeddings import Embedding
     from keras.utils import np_utils
     from keras.layers import GlobalMaxPooling1D, Conv1D, MaxPooling1D, Flatten, Bidirectional, SpatialDropout1D
     from keras.utils import np_utils
     from keras.layers import GlobalMaxPooling1D, Conv1D, MaxPooling1D, Flatten, Bidirectional, SpatialDropout1D
     from sklearn.model_selection import train_test_split
     from keras.preprocessing import sequence
     from keras.preprocessing.text import Tokenizer
     from keras.callbacks import EarlyStopping
     from nltk.corpus import stopwords
     import matplotlib.pyplot as plt
```

Figure 6.1 Importing Libraries

We are importing NumPy for array operations and pandas to process data. Also importing PorterStemmer and WordNetLemmatizer from nltk library for data preprocessing. Also imported essential libraries for developing our Keras model.

2.3.6.2 Load Data

```
[2]: data = pd.read_csv('train.csv')
     data.head()
```

[2]:

	PhraseId	SentenceId	Phrase	Sentiment
0	1	1	A series of escapades demonstrating the adage ...	1
1	2	1	A series of escapades demonstrating the adage ...	2
2	3	1	A series	2
3	4	1	A	2
4	5	1	series	2

Figure 6.2 Loading data

2.3.6.3 Data Preprocessing:

Before going forward we are able to do a little record cleaning and preprocessing. There are several information preprocessing techniques like,

- Lower Case

```
: data['Phrase'] = data['Phrase'].str.lower()
  data.head()
```

:

	PhraseId	SentenceId	Phrase	Sentiment
0	1	1	a series of escapades demonstrating the adage ...	1
1	2	1	a series of escapades demonstrating the adage ...	2
2	3	1	a series	2
3	4	1	a	2
4	5	1	series	2

Figure 6.3 lower case

- Removing Punctuations

```
import re
import string
data['Phrase'] = data['Phrase'].apply(lambda x: re.sub('[%s]' % re.escape(string.punctuation), '' , x))
data.head()
```

	PhraseId	SentenceId	Phrase	Sentiment
0	1	1	a series of escapades demonstrating the adage ...	1
1	2	1	a series of escapades demonstrating the adage ...	2
2	3	1	a series	2
3	4	1	a	2
4	5	1	series	2

Figure 6.4 Removing punctuations

- Removing Stopwords

```
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
stop_words.add('subject')
stop_words.add('http')
def remove_stopwords(text):
    return " ".join([sentence for sentence in str(text).split() if sentence not in stop_words])
data['Phrase'] = data['Phrase'].apply(lambda x: remove_stopwords(x))
data.head()
```

	PhraseId	SentenceId	Phrase	Sentiment
0	1	1	series escapades demonstrating adage good goos...	1
1	2	1	series escapades demonstrating adage good goose	2
2	3	1	series	2
3	4	1		2
4	5	1	series	2

Figure 6.5 Removing Stopwords

- Stemming

```
def stemming(text):
    tokens = word_tokenize(text)
    ps = PorterStemmer()
    tokens = [ps.stem(word) for word in tokens]
    return ' '.join(tokens)
data['Phrase'] = list(map(stemming,data['Phrase']))
data.head()
```

	PhraseId	SentenceId	Phrase	Sentiment
0	1	1	seri escapad demonstr adag good goos also good...	1
1	2	1	seri escapad demonstr adag good goos	2
2	3	1	seri	2
3	4	1		2
4	5	1	seri	2

Figure 6.6 Stemming

- Lemmatization

```
def getLemmText(text):
    tokens = word_tokenize(text)
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(word) for word in tokens]
    return ' '.join(tokens)
data["Phrase"] = list(map(getLemmText,data["Phrase"]))
data.head()
```

	PhraseId	SentenceId	Phrase	Sentiment
0	1	1	seri escapad demonstr adag good goo also good ...	1
1	2	1	seri escapad demonstr adag good goo	2
2	3	1	seri	2
3	4	1		2
4	5	1	seri	2

Figure 6.7 Lemmatization

2.3.6.4 Splitting dataset into training and test sets

We will split the data into training and test sets. We will do it using train_test_split from the model_selection module of scikit-learn. Here we will split our data in such a way that 2/3rd data row we will use as training data and 1/3rd will use to validate the model.

```
xtrain, xtest, ytrain, ytest = train_test_split(data["Phrase"], data["Sentiment"], test_size=0.33, random_state=53)
print(xtrain.shape)
print(xtest.shape)
print(ytrain)
```

```
(104560,)
(51500,)
19545     2
73333     1
60892     2
16872     1
154131    2
         ..
139096    2
154431    3
8854      1
55717     2
35701     2
Name: Sentiment, Length: 104560, dtype: int64
```

Figure 6.8 Splitting data

2.3.6.5 Defining Hyperparameters

```
EMBEDDING_DIMENSION = 64
VOCABULARY_SIZE = 2000
MAX_LENGTH = 100
OOV_TOK = '<OOV>'
TRUNCATE_TYPE = 'post'
PADDING_TYPE = 'post'
```

Figure 6.9 Defining hyperparameters

- “EMBEDDING_DIMENSION: It defines the embedding dimensions of our vector.
- “VOCABULARY_SIZE”: It defines the maximum wide variety of phrases in tokenizer.
- “MAX_LENGTH”: It defines the most duration of every sentence, such as padding.
- “OOV_TOK”: This is to position a special value in when an unseen phrase is encountered

2.3.6.6 Tokenization

The subsequent step is to tokenize our data and construct word_index from it. We will use Keras Tokenizer. In our instance, it's going to take 2,000 of the most not unusual words. We will put <00V> for those phrases which aren't in the word_index. fit_on_text.

```
tokenizer = Tokenizer(num_words=VOCABULARY_SIZE, oov_token=OOV_TOK)
tokenizer.fit_on_texts(list(xtrain) + list(xtest))
```

Figure 6.10 Tokenization

The subsequent step is to turn the ones tokens into lists of sequences. We will use texts_to_sequences() approach to do this. So, our phrase dictionary will be like this.

Turning tokens into list of Sequence

```
xtrain_sequences = tokenizer.texts_to_sequences(xtrain)
xtest_sequences = tokenizer.texts_to_sequences(xtest)
word_index = tokenizer.word_index
print('Vocabulary size:', len(word_index))
dict(list(word_index.items())[0:10])
```

```
Vocabulary size: 11856
{'<OOV>': 1,
 'film': 2,
 'movi': 3,
 'nt': 4,
 'one': 5,
 'like': 6,
 'make': 7,
 'charact': 8,
 'stori': 9,
 'time': 10}
```

Figure 6.11 Turning tokens to split

2.3.6.7 Padding:

Now we are able to add padding to our records to make it uniform. Keras makes it smooth to pad our information by means of the usage of pad_sequences characteristic. And we can print the 101nth document after making use of padding.

```
xtrain_pad = sequence.pad_sequences(xtrain_sequences, maxlen=MAX_LENGTH, padding=PADDING_TYPE, truncating=TRUNCATE_TYPE)
xtest_pad = sequence.pad_sequences(xtest_sequences, maxlen=MAX_LENGTH, padding=PADDING_TYPE, truncating=TRUNCATE_TYPE)
print(len(xtrain_sequences[0]))
print(len(xtrain_pad[0]))
print(xtrain_pad[1000])
```

```
3
100
[   1  159   15  410  243  541   38 1568 1899  849    1 1118    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0]
```

Figure 6.12 Padding

Before schooling a deep neural community, we must explore what our authentic textual content and text after padding appear like.

Text after our padding look like

```
reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])
def decode_article(text):
 return ''.join([reverse_word_index.get(i, '') for i in text])
print(decode_article(xtrain_pad[1]))
```

```
commonsaccharingenr
```

Figure 6.13 Text after padding

2.3.6.8 Implementation of Long Short Term Memory (LSTM):

We finished records preprocessing and word embedding. Now we will create a sequential model, with Embedding an LSTM layer.

- Model Initialization

```
model = Sequential()
model.add(Embedding(len(word_index) + 1,EMBEDDING_DIMENSION))
model.add(SpatialDropout1D(0.3))
model.add(Bidirectional(LSTM(EMBEDDING_DIMENSION, dropout=0.3, recurrent_dropout=0.3)))
model.add(Dense(EMBEDDING_DIMENSION, activation='relu'))
model.add(Dropout(0.8))
model.add(Dense(EMBEDDING_DIMENSION, activation='relu'))
model.add(Dropout(0.8))
model.add(Dense(5))
model.add(Activation('softmax'))
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam',metrics=['accuracy'])
```

Figure 6.14 Model Initilization

- Model Training

```
xtest_pad = np.array([0 for _ in range(len(xtrain_pad)//2)] + [1 for _ in range(len(xtrain_pad)//2)])
history = model.fit(xtrain_pad, ytrain, epochs=5, batch_size=64, validation_data=(xtest_pad, ytrain))
```

```
Epoch 1/5
1634/1634 [==============================] - 256s 157ms/step - loss: 1.1546 - accuracy: 0.5443 - val_loss: 1.3384 - val_accuracy: 0.5107
Epoch 2/5
1634/1634 [==============================] - 252s 154ms/step - loss: 1.1123 - accuracy: 0.5571 - val_loss: 1.3648 - val_accuracy: 0.5107
Epoch 3/5
1634/1634 [==============================] - 253s 155ms/step - loss: 1.0775 - accuracy: 0.5673 - val_loss: 1.3756 - val_accuracy: 0.5107
Epoch 4/5
1634/1634 [==============================] - 252s 154ms/step - loss: 1.0591 - accuracy: 0.5786 - val_loss: 1.3708 - val_accuracy: 0.5107
Epoch 5/5
1634/1634 [==============================] - 255s 156ms/step - loss: 1.0433 - accuracy: 0.5855 - val_loss: 1.3735 - val_accuracy: 0.5107
```

Figure 6.15 Model training

- Model Evaluation

```
def graph_plots(history, string):
  plt.plot(history.history[string])
  plt.plot(history.history['val_'+string])
  plt.xlabel("Epochs")
  plt.ylabel(string)
  plt.legend([string, 'val_'+string])
  plt.show()

graph_plots(history, "accuracy")
graph_plots(history, "loss")
```

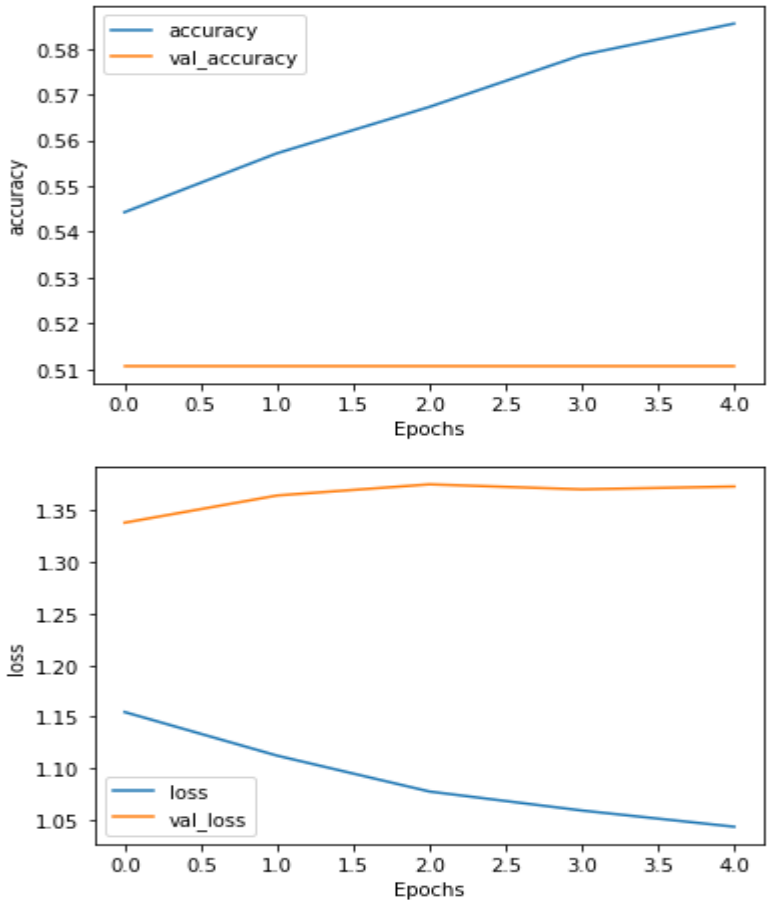

Figure 6.16 Model evaluation

2.3.7 Future Work:

1. We can filter the specific businesses like restaurants and then use LSTM for sentiment analysis.

2. We can use much larger dataset with more epochs to increase the accuracy.
3. More hidden dense layers can be used to improve the accuracy. We can tune other hyper parameters as well.

2.4 MULTI LABEL TEXT CLASSIFICATION WITH BERT AND PYTORCH LIGHTNING

Multi-label text classification (or tagging text) is one of the most common tasks you'll encounter when doing NLP. Modern Transformer-based models (like BERT) make use of pre-training on vast amounts of text data that makes fine-tuning faster, use fewer resources and more accurate on small(er) datasets.

At the end of 2018 researchers at Google AI Language open-sourced a new method for Natural Language Processing (NLP) known as BERT (Bidirectional Encoder Representations from Transformers) — a main leap forward which took the Deep Learning community by hurricane due to its exquisite overall performance.

2.4.1 Why is BERT Needed

One of the largest challenges in NLP is the lack of enough schooling records. Overall there may be a sizable quantity of textual content records to be had, but if we want to create challenge-particular datasets, we need to break up that pile into the very many numerous fields. And whilst we try this, we become with only some thousand or a few hundred thousand human-classified training examples. Unfortunately, if you want to perform nicely, deep studying primarily based on NLP fashions require lots larger quantities of facts — they see fundamental enhancements while educated on tens of millions, or billions, of annotated education examples. To help bridge this gap in records, researchers have evolved numerous techniques for training well known purpose language illustration models the use of the massive piles of unannotated text on the web (that is called pre-training). These trendy reasons pre-educated models can then be pleasant-tuned on smaller challenge-unique datasets, e.G., when working with troubles like query answering and sentiment analysis. This method results in tremendous accuracy enhancements

compared to schooling at the smaller venture-specific datasets from scratch. BERT is a recent addition to these strategies for NLP pre-education; it induced a stir in the deep learning network as it presented state-of-the-art effects in a huge form of NLP duties, like query answering.

2.4.2 How does BERT work

The high-quality part approximately BERT is that it may be download and used totally free — we can both use the BERT models to extract excessive satisfactory language features from our textual content records, or we will best-song those fashions on a selected mission, like sentiment analysis and question answering, with our own data to supply cutting-edge predictions.
BERT is predicated on a Transformer (the eye mechanism that learns contextual relationships between phrases in a text). A fundamental Transformer includes an encoder to study the text input and a decoder to provide a prediction for the challenge. Since BERT's intention is to generate a language illustration version, it most effectively wishes the encoder component. The input to the encoder for BERT is a chain of tokens, which can be first converted into vectors and then processed in the neural community. But earlier than processing can begin, BERT needs the input to be massaged and embellished with some extra metadata:

- Token embeddings: A [CLS] token is delivered to the input word tokens at the start of the primary sentence and a [SEP] token is inserted on the give up of each sentence.
- Segment embeddings: A marker indicating Sentence A or Sentence B is brought to each token. This permits the encoder to differentiate between sentences.
- Positional embeddings: A positional embedding is delivered to each token to signify its function inside the sentence.

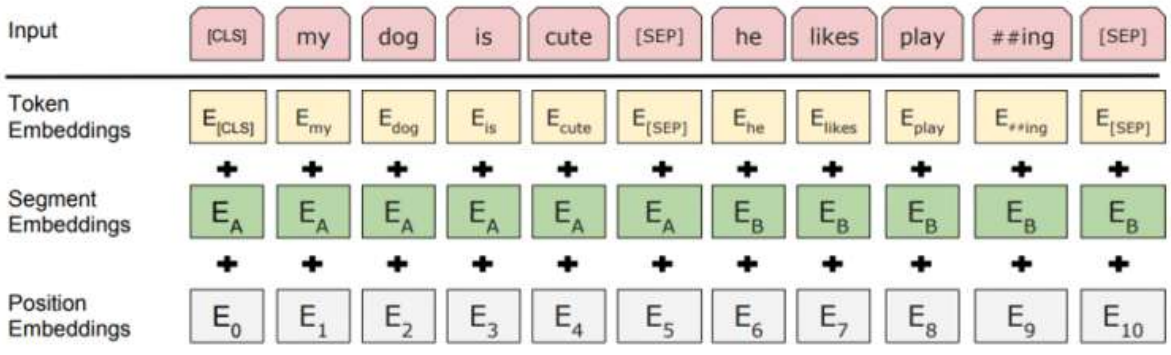

Figure 7 Embedding

2.4.3 Architecture

There are four sorts of pre-educated versions of BERT relying on the scale of the model structure:

- BERT-Base: 12-layer, 768-hidden-nodes, 12-interest-heads, 110M parameters
- BERT-Large: 24-layer, 1024-hidden-nodes, sixteen-attention-heads, 340M parameters

2.4.4 Coding

2.4.4.1 Creating Environment

```
:  import os

   os.environ["CUDA_DEVICE_ORDER"] = "PCI_BUS_ID"

   os.environ["CUDA_VISIBLE_DEVICES"] = '1,3'
```

Figure 8.1 Creating environment

2.4.4.2 Importing Libraries

```
import pandas as pd
import numpy as np
from tqdm.auto import tqdm
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
from transformers import BertTokenizerFast as BertTokenizer, BertModel, AdamW, get_linear_schedule_with_warmup
import pytorch_lightning as pl
from pytorch_lightning.metrics.functional import accuracy, f1, auroc
from pytorch_lightning.callbacks import ModelCheckpoint, EarlyStopping
from pytorch_lightning.loggers import TensorBoardLogger
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, multilabel_confusion_matrix
import seaborn as sns
from pylab import rcParams
import matplotlib.pyplot as plt
from matplotlib import rc
%matplotlib inline
%config InlineBackend.figure_format='retina'
RANDOM_SEED = 42
sns.set(style='whitegrid', palette='muted', font_scale=1.2)
HAPPY_COLORS_PALETTE = ["#01BEFE", "#FFDD00", "#FF7D00", "#FF006D", "#ADFF02", "#8F00FF"]
sns.set_palette(sns.color_palette(HAPPY_COLORS_PALETTE))
rcParams['figure.figsize'] = 12, 8
pl.seed_everything(RANDOM_SEED)
```

```
Global seed set to 42
```

```
42
```

Figure 8.2 Importing libraries

2.4.4.3 Load Data

```
data = pd.read_csv('train_master.csv')
data.head()
```

	Unnamed: 0	PhraseId	SentenceId	Phrase	Negative	Somewhat_Negative	Neutral	Somewhat_Positive	Positive
0	0	1	1	A series of escapades demonstrating the adage ...	0	1	0	0	0
1	1	2	1	A series of escapades demonstrating the adage ...	0	0	1	0	0
2	2	3	1	A series	0	0	1	0	0
3	3	4	1	A	0	0	1	0	0
4	4	5	1	series	0	0	1	0	0

Figure 8.3 Load data

2.4.4.4 Splitting the data

```
train_data, val_data = train_test_split(data, test_size=0.05)
train_data.shape, val_data.shape
```

```
((148257, 9), (7803, 9))
```

Figure 8.4 Splitting data

2.4.4.5 Preprocessing

distribution of labels

```
LABEL_COLUMNS = data.columns.tolist()[4:]
```

```
data[LABEL_COLUMNS].sum().sort_values().plot(kind="barh");
```

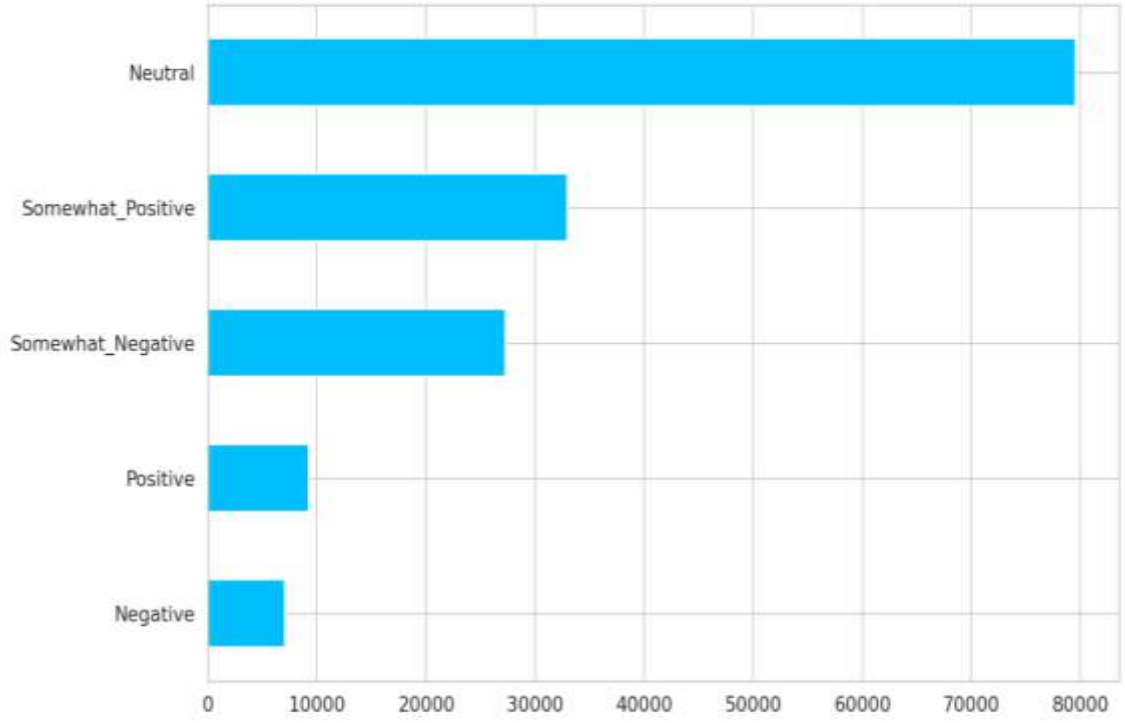

Figure 8.5 Distribution of labels

creating a new training set

```
#train_data = pd.concat([
#  train_toxic,
#  train_clean.sample(15_000)
#])
train_data.shape, val_data.shape
```

```
((148257, 9), (7803, 9))
```

Figure 8.6 Creating new training set

2.4.4.6 Tokenization

We need to convert the raw text into a list of tokens. For that, we'll use the built-in BertTokenizer:

Using bulit-in BertTokenizer

```
BERT_MODEL_NAME = 'bert-base-cased'
tokenizer = BertTokenizer.from_pretrained(BERT_MODEL_NAME)
```

```
sample_row = data.iloc[100]
sample_comment = sample_row.Phrase
sample_labels = sample_row[LABEL_COLUMNS]
print(sample_comment)
print()
print(sample_labels.to_dict())
```

```
would have a hard time sitting through this one .

{'Negative': 0, 'Somewhat_Negative': 1, 'Neutral': 0, 'Somewhat_Positive': 0, 'Positive': 0}
```

Figure 8.6 bulit-in BertTokenizer

```
encoding = tokenizer.encode_plus(
  sample_comment,
  add_special_tokens=True,
  max_length=512,
  return_token_type_ids=False,
  padding="max_length",
  return_attention_mask=True,
  return_tensors='pt',
)
encoding.keys()
```

```
dict_keys(['input_ids', 'attention_mask'])
```

```
encoding["input_ids"].shape, encoding["attention_mask"].shape
```

```
(torch.Size([1, 512]), torch.Size([1, 512]))
```

Figure 8.7 Encoding

The result of the encoding is a dictionary with token ids input_ids and an attention mask attention_mask (which tokens should be used by the model 1 - use or 0 - don't use).

```
encoding["input_ids"].squeeze()[:20]
```

```
tensor([ 101, 1156, 1138,  170, 1662, 1159, 2807, 1194, 1142, 1141,  119,  102,
           0,    0,    0,    0,    0,    0,    0,    0])
```

```
encoding["attention_mask"].squeeze()[:20]
```

```
tensor([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0])
```

Figure 8.8 result of encoding

You can also inverse the tokenization and get back (kinda) the words from the token ids:

```
print(tokenizer.convert_ids_to_tokens(encoding["input_ids"].squeeze())[:10])
```

```
['[CLS]', 'would', 'have', 'a', 'hard', 'time', 'sitting', 'through', 'this', 'one']
```

Figure 8.9 Inverse tokenizer

We need to specify the maximum number of tokens when encoding (512 is the maximum we can do).

number of tokens per sentiment

```
token_counts = []
for _, row in train_data.iterrows():
  token_count = len(tokenizer.encode(
    row["Phrase"],
    max_length=512,
    truncation=True
  ))
  token_counts.append(token_count)
```

```
sns.histplot(token_counts)
plt.xlim([0, 512]);
```

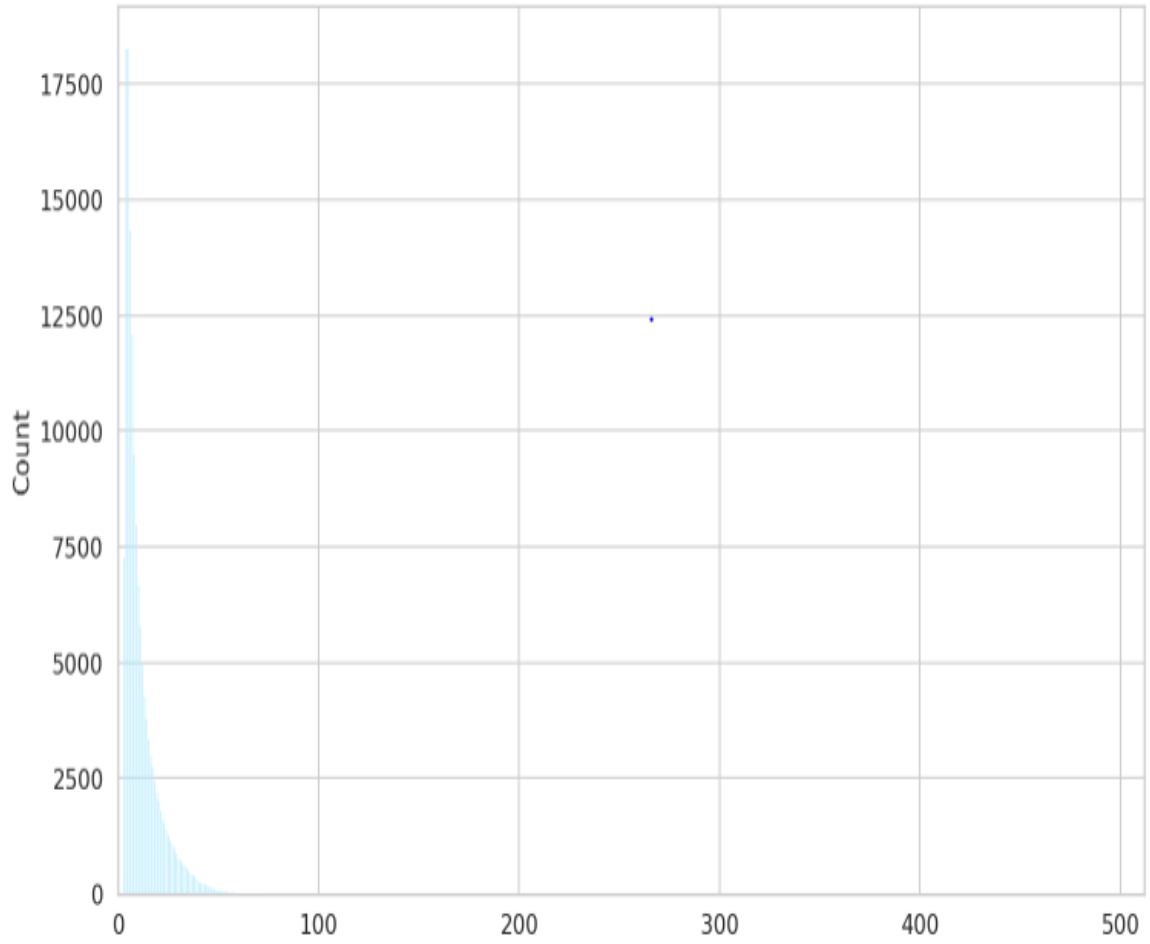

Figure 8.10 number of tokens per sentiment

Most of the comments contain less than 300 tokens or more than 512. So, we'll stick with the limit of 512.

```
MAX_TOKEN_COUNT = 512
```

2.4.4.7 Dataset

We'll wrap the tokenization process in a PyTorch Dataset, along with converting the labels to tensors

```
class ToxicCommentsDataset(Dataset):
  def __init__(
    self,
    data: pd.DataFrame,
    tokenizer: BertTokenizer,
    max_token_len: int = 128
  ):
    self.tokenizer = tokenizer
    self.data = data
    self.max_token_len = max_token_len
  def __len__(self):
    return len(self.data)
  def __getitem__(self, index: int):
    data_row = self.data.iloc[index]
    Phrase = data_row.Phrase
    labels = data_row[LABEL_COLUMNS]
    encoding = self.tokenizer.encode_plus(
      Phrase,
      add_special_tokens=True,
      max_length=self.max_token_len,
      return_token_type_ids=False,
      padding="max_length",
      truncation=True,
      return_attention_mask=True,
      return_tensors='pt',
    )
    return dict(
      Phrase=Phrase,
      input_ids=encoding["input_ids"].flatten(),
      attention_mask=encoding["attention_mask"].flatten(),
      labels=torch.FloatTensor(labels)
    )
```

Figure 8.11 converting the labels to tensors

Let’s have a look at a sample item from the dataset:

```
train_dataset = ToxicCommentsDataset(
  train_data,
  tokenizer,
  max_token_len=MAX_TOKEN_COUNT
)
sample_item = train_dataset[0]
sample_item.keys()
```

```
dict_keys(['Phrase', 'input_ids', 'attention_mask', 'labels'])
```

```
sample_item["Phrase"]
```

```
'that would make watching such a graphic treatment of the crimes bearable'
```

```
sample_item["labels"]
```

```
tensor([0., 1., 0., 0., 0.])
```

```
sample_item["input_ids"].shape
```

```
torch.Size([512])
```

Figure 8.12 sample from the dataset

Let’s load the BERT model and pass a sample of batch data through

```
bert_model = BertModel.from_pretrained(BERT_MODEL_NAME, return_dict=True)
sample_batch = next(iter(DataLoader(train_dataset, batch_size=8, num_workers=2)))
sample_batch["input_ids"].shape, sample_batch["attention_mask"].shape
```

```
Some weights of the model checkpoint at bert-base-cased were not used when initializin
ht', 'cls.predictions.transform.dense.weight', 'cls.predictions.decoder.weight', 'cls.
- This IS expected if you are initializing BertModel from the checkpoint of a model tr
eTraining model).
- This IS NOT expected if you are initializing BertModel from the checkpoint of a mode
cation model).
```

```
(torch.Size([8, 512]), torch.Size([8, 512]))
```

```
output = bert_model(sample_batch["input_ids"], sample_batch["attention_mask"])
```

```
output.last_hidden_state.shape, output.pooler_output.shape
```

```
(torch.Size([8, 512, 768]), torch.Size([8, 768]))
```

Figure 8.13 bert model and passing a sample

The 768 dimension comes from the BERT hidden size:

```
bert_model.config.hidden_size
```

```
768
```

The larger version of BERT has more attention heads and a larger hidden size.

```
class ToxicCommentDataModule(pl.LightningDataModule):
  def __init__(self, train_data, test_data, tokenizer, batch_size=8, max_token_len=128):
    super().__init__()
    self.batch_size = batch_size
    self.train_data = train_data
    self.test_data = test_data
    self.tokenizer = tokenizer
    self.max_token_len = max_token_len
  def setup(self, stage=None):
    self.train_dataset = ToxicCommentsDataset(
      self.train_data,
      self.tokenizer,
      self.max_token_len
    )
    self.test_dataset = ToxicCommentsDataset(
      self.test_data,
      self.tokenizer,
      self.max_token_len
    )
  def train_dataloader(self):
    return DataLoader(
      self.train_dataset,
      batch_size=self.batch_size,
      shuffle=True,
      num_workers=2
    )
  def val_dataloader(self):
    return DataLoader(
      self.test_dataset,
      batch_size=self.batch_size,
      num_workers=2
    )
  def test_dataloader(self):
    return DataLoader(
      self.test_dataset,
      batch_size=self.batch_size,
      num_workers=2
    )
```

Figure 8.14 wrap our dataset into Lightningdatamodule

ToxicCommentDataModule encapsulates all data loading logic and returns the necessary data loaders.

create an instance of our data module

```
N_EPOCHS = 10
BATCH_SIZE = 12
data_module = ToxicCommentDataModule(
  train_data,
  val_data,
  tokenizer,
  batch_size=BATCH_SIZE,
  max_token_len=MAX_TOKEN_COUNT
)
```

Figure 8.15 instance of our data module

2.4.4.8 Model

Our model will use a pre-trained BertModel and a linear layer to convert the BERT representation to a classification task.

Most of the implementation is just a boilerplate. Two points of interest are the way we configure the optimizers and calculating the area under ROC. We'll dive a bit deeper into those next.

```
class ToxicCommentTagger(pl.LightningModule):
  def __init__(self, n_classes: int, n_training_steps=None, n_warmup_steps=None):
    super().__init__()
    self.bert = BertModel.from_pretrained(BERT_MODEL_NAME, return_dict=True)
    self.classifier = nn.Linear(self.bert.config.hidden_size, n_classes)
    self.n_training_steps = n_training_steps
    self.n_warmup_steps = n_warmup_steps
    self.criterion = nn.BCELoss()
  def forward(self, input_ids, attention_mask, labels=None):
    output = self.bert(input_ids, attention_mask=attention_mask)
    output = self.classifier(output.pooler_output)
    output = torch.sigmoid(output)
    loss = 0
    if labels is not None:
        loss = self.criterion(output, labels)
    return loss, output
  def training_step(self, batch, batch_idx):
    input_ids = batch["input_ids"]
    attention_mask = batch["attention_mask"]
    labels = batch["labels"]
    loss, outputs = self(input_ids, attention_mask, labels)
    self.log("train_loss", loss, prog_bar=True, logger=True)
    return {"loss": loss, "predictions": outputs, "labels": labels}
  def validation_step(self, batch, batch_idx):
    input_ids = batch["input_ids"]
    attention_mask = batch["attention_mask"]
    labels = batch["labels"]
    loss, outputs = self(input_ids, attention_mask, labels)
    self.log("val_loss", loss, prog_bar=True, logger=True)
    return loss
  def test_step(self, batch, batch_idx):
    input_ids = batch["input_ids"]
    attention_mask = batch["attention_mask"]
    labels = batch["labels"]
    loss, outputs = self(input_ids, attention_mask, labels)
    self.log("test_loss", loss, prog_bar=True, logger=True)
    return loss
  def training_epoch_end(self, outputs):
    labels = []
    predictions = []
    for output in outputs:
      for out_labels in output["labels"].detach().cpu():
        labels.append(out_labels)
      for out_predictions in output["predictions"].detach().cpu():
        predictions.append(out_predictions)
    labels = torch.stack(labels).int()
    predictions = torch.stack(predictions)
    for i, name in enumerate(LABEL_COLUMNS):
      class_roc_auc = auroc(predictions[:, i], labels[:, i])
      self.logger.experiment.add_scalar(f"{name}_roc_auc/Train", class_roc_auc, self.current_epoch)
  def configure_optimizers(self):
    optimizer = AdamW(self.parameters(), lr=2e-5)
    scheduler = get_linear_schedule_with_warmup(
      optimizer,
      num_warmup_steps=self.n_warmup_steps,
      num_training_steps=self.n_training_steps
    )
    return dict(
      optimizer=optimizer,
      lr_scheduler=dict(
        scheduler=scheduler,
        interval='step'
      )
    )
```

Figure 8.16 packing everything in Lightning Module

2.4.4.9 Optimizer Scheduler

The job of a scheduler is to change the learning rate of the optimizer during training. This might lead to better performance of our model. We'll use the get_linear_schedule_with_warmup.

```
dummy_model = nn.Linear(2, 1)
optimizer = AdamW(params=dummy_model.parameters(), lr=0.001)
warmup_steps = 20
total_training_steps = 100
scheduler = get_linear_schedule_with_warmup(
  optimizer,
  num_warmup_steps=warmup_steps,
  num_training_steps=total_training_steps
)
learning_rate_history = []
for step in range(total_training_steps):
  optimizer.step()
  scheduler.step()
  learning_rate_history.append(optimizer.param_groups[0]['lr'])
plt.plot(learning_rate_history, label="learning rate")
plt.axvline(x=warmup_steps, color="red", linestyle=(0, (5, 10)), label="warmup end")
plt.legend()
plt.xlabel("Step")
plt.ylabel("Learning rate")
plt.tight_layout();
```

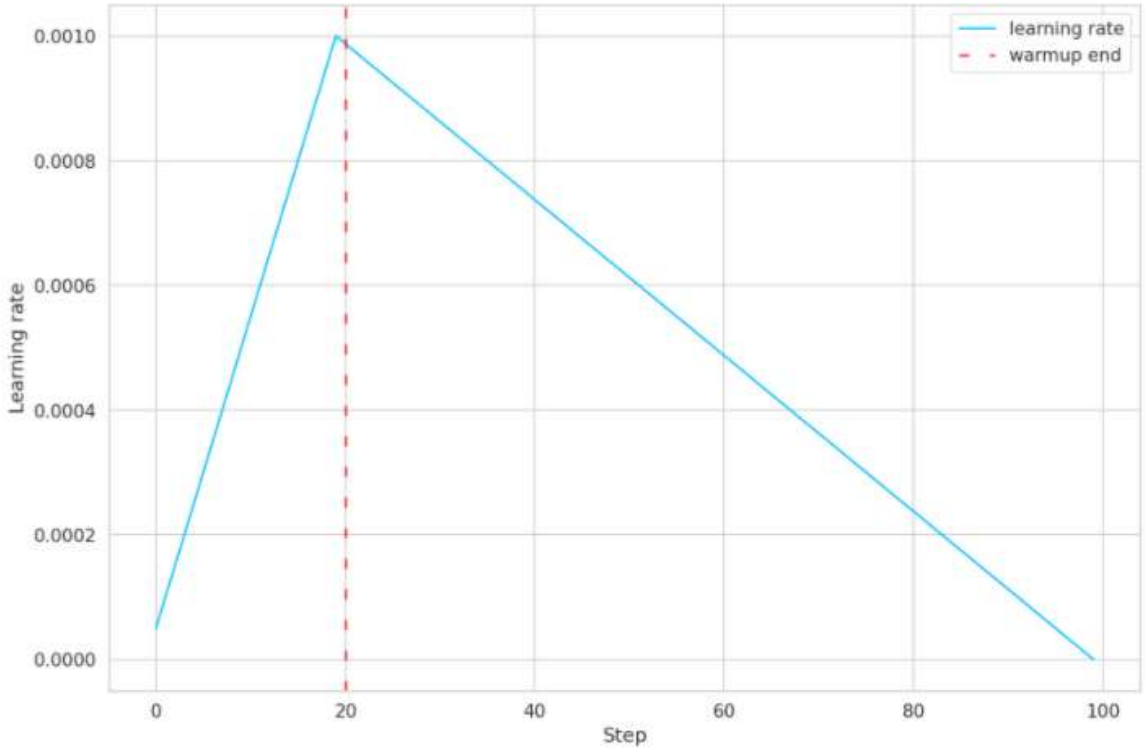

Figure 8.17 Optmizer shedule

We simulate 100 training steps and tell the scheduler to warm up for the first 20. The learning rate grows to the initial fixed value of 0.001 during the warm-up and then goes down (linearly) to 0.

To use the scheduler, we need to calculate the number of training and warm-up steps. The number of training steps per epoch is equal to number of training examples / batch size. The number of total training steps is training steps per epoch * number of epochs:

```
steps_per_epoch=len(train_data) // BATCH_SIZE
total_training_steps = steps_per_epoch * N_EPOCHS
```

We'll use a fifth of the training steps for a warm-up:

```
warmup_steps = total_training_steps // 5
warmup_steps, total_training_steps
```

```
(24708, 123540)
```

We can now create an instance of our model

```
model = ToxicCommentTagger(
  n_classes=len(LABEL_COLUMNS),
  n_warmup_steps=warmup_steps,
  n_training_steps=total_training_steps
)
```

```
Some weights of the model checkpoint at bert-base-cased were not used when initializing BertModel: ['cls.pred
ls.predictions.transform.LayerNorm.bias', 'cls.predictions.bias', 'cls.seq_relationship.weight']
- This IS expected if you are initializing BertModel from the checkpoint of a model trained on another task
- This IS NOT expected if you are initializing BertModel from the checkpoint of a model that you expect to be
```

Figure 8.18 creating an instance of our model

2.4.4.10 Evaluation

Multi-label classification boils down to doing binary classification for each label/tag.We'll use Binary Cross Entropy to measure the error for each label. PyTorch has BCELoss, which we're going to combine with a sigmoid function (as we did in the model implementation).

```
criterion = nn.BCELoss()
prediction = torch.FloatTensor(
  [10.95873564, 1.07321467, 1.58524066, 0.03839076, 15.72987556, 1.09513213]
)
labels = torch.FloatTensor(
  [1., 0., 0., 0., 1., 0.]
)
```

```
torch.sigmoid(prediction)
```

```
tensor([1.0000, 0.7452, 0.8299, 0.5096, 1.0000, 0.7493])
```

```
criterion(torch.sigmoid(prediction), labels)
```

```
tensor(0.8725)
```

Figure 8.19 Evaluation

We can use the same approach to calculate the loss of the predictions

```
_, predictions = model(sample_batch["input_ids"], sample_batch["attention_mask"])
predictions
```

```
tensor([[0.5580, 0.5190, 0.3674, 0.3750, 0.4789],
        [0.5918, 0.4959, 0.3639, 0.3666, 0.5021],
        [0.5549, 0.5441, 0.4071, 0.3812, 0.4795],
        [0.5532, 0.5414, 0.4032, 0.3753, 0.5005],
        [0.5825, 0.5218, 0.3686, 0.3710, 0.4966],
        [0.5576, 0.5287, 0.3919, 0.3752, 0.4893],
        [0.5149, 0.5340, 0.4192, 0.3790, 0.4947],
        [0.5397, 0.4904, 0.4043, 0.3884, 0.4835]], grad_fn=<SigmoidBackward>)
```

```
criterion(predictions, sample_batch["labels"])
```

```
tensor(0.6721, grad_fn=<BinaryCrossEntropyBackward>)
```

Figure 8.20 calculate the loss of the predictions

2.4.4.11 ROC Curve

Another metric we're going to use is the area under the Receiver operating characteristic (ROC) for each tag. ROC is created by plotting the True Positive Rate (TPR) vs False Positive Rate (FPR):

$$TPR=TP+FN/TP$$

$$FPR=FP+TN/FP$$

```
from sklearn import metrics
fpr = [0.        , 0.        , 0.        , 0.02857143, 0.02857143,
       0.11428571, 0.11428571, 0.2       , 0.4       , 1.        ]
tpr = [0.        , 0.01265823, 0.67202532, 0.76202532, 0.91468354,
       0.97468354, 0.98734177, 0.98734177, 1.        , 1.        ]
_, ax = plt.subplots()
ax.plot(fpr, tpr, label="ROC")
ax.plot([0.05, 0.95], [0.05, 0.95], transform=ax.transAxes, label="Random classifier", color="red")
ax.legend(loc=4)
ax.set_xlabel("False positive rate")
ax.set_ylabel("True positive rate")
ax.set_title("Example ROC curve")
plt.show();
```

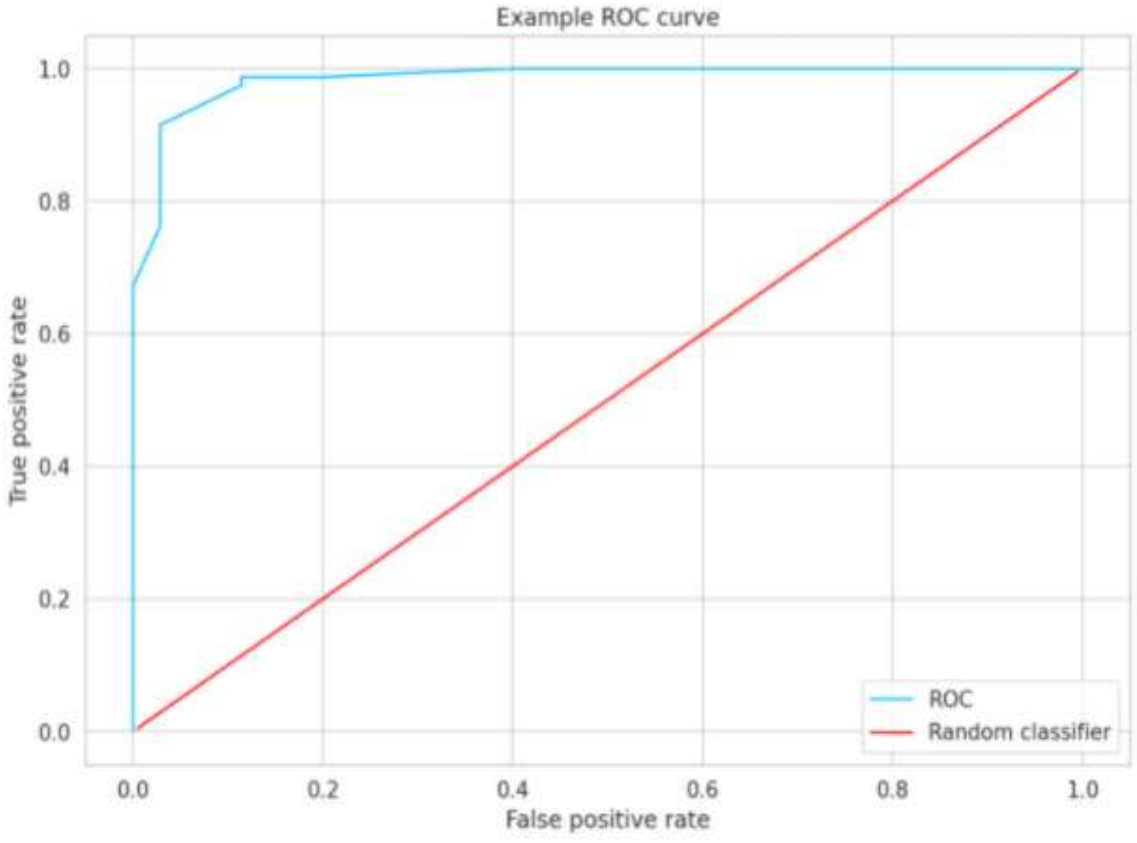

Figure 8.21 ROC Curve

2.5 NGRAMS

N-grams of texts are extensively used in text mining and natural language processing tasks. They are basically a set of co-occurring words within a given window and when computing the n-grams you typically move one word forward (although you can move X words forward in more advanced scenarios).

If X=Num of words in a given sentence K, the number of n-grams for sentence K would be:

$$Ngrams_K = X - (N - 1)$$

N-grams are used for a variety of different task. For example, when developing a language model, n-grams are used to develop not just unigram models but also bigram and trigram models. Google and Microsoft have developed web scale n-gram models that can be used in a variety of tasks such as spelling correction, word breaking and text summarization.

Another use of n-grams is for developing features for supervised Machine Learning models such as SVMs, MaxEnt models, Naive Bayes, etc. The idea is to use tokens such as bigrams in the feature space instead of just unigrams. But please be warned that from my personal experience and various research papers that I have reviewed, the use of bigrams and trigrams in your feature space may not necessarily yield any significant improvement.

2.5.1 IMPORTING THE LIBRARIES

```
import pandas as pd
import re
import unicodedata
import nltk
from gensim.models import Word2Vec
from nltk.corpus import stopwords
# add appropriate words that will be ignored in the analysis
ADDITIONAL_STOPWORDS = ['covfefe']

import matplotlib.pyplot as plt
```

```
df = pd.read_csv('train.csv')
```

Figure 9.1 importing libraries

2.5.2 CLEANING THE DATA

```
def basic_clean(text):
  wnl = nltk.stem.WordNetLemmatizer()
  stopwords = nltk.corpus.stopwords.words('english') + ADDITIONAL_STOPWORDS
  text = (unicodedata.normalize('NFKD', text)
    .encode('ascii', 'ignore')
    .decode('utf-8', 'ignore')
    .lower())
  words = re.sub(r'[^\w\s]', '', text).split()
  return [wnl.lemmatize(word) for word in words if word not in stopwords]
```

```
words = basic_clean(''.join(str(df['Phrase'].tolist())))
```

Figure 9.2 cleaning data

2.5.3 BIGRAM

```
(pd.Series(nltk.ngrams(words, 2)).value_counts())[:10]
```

```
(ca, nt)               489
(romantic, comedy)     326
(feel, like)           319
(wo, nt)               253
(new, york)            193
(love, story)          192
(lrb, rrb)             181
(subject, matter)      172
(special, effect)      171
(running, time)        169
dtype: int64
```

Figure 9.3 bigram

2.5.4 TRIGRAM

```
(pd.Series(nltk.ngrams(words, 3)).value_counts())[:10]
```

```
(world, war, ii)          71
(ca, nt, help)            49
(fat, greek, wedding)     47
(big, fat, greek)         45
(trouble, every, day)     41
(new, york, city)         36
(nt, feel, like)          36
(anne, rice, novel)       35
(make, feel, like)        33
(nearly, three, hour)     33
dtype: int64
```

Figure 9.4 trigram

2.5.5 PLOTING MOST FREQUENT WORDS IN BIGRAMS

```
bigrams_series.sort_values().plot.barh(color='blue', width=.9, figsize=(12, 8))
plt.title('20 Most Frequently Occuring Bigrams')
plt.ylabel('Bigram')
plt.xlabel('# of Occurances')
```

```
Text(0.5, 0, '# of Occurances')
```

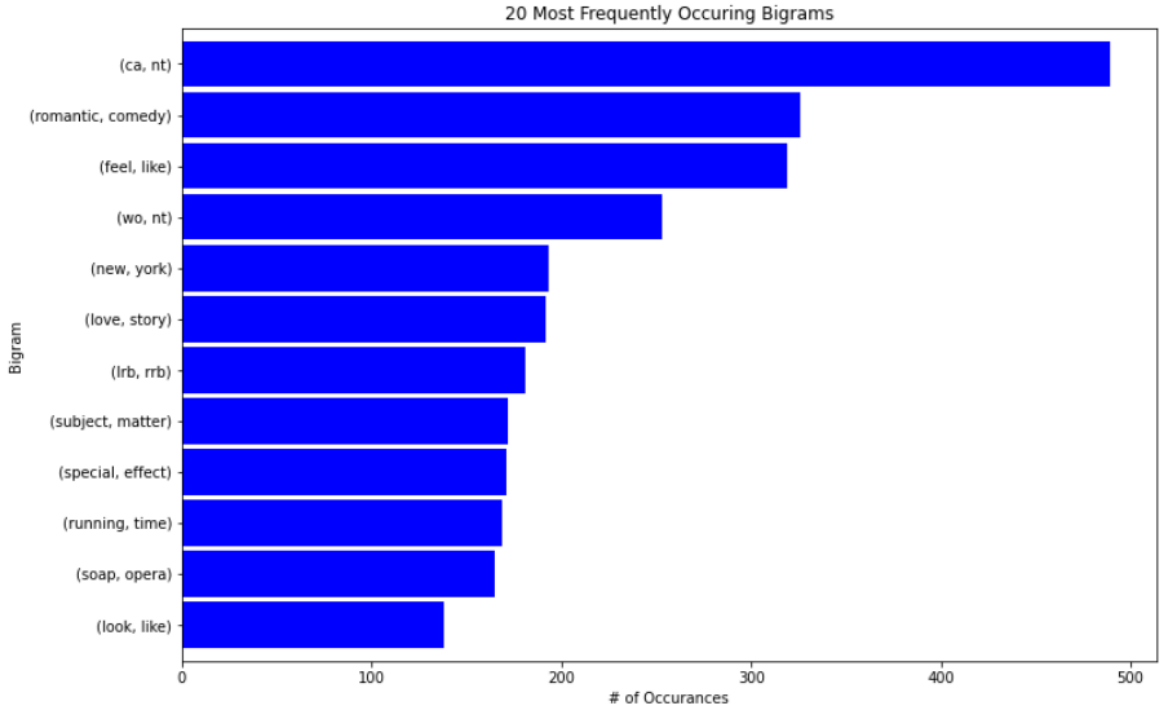

Figure 9.5 ploting bigram

2.6 WORD EMBEDDING

The mapping of words into numerical vector spaces — has proved to be an incredibly important method for natural language processing (NLP) tasks in recent years, enabling various machine learning models that rely on vector representation as input to enjoy richer representations of text input. These representations preserve more semantic and syntactic information on words, leading to improved performance in almost every imaginable NLP task.

A word embedding is a learned representation for text where words that have the same meaning have a similar representation.

It is this approach to representing words and documents that may be considered one of the key breakthroughs of deep learning on challenging natural language processing problems.

Word embeddings are in fact a class of techniques where individual words are represented as real-valued vectors in a predefined vector space. Each word is mapped to one vector and the vector values are learned in a way that resembles a neural network, and hence the technique is often lumped into the field of deep learning.

Key to the approach is the idea of using a dense distributed representation for each word.

Each word is represented by a real-valued vector, often tens or hundreds of dimensions. This is contrasted to the thousands or millions of dimensions required for sparse word representations, such as a one-hot encoding.

2.6.1 Word Embedding Algorithms

Word embedding methods learn a real-valued vector representation for a predefined fixed sized vocabulary from a corpus of text.

The learning process is either joint with the neural network model on some task, such as document classification, or is an unsupervised process, using document statistics.

This section reviews three techniques that can be used to learn a word embedding from text data.

2.6.1.1 Embedding Layer

An embedding layer, for lack of a better name, is a word embedding that is learned jointly with a neural network model on a specific natural language processing task, such as language modeling or document classification.
It requires that document text be cleaned and prepared such that each word is one-hot encoded. The size of the vector space is specified as part of the model, such as 50, 100, or 300 dimensions. The vectors are initialized with small random numbers. The embedding layer is used on the front end of a neural network and is fit in a supervised way using the Backpropagation algorithm.

2.6.1.2 Word2Vec

Word2Vec is a statistical method for efficiently learning a standalone word embedding from a text corpus.

It was developed by Tomas Mikolov, et al. at Google in 2013 as a response to make the neural-network-based training of the embedding more efficient and since then has become the de facto standard for developing pre-trained word embedding.

Two different learning models were introduced that can be used as part of the word2vec approach to learn the word embedding; they are:

- Continuous Bag-of-Words, or CBOW model.
- Continuous Skip-Gram Model.

The CBOW model learns the embedding by predicting the current word based on its context. The continuous skip-gram model learns by predicting the surrounding words given a current word.

The continuous skip-gram model learns by predicting the surrounding words given a current word.

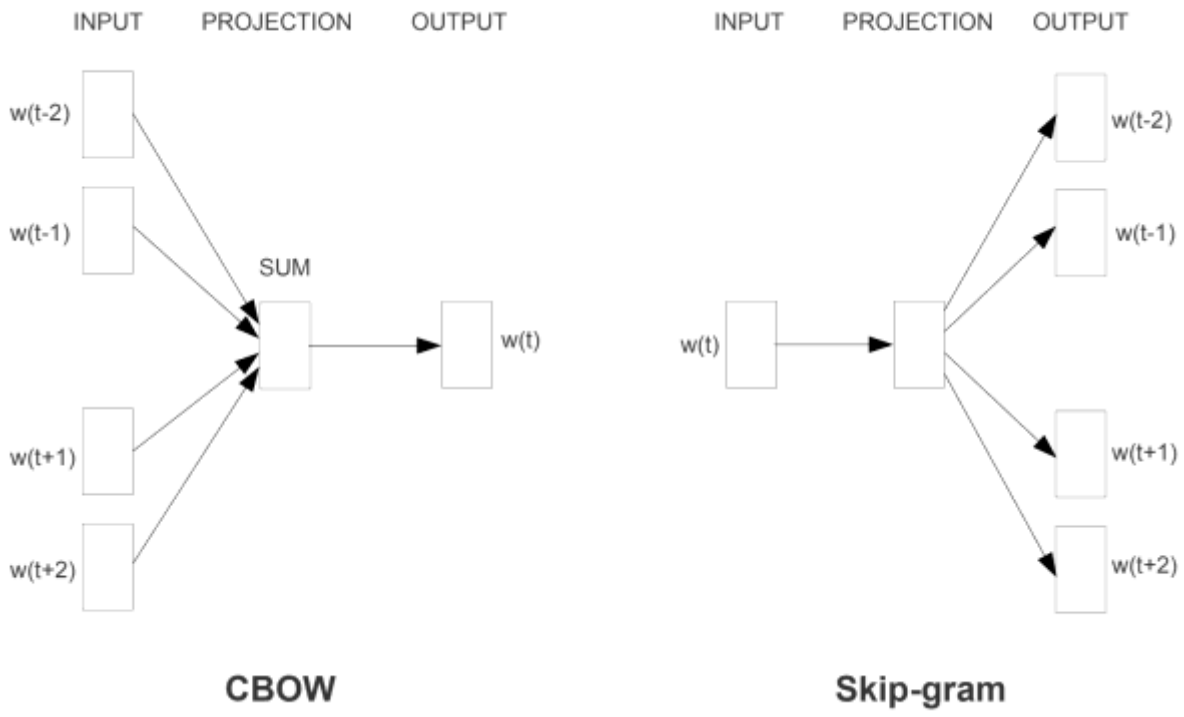

Figure 10 Word2Vec Training Models

2.6.2 READING THE DATA

```
text = pd.read_csv("train.csv")
text.head()
```

	PhraseId	SentenceId	Phrase	Sentiment
0	1	1	A series of escapades demonstrating the adage ...	1
1	2	1	A series of escapades demonstrating the adage ...	2
2	3	1	A series	2
3	4	1	A	2
4	5	1	series	2

```
corpus_text = 'n'.join(text[:1000]['Phrase'])
```

Figure 11.1 reading data

2.6.3 CLEANING THE DATA

```
#remove extra characters
corpus_text = re.sub(r"[[0-9]*\]", " ", corpus_text)
#remove the extra spaces between words
corpus_text = re.sub(r"\s+", " ", corpus_text)
#convert all letters to lowercase
corpus_text = corpus_text.lower()
```

```
<ipython-input-22-8406d6671d69>:2: FutureWarning: Possible nested set at position 1
  corpus_text = re.sub(r"[[0-9]*\]", " ", corpus_text)
```

Figure 11.2 cleaning data

2.6.4 TOKENIZING THE SENTENCES AND INVOKING THE WORD2VEC

```
#tokenize the text to list of sentences
tokenized_sentence = nltk.sent_tokenize(corpus_text)
#tokenize the list of sentences to list of words
tokenized_words = [nltk.word_tokenize(sentence) for sentence in tokenized_sentence]
#define the english stopwords
stop_words = stopwords.words('english')
#remove the stop words from the text
for i, _ in enumerate(tokenized_words):
    tokenized_words[i] = [word for word in tokenized_words[i] if word not in stop_words]
```

```
#invoke the Word2Vec with the tokenized words as argument
model = Word2Vec(tokenized_words, min_count=4)
```

Figure 11.3 tokenizing the sentences

2.6.5 RETURNING THE LIST OF WORDS

```
#invoke the Word2Vec with the tokenized words as argument
model = Word2Vec(tokenized_words, min_count=4)
```

```
#return the list of words learned
learned_words = model.wv.key_to_index.keys()
#print the learned words
print(learned_words)
```

```
dict_keys([',', "'s", 'good', '--', 'like', 'much', 'film', 'movie', 'wit', 'first', 'us', 'movies', '
'familiar', 'hard', "'", 'little', 'bartlett', 'amounts', 'throw', 'sense', 'distort', 'publishing', '
'martial-arts', 'deceit', 'heroes', 'kong', 'beyond', 'extravagant', 'frothing', 'gambles', 'theater',
e', "n't", 'house', 'reason', 'apart', 'dialogue', 'sweet', 'high', 'betrayal', 'ways', 'something', '
e', 'snow', 'runaway', 'easily', 'feel', 'amuses', 'serve', 'realization', 'schnitzler', 'terror', 'dr
ter', '.n', 'glacial', 'winning', 'youth', 'escapism', 'recommend', 'flick', 'cartoon', 'dream', 'mode
y', 'intrigue', 'whatever', 'gag', 'defend', 'gander', 'paralyzed', 'nothing', 'well-constructed', 'su
'actually', 'sitting', 'shakespearean', 'arthur', 'time', 'opportunities', 'title', 'rather', 'say', "
zzily', 'indication', 'party', 'mainland', 'ultimately', 'self-indulgent', 'grenade', '7', 'mile', 'ex
oing', 'considers', 'inoffensive', 'sometimes', 'performances', 'show', 'vaudeville', 'structure', 'ad
'boilerplate', 'times', 'watching', 'poetry', 'offering', 'sounding', 'make', 'gorgeous', 'cliches', '
e', 'age', '.na', 'hicks', 'juicy', 'forces', 'positively', 'ethnography', 'suspect', 'moonlight', 'ma
alkers', 'norris', '...', 'ends', 'becomes', 'couples', 'year', 'remakes', 'less', 'epic', 'woman', 'n
'leave', 'followed', 'mess', 'every', 'demonstrating'])
```

Figure 11.4 returning the list of words

2.6.6 CHECKING THE VECTOR REPRESENTATION

```
#check the vector representation for the word 'AI'
model.wv['film']
```

```
array([ 2.35352363e-03, -2.81072990e-03, -1.08303269e-03,  5.87057602e-03,
       -8.91270756e-04, -9.64859128e-03,  1.16013046e-02,  1.28138382e-02,
       -1.20707238e-02, -2.20962800e-03,  3.65762576e-03, -7.11870519e-03,
       -2.36910209e-03,  1.36925802e-02, -6.82463264e-03, -1.71453191e-03,
        1.01165781e-02, -6.82181958e-03, -8.23860522e-04, -2.52547953e-02,
       -1.64487958e-03,  1.49926753e-03,  1.15697142e-02, -3.11275152e-03,
        1.90599298e-03,  5.33285365e-03, -5.00236172e-03, -1.03713721e-02,
       -5.58876479e-03,  9.36546363e-04,  1.30549334e-02,  1.42999431e-02,
        2.77837389e-03,  1.60842831e-03, -1.17535526e-02,  1.20575577e-02,
       -4.82641568e-04, -1.10793207e-02, -9.56834853e-03, -1.82196219e-02,
        7.75873102e-03, -1.47306304e-02,  3.30149289e-03, -6.37816684e-03,
        8.25746451e-03,  1.34674571e-02, -8.87500029e-03, -7.75056658e-03,
       -4.05738479e-04,  2.78152549e-03,  6.00611791e-03, -9.89194959e-03,
       -3.01944651e-03,  2.40492844e-03, -1.02210073e-02,  2.85248127e-04,
        2.07268773e-03, -5.74031333e-03, -1.61896814e-02, -1.75576145e-03,
        8.87436792e-04, -4.83086507e-04,  7.40679912e-03, -2.10557273e-03,
       -1.23811541e-02,  8.87045544e-03, -2.50185071e-03,  4.87761525e-03,
       -1.34105217e-02,  3.50670842e-03, -7.07589230e-03,  1.08943190e-02,
        1.36740168e-03,  1.01850674e-06,  1.54147530e-02, -2.02824385e-03,
        6.33368874e-03,  5.90866292e-03, -1.03679067e-02,  8.95957742e-03,
        5.04079275e-03, -2.71534617e-03, -1.47931585e-02,  1.45311598e-02,
       -7.74719287e-03, -5.63828787e-03,  1.50307259e-02,  4.50519798e-03,
       -2.00458438e-04, -2.76491744e-03,  8.47306009e-03,  9.00633261e-03,
       -2.95364950e-03,  7.03003211e-03,  1.40681183e-02,  9.91589762e-03,
        1.50467698e-02, -9.28675849e-03, -2.33322079e-03,  5.69652487e-03],
      dtype=float32)
```

Figure 11.5 checking the vector reprensentation

2.6.7 CHECKING THE LIST OF SIMILAR WORDS

```
#list the similar words to AI
model.wv.most_similar('movie')
```

```
[(',', 0.6716362833976746),
 ('terror', 0.579094409942627),
 ("'s", 0.5755414366722107),
 ('movies', 0.5703694820404053),
 ('glacial', 0.5598156452178955),
 ('perspective', 0.5557324886322021),
 ('actors', 0.5455741286277771),
 ('us', 0.544561505317688),
 ('film', 0.5435364246368408),
 ('going', 0.5426031351089478)]
```

Figure 11.6 checking the list of similar words

2.7 COSINE SIMILARITY

Text Similarity has to determine how the two text documents close to each other in terms of their context or meaning. There are various text similarity metric exist such as Cosine similarity**,** Euclidean distance and Jaccard Similarity. All these metrics have their own specification to measure the similarity between two queries.

In this tutorial, you will discover the Cosine similarity metric with example. You will also get to understand the mathematics behind the cosine similarity metric with example. Please refer to this tutorial to explore the Jaccard Similarity**.**

Cosine similarity is one of the metric to measure the text-similarity between two documents irrespective of their size in Natural language Processing. A word is represented into a vector form. The text documents are represented in n-dimensional vector space.

Mathematically, Cosine similarity metric measures the cosine of the angle between two n-dimensional vectors projected in a multi-dimensional space. The Cosine similarity of two documents will range from 0 to 1. If the Cosine similarity score is 1, it means two vectors have the same orientation. The value closer to 0 indicates that the two documents have less similarity.

The mathematical equation of Cosine similarity between two non-zero vectors is:

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

Cosine similarity is a metric used to determine how similar the documents are irrespective of their size.

Mathematically, it measures the cosine of the angle between two vectors projected in a multi-dimensional space. In this context, the two vectors I am talking about are arrays containing the word counts of two documents.

When plotted on a multi-dimensional space, where each dimension corresponds to a word in the document, the cosine similarity captures the orientation (the angle) of the documents and not the magnitude. If you want the magnitude, compute the Euclidean distance instead.

The cosine similarity is advantageous because even if the two similar documents are far apart by the Euclidean distance because of the size (like, the word 'cricket' appeared 50 times in one document and 10 times in another) they could still have a smaller angle between them. Smaller the angle, higher the similarity.

2.7.1 CALCULATING THE COSINE SIMILARITY BETWEEN TWO TEXT DATA

2.7.1.1 Reading the data

```
doc_1 = "Good movie with great story"
doc_2 = 'n'.join(text[:10]['Phrase'])
```

```
data = [doc_1, doc_2]
```

Figure 12.1 reading the data

2.7.1.2 Clear visualization of vectorize data along with tokens.

```
from sklearn.feature_extraction.text import TfidfVectorizer

Tfidf_vect = TfidfVectorizer()
vector_matrix = Tfidf_vect.fit_transform(data)

tokens = Tfidf_vect.get_feature_names()
create_dataframe(vector_matrix.toarray(),tokens)
```

	adage	also	amounts	amuses	but	demonstrating	escapades	for	gander	good	...
doc_1	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.355200	...
doc_2	0.262168	0.052434	0.052434	0.052434	0.052434	0.209734	0.157301	0.314602	0.052434	0.223842	...

2 rows × 32 columns

Figure 12.2 vectorizing data along with tokens

2.7.1.3 Cosine Similarity of two text data

```
cosine_similarity_matrix = cosine_similarity(vector_matrix)
create_dataframe(cosine_similarity_matrix,['doc_1','doc_2'])
```

	doc_1	doc_2
doc_1	1.00000	0.09276
doc_2	0.09276	1.00000

Figure 12.3 cosine similarity of text data

3. CONCLUSION

NLP based totally chatbots can assist enhance your business processes and raise consumer experience to the following stage even as additionally growing overall boom and profitability. It offers technological benefits to stay competitive inside the market-saving time, effort and charges that in addition leads to increased purchaser pleasure and extended engagements in your business.

Text processing is a method used under the NLP to clean the text and prepare it for the model building.Text preprocessing includes many techniques which help to clean the raw data transfer into model building.

Extracting data from PDF is the underlying object model and PyPDF2 is a powerful library that enables you to access it.

Exploratory Data Analysis (EDA) is a powerful device that could spotlight troubles to be addressed, result in insights, and recommend styles in the data. The EDA method for data construction and analysis is massively practiced in the modern world of data science and also in AI.

LSTM outperforms the other models when we want our model to learn from long term dependencies. LSTM's ability to forget, remember and update the information pushes it one step ahead of RNNs.

BERT is certainly a breakthrough inside the use of Machine Learning for Natural Language Processing. The truth that it's approachable and allows fast fine-tuning will likely permit a extensive range of practical programs in the future. In this precis, we explain the primary thoughts of the paper even as not drowning in excessive technical info.

Word embeddings are considered to be one of the successful applications of unsupervised learning at present. They do not require any annotated corpora. Embeddings use a lower-dimensional space while preserving semantic relationships.

Finding an effective and efficient way to calculate text similarity is a critical problem in text mining and information retrieval. One of the most popular similarity measures is cosine similarity, which is based on Euclidean distance. It has been shown useful in many applications, however, cosine similarity is not ideal. Euclidean distance is based on L2 norm and does not work well with high-dimensional data.

4.APPENDICES

4.1 CONTENT EXTRACTION FROM PDF

PDF or Portable Document File layout is one of the most common report formats in use these days. It is extensively used across companies, in authorities, places of work, healthcare and other industries. As a result, there may be a massive body of unstructured statistics that exists in PDF layout and to extract and analyse these records to generate meaningful insights is a commonplace challenge among facts scientists.

For a financial institution and lately came across a scenario where it had to extract records from a big extent of PDF forms. While there is a good frame of labor to be had to describe easy text extraction from PDF documents, so, a comprehensive manual to extract statistics from PDF bureaucracy.

There are numerous Python libraries dedicated to working with PDF files, some more famous than the others. I can use PyPDF2 for the motive of this newsletter. PyPDF2 is a Pure-Python library built as a PDF toolkit. Being Pure-Python, it is able to run on any Python platform with none dependencies or external libraries.

List of packages used for extraction text from pdf files.

- PyPDF2
- Tika
- Textract
- PyMuPDF
- PDFtotext
- PDFminer
- Tabula

4.1.1 PyPDF2

PyPDF2 is a pure-Python package deal that may be used for plenty one-of-a-kind forms of PDF operations. PyPDF2 can be used to perform the subsequent tasks.

- Extract document statistics from a PDF in Python

Figure 13.1 Extract document statistics from a PDF in Python

- Copying pages and creating new pdf

```
#copying  pages and creating new pdf
```

```
[7]: import PyPDF2
pdf1File = open('Data/independence.pdf', 'rb')
pdf2File = open('Data/themecelebrate.pdf', 'rb')
pdf1Reader = PyPDF2.PdfFileReader(pdf1File)
pdf2Reader = PyPDF2.PdfFileReader(pdf2File)
pdfWriter = PyPDF2.PdfFileWriter()
for pageNum in range(pdf1Reader.numPages):
    pageObj = pdf1Reader.getPage(pageNum)
    pdfWriter.addPage(pageObj)
for pageNum in range(pdf2Reader.numPages):
    pageObj = pdf2Reader.getPage(pageNum)
    pdfWriter.addPage(pageObj)
pdfOutputFile = open('Data/random.pdf', 'wb')
pdfWriter.write(pdfOutputFile)
pdfOutputFile.close()
pdf1File.close()
pdf2File.close()
```

Figure 13.2 Copying pages and creating new pdf

- Rotating pages

```
#rotating pages
```

```
[17]: import PyPDF2
minutesFile = open('Data/independence.pdf', 'rb')
pdfReader = PyPDF2.PdfFileReader(minutesFile)
page = pdfReader.getPage(0)
page.rotateClockwise(90)
pdfWriter = PyPDF2.PdfFileWriter()
pdfWriter.addPage(page)
resultPdfFile = open('Data/rotatedPage.pdf', 'wb')
pdfWriter.write(resultPdfFile)
resultPdfFile.close()
minutesFile.close()
```

Figure 13.3 Roating pages

- Splitting PDF's

```
#splitting pdf
```

```
[9]: from PyPDF2 import PdfFileWriter, PdfFileReader
input_pdf = PdfFileReader("Data/independence.pdf")
output = PdfFileWriter()
output.addPage(input_pdf.getPage(0))
with open("Data/splitted.pdf", "wb") as output_stream:
    output.write(output_stream)
```

Figure 13.4 Splitting PDF

- Merging PDf's

```
#mergeing pdf
```

```
[10]: from PyPDF2 import PdfFileReader, PdfFileMerger
pdf_file1 = PdfFileReader("Data/independence.pdf")
pdf_file2 = PdfFileReader("Data/themecelebrate.pdf")
output = PdfFileMerger()
output.append(pdf_file1)
output.append(pdf_file2)

with open("Data/merged.pdf", "wb") as output_stream:
    output.write(output_stream)
```

Figure 13.5 Merging PDF

- Adding watermarks

```
#adding watermarks
```

```
[11]: import PyPDF2
pdf_file = "Data/watermark.pdf"
watermark = "Data/techmahindrawatermark.pdf"
merged_file = "Data/watermarkmerged.pdf"
input_file = open(pdf_file,'rb')
input_pdf = PyPDF2.PdfFileReader(input_file)
watermark_file = open(watermark,'rb')
watermark_pdf = PyPDF2.PdfFileReader(watermark_file)
pdf_page = input_pdf.getPage(0)
watermark_page = watermark_pdf.getPage(0)
pdf_page.mergePage(watermark_page)
output = PyPDF2.PdfFileWriter()
output.addPage(pdf_page)
merged_file = open(merged_file,'wb')
output.write(merged_file)
merged_file.close()
watermark_file.close()
input_file.close()
```

Figure 13.6 Adding watermark

- Encrypt a PDF

```
#Encrypt a PDF
```

```
[12]: from PyPDF2 import PdfFileWriter, PdfFileReader
out = PdfFileWriter()
file = PdfFileReader("Data/splitted.pdf")
num = file.numPages
for idx in range(num):
    out.addPage(page)
    password = "sowmya"
    out.encrypt(password)
with open("Data/encrypt_random.pdf", "wb") as f:
    out.write(f)
```

Figure 13.7 Encrypt a PDF

Cons of using the PyPDF2 package deal:

- This package extracts textual content however does no longer keep the shape of the text in the authentic PDF.
- Unnecessary areas and newlines are included within the extracted text.
- It no longer holds the table structure.

4.1.2 Tika

Tika is a Java-based package deal. Tika-Python is Python binding to the Apache Tika REST services which allows Tika to be called natively in python language. To use the Tika package deal in python, we want to have java mounted on your gadget. When you run the code for the first time, it will provoke the reference to the Java server. This results in not on time extraction of text from PDF the use of the Tika package if the code is strolling for the first time inside the system.

Below are a few extra obligations finished whilst extracting texts from PDF.

- Extract contents of the PDF report
- Extract Meta-Data of PDF report
- Extract keys (metadata and content material for dictionary)
- To know the Tika server fame

Some main hazards of using the Tika package deal are:

- Needs java hooked up
- Java server connection is time-consuming
- Does not keep desk shape

4.1.3 Textract

While several packages exist for extracting content material from numerous formats of files on their own, the Textract bundle gives an unmarried interface for extracting content from any sort of report, with none beside the point markup.

Textract is used to extract textual content from PDF files as well as different report formats. The different document formats consist of csv, document, eml, epub, json, jpg, mp3, msg, xls, and so forth.

The most noteworthy point of the use of the Textract package deal is that it extracts statistics from files in byte format. To convert byte data into a string we want to use different python packages for decoding like codecs.

4.1.4 PyMuPDF

PyMuPDF is a python binding for MuPDF which is a lightweight PDF viewer. PyMuPDF isn't entirely python based. This package deal is thought for both, its pinnacle performance and excessive rendering excellent.

With PyMuPDF, we can get right of entry to files with extensions like *.Pdf, *.Xps, *.Oxps, *.Epub, *.Cbz or *.Fb2 out of your Python scripts. Several popular image codecs are supported as nicely, inclusive of multi-web page TIFF pics.

PyMuPDF extracts the facts of multipage documents additionally. It gives us the privilege to extract records for a selected page in case you enter the web page number.

```
#PyMuPDF

[13]: import fitz

[14]: print(fitz.__doc__)
      doc = fitz.open('Data/independence.pdf')

PyMuPDF 1.18.16: Python bindings for the MuPDF 1.18.0 library.
Version date: 2021-08-05 00:00:01.
Built for Python 3.6 on linux (64-bit).
```

Figure 14 PyMuPDF

4.1.5 PDFtotext

PDFtotext is a python-based total bundle that can be used to extract texts from PDF documents. As the name indicates, it supports handiest PDF documents whilst different record formats are not supported.

The record is extracted within the form of an item. The shape of the PDF is preserved.

4.1.6 PDFminer

This is but any other purely python-based package that is used to extract best PDF files. It can also convert PDF documents into other report codecs like HTML/XML. There are numerous variations of PDFminer and the cutting-edge model is compatible with python three.6 and above.

PDFminer provides its provider in the shape of an API request. Thus, the results acquired from this package take barely greater time than different merely python-based programs.

There are several parameters for use at the same time as calling this package deal. The complete description of the parameters can be observed right here.

The code used to extract text from PDF using PDFminer package deal is tedious and longer as compared to easy code used for different applications that are given underneath alongside Input PDF and output extracted textual content.

```
#PDFminer
```

```
[15]: from pdfminer.pdfinterp import PDFResourceManager, PDFPageInterpreter
from pdfminer.converter import TextConverter
from pdfminer.layout import LAParams
from pdfminer.pdfpage import PDFPage
from io import StringIO
path='Data/independence.pdf'
def convert_pdf_to_txt():
    rsrcmgr = PDFResourceManager()
    retstr = StringIO()
    codec = 'utf-8'
    laparams = LAParams()
    device = TextConverter(rsrcmgr,retstr,codec=codec,laparams=laparams)
    fp = open(path, 'rb')
    interpreter = PDFPageInterpreter(rsrcmgr, device)
    password = ""
    maxpages = 0
    caching = True
    pagenos=set()
    for page in PDFPage.get_pages(fp, pagenos, maxpages=maxpages, password=password,caching=caching, check_extractable=True):
        interpreter.process_page(page)
    text = retstr.getvalue()
    fp.close()
    device.close()
    retstr.close()
    return text
pdf_miner_text = convert_pdf_to_txt()
```

Figure 15 PDFminer

4.1.7 Tabula

This java-based package deal is particularly used to study tables in a PDF. It is an easy python wrapper for tabula-java.

The facts extraction is saved inside the python DataFrame in python which later can be converted into csv, tsv, excel, or json report layout.

5. REFERENCES

1. Rhea Sukthanker, Soujanya Poria, Erik Cambria, Ramkumar Thirunavukarasu (July 2020) *Anaphora and coreference resolution: A review* https://arxiv.org/abs/1805.11824
2. [Sharid Loaiciga, Liane Guillou, Christian Hardmeier (September 2017) What is it? Disambiguating the different readings of the pronoun 'it' https://www.aclweb.org/anthology/D17-1137/
3. Stanford lecture (CS224n) by Christopher Manning (2019) https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1162/handouts/cs224n-lecture10-coreference.pdf
4. Coreference *Wikipedia* https://en.wikipedia.org/wiki/Coreference
5. Bader Aljaber; Nicola Stokes; James Bailey; Jian Pei. "Document clustering of scientific texts using citation contexts," Information Retrieval, V.13, N.2, 101-131, DOI: 10.1007/s10791-009-9108-x, 2009.
6. Constans, Pere. "A Simple Extraction Procedure for Bibliographical Author Field," Word Journal OF The International Linguistic Association, February, 2009.
7. Gupta, D.; Morris, B.; Catapano, T.; Sautter, G. "A New Approach towards Bibliographic Reference Identification, Parsing and Inline Citation Matching," In Proceedings of IC3. 2009, 93-102.
8. Hua Yang; Norikazu Onda; Massaki Kashimura; Shinji Ozawa. Extraction of bibliography information based on image of book cover. In Proceedings of the 10th International Conference on Image Analysis and Processing IEEE Computer Society Washington, DC, USA, 1999.
9. Ohta, M., Yakushi, T, Takasu, A. "Bibliographic Element Extraction from Scanned Documents Using Conditional Random Fields" In Proceedings of ICDIM, 2008, 99-104.
10. "PDF Reference", Adobe system Incorporated.
11. "Acrobat 4: Adobe's bid to make it more than a viewer" Walter, M. Seybold report on Internet Publishing. Vol. 3, no. 7.
12. . Nagy G., Seth S. and Viswanathan M., "A prototype document image analysis system for technical Journals", IEEE Computer, vol. 25, pp. 10-22, 1992.

13. Dengel, A. “Initial learning of document structures”, the Second International Conference on Document Analysis and Recognition (ICDAR),1993 pg 86-90.
14. Dengel, A., Dubiel, F.“Clustering and classification of document structure? A machine
15. learning approach.” ICDAR 1995, pp 587-591.
16. Kise, K. Incremental acquisition of knowledge about layout structures from examples of documents. ICDAR 1993, (pp.668-671)
17. Anscombe, F. (1973), Graphs in Statistical Analysis, The American Statistician, pp. 195-199.
18. Anscombe, F. And Tukey, J. W. (1963), The Examination and Analysis of Residuals, Technometrics, pp. 141-160.
19. Barnett and Lewis (1994), Outliers in Statistical Data, 3rd. Ed., John Wiley and Sons.
20. Birnbaum, Z. W. And Saunders, S. C. (1958), A Statistical Model for Life-Length of Materials, Journal of the American Statistical Association, fifty three(281), pp. 151-160.
21. Bloomfield, Peter (1976), Fourier Analysis of Time Series, John Wiley and Sons.
22. Box, G. E. P. And Cox, D. R. (1964), An Analysis of Transformations, Journal of the Royal Statistical Society, pp. 211-243, discussion pp. 244-252.
23. Box, G. E. P., Hunter, W. G., and Hunter, J. S. (1978), Statistics for Experimenters: An Introduction to Design, Data Analysis, and Model Building, John Wiley and Sons.
24. Box, G. E. P., and Jenkins, G. (1976), Time Series Analysis: Forecasting and Control, Holden-Day.
25. Bradley, (1968). Distribution-Free Statistical Tests, Chapter 12.
26. Brown, M. B. And Forsythe, A. B. (1974), Journal of the American Statistical Association, sixty nine, pp. 364-367.