SENTIMENT ANALYSIS USING BERT MODEL

A Main project thesis submitted in partial fulfilment of requirements for the award of degree for VIII semester

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

by

POPURI TRIVENI	(Reg No:19131A05H9)
METTU AMARNADH REDDY	(Reg No:19131A05D2)
ROKALLA SRUTHI	(Reg No:19131A05K6)
PANGA YOGESH	(Reg No:19131A05G9)

Under the esteemed guidance of

Mrs. G. VANI

Assistant Professor

Department of Computer Science and Engineering

GAYATRI VIDYA PARISHAD COLLEGE OF ENGINEERING (AUTONOMOUS)

(Affiliated to JNTU-K, Kakinada)

VISAKHAPATNAM

2022 – 2023

GAYATRI VIDYA PARISHAD COLLEGE OF ENGINEERING (Autonomous)
Approved by AICTE, New Delhi and Affiliated to JNTU-Kakinada
Re-accredited by NAAC with "A" Grade with a CGPA of 3.47/4.00
Madhurawada, Visakhapatnam - 530 048.

CERTIFICATE

This is to certify that the main project entitled "**Sentiment Analysis using BERT model**" being submitted by

POPURI TRIVENI	**(19131A05H9)**
METTU AMARNADH REDDY	**(19131A05D2)**
ROKALLA SRUTHI	**(19131A05K6)**
PANGA YOGESH	**(19131A05G9)**

in partial fulfilment for the award of the degree "Bachelor of Technology" in Computer Science and Engineering to the Jawaharlal Nehru Technological University, Kakinada is the record of bonafide work done under my guidance and supervision during VIII semester in the academic year 2022-2023.

Project Guide	**Head of the Department**
Mrs. G. Vani	Dr. D.N.D. Harini
Assistant Professor	Associate Professor & HOD
Department of CSE	Department of CSE
GVPCE(A)	GVPCE(A)

DECLARATION

We hereby declare that this project entitled “**Sentiment Analysis using BERT model”** is a bonafide work done by us and submitted to **Department of Computer Science and Engineering, GVP College of Engineering (Autonomous), Visakhapatnam,** in partial fulfilment for the award of the degree of B.Tech is of our own and it is not submitted to any other university or has been published any time before.

PLACE: VISAKHAPATNAM	P. TRIVENI (19131A05H9)
DATE:	M. AMARNADH REDDY (19131A05D2)
	R. SRUTHI (19131A05K6)
	P. YOGESH (19131A05G9)

ACKNOWLEDGEMENT

We would like to express our deep sense of gratitude to our esteemed institute **Gayatri Vidya Parishad College of Engineering (Autonomous),** which has provided us an opportunity to fulfil our cherished desire.

We express our sincere thanks to our Principal **Dr. A.B. KOTESWARA RAO, Gayatri Vidya Parishad College of Engineering (Autonomous)** for his encouragement to us during this project, giving us a chance to explore and learn in the form of main project.

We express our deep sense of Gratitude to **Dr. D.N.D. HARINI, Associate Professor and Head of the Department of Computer science and Engineering, Gayatri Vidya Parishad College of Engineering (Autonomous)** for giving us an opportunity to do the project in college.

We express our profound gratitude and our deep indebtedness to our guide **Mrs. G. Vani, Assistant Professor, Department of Computer Science and Engineering**, whose valuable suggestions, guidance and comprehensive assistance helped us a lot in realizing my present project.

We also thank our coordinator, **Dr. CH. SITA KUMARI, Associate Professor, Department of Computer Science and Engineering**, for the kind suggestions and guidance for the successful completion of our project work.

POPURI TRIVENI	**(19131A05H9)**
METTU AMARNADH REDDY	**(19131A05D2)**
ROKALLA SRUTHI	**(19131A05K6)**
PANGA YOGESH	**(19131A05G9)**

ABSTRACT

Sentiment analysis is a natural language processing (NLP) task that involves analysing text data to determine the sentiment expressed in it. With the increasing use of social media and online communication, sentiment analysis has become an essential tool for businesses to understand customer feedback and make data-driven decisions. In recent years, deep learning models, such as the Bidirectional Encoder Representations from Transformers (BERT) model, have shown remarkable performance in sentiment analysis tasks. In this project, we propose to use the BERT model for sentiment analysis on a dataset of amazon reviews of books. We will fine-tune the pre-trained BERT model on the dataset of amazon reviews of books, which involves adapting the pre-trained model to the specific domain of customer reviews. We will evaluate the performance of the BERT model using metrics such as accuracy, precision, recall, and F1-score. The outcome of this project will be a sentiment analysis model that can accurately classify the sentiment of amazon reviews, which can be used by businesses to improve their products and services.

Keywords: Sentiment Analysis, BERT model, performance, accuracy, precision, recall, F1-score.

TABLE OF CONTENTS

1.INTRODUCTION

Sentiment analysis is one of the prevalent tasks in natural language processing (NLP), which involves identifying the sentiment of a given piece of text as positive, negative or neutral. With the fast development of social media and E-commerce platforms in recent years, sentiment analysis of online reviews aims at mining users' opinions towards social events or products. Knowing how users feel or think about a certain brand, product, idea or topic is a valuable source of information for companies, organizations and researchers, but it can be a challenging task. With effective detection of sentiment polarity expressed in these reviews, stakeholders have the opportunity to guide public opinion towards their favourable direction, or improve products for better user experience.

There is a large body of existing work in the field of sentiment analysis, with new methods and techniques being developed and published regularly. Some machine learning-based methods use various algorithms, such as Naive Bayes, SVM, Random Forest and deep learning to train a model on a labelled dataset of text samples with corresponding sentiment labels. These models can be used to classify new text samples and predict the sentiment expressed in them and some Lexicon-based methods use a pre-defined list of words and their corresponding sentiment scores, called a lexicon, to classify text samples.

With the increasing need for higher accuracy, using deep learning methods for sentiment classification is becoming a key topic for social media related research. These methods use deep neural networks, such as CNNs, RNNs and LSTMs, to classify text samples. They have shown to be very effective in natural language processing tasks, including sentiment analysis.

In this project, we will be implementing a sentiment analysis model using BERT(Bidirectional Encoder Representations from Transformers) model, fine-tuned on a dataset of Amazon book reviews, and evaluating its performance. The key innovation of BERT is the use of a transformer architecture, which is a type of neural network that allows bidirectional training, enabling the model to capture context from both directions of a sequence. It is a transformer-based model that can be fine-tuned for a wide range of natural language processing tasks, such as sentiment analysis, question answering, and language translation.

2.SOFTWARE REQUIREMENT ANALYSIS

2.1 SOFTEWARE DESCRIPTION

2.1.1 GOOGLE COLAB

Google Colab, short for Google Colaboratory, is a free cloud-based platform provided by Google that enables users to run and execute code using Python and other languages in a Jupyter notebook environment. It provides access to powerful computing resources, including CPUs, GPUs, and TPUs, without the need for users to install and configure hardware and software on their local machines. This makes it an excellent tool for machine learning and data analysis tasks that require high computational resources.

Google Colab allows users to import and export data, collaborate with others in real-time, and share their notebooks easily. It also supports various libraries, including TensorFlow, Keras, PyTorch, and many others, which can be installed and used seamlessly within the notebook environment. Users can run their code on a virtual machine instance, which can be customized with various specifications such as CPU, GPU, and RAM.

Google Colab has become increasingly popular among researchers, students, and practitioners in the machine learning community due to its ease of use, accessibility, and powerful computing resources. It has democratized access to computing resources, enabling anyone with an internet connection to work on machine learning and data analysis projects.

Google Colab provides tons of exciting features that any modern IDE offers, and much more. Some of the most exciting features are listed below.

- Interactive tutorials to learn machine learning and neural networks.
- Write and execute Python 3 code without having a local setup.
- Execute terminal commands from the Notebook.
- Import datasets from external sources such as Kaggle.
- Save your Notebooks to Google Drive.
- Import Notebooks from Google Drive.

2.1.2 PYCHARM

PyCharm is an Integrated Development Environment (IDE) for Python programming language. It is developed by JetBrains and is available as a community edition and a professional edition. PyCharm provides developers with a variety of features and tools that make it easy to write, debug, and test Python code.

Some of the key features of PyCharm include code highlighting and completion, debugging tools, version control integration, database tools, and support for web development frameworks like Django and Flask. PyCharm also includes a built-in test runner, code profiling tools, and integration with popular scientific computing libraries like NumPy and Pandas.

PyCharm is available on Windows, macOS, and Linux operating systems. It offers a free community edition that provides basic features for Python development, while the professional edition comes with additional features and support for other programming languages.PyCharm is widely used by Python developers and is considered to be one of the best IDEs available for Python programming. Its powerful features, ease of use, and integration with other tools make it an ideal choice for both beginners and experienced developers.

2.2 NUMPY

2.2.1 INTRODUCTION

NumPy is a Python package. It stands for 'Numerical Python'. It is a library consisting of multidimensional array objects and a collection of routines for processing of array. Numeric, the ancestor of NumPy, was developed by Jim Hugunin. Another package Numarray was also developed, having some additional functionalities. In 2005, Travis Oliphant created NumPy package by incorporating the features of Numarray into Numeric package. There are many contributors to this open-source project.

2.2.2 OPERATIONS USING NUMPY

Using NumPy, a developer can perform the following operations –

- Mathematical and logical operations on arrays.
- Fourier transforms and routines for shape manipulation.
- Operations related to linear algebra. NumPy has in-built functions for linear algebra and random number generation.

2.3 PANDAS

2.3.1 INTRODUCTION

Pandas is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures. Prior to Pandas, Python was majorly used for data munging and preparation. It had very little contribution towards data analysis. Pandas solved this problem. Using Pandas, five typical steps can be accomplished in the processing and analysis of data, regardless of the origin of data — load, prepare, manipulate, model, and analyse. Python with Pandas is used in a wide range of fields inc.

2.3.2 OPERATIONS USING PANDAS

Using PANDAS, the following functions can be performed.

- Loading data this is read a CSV file.
- Column Insertion and deletion.
- Data selection and sorting.
- Column and Row renaming.
- Handling missing values and duplicated data.
- Data Exploration and Visualisation.

2.4 SEABORN

2.4.1 INTRODUCTION

Seaborn is an amazing visualization library for statistical graphics plotting in Python. It provides beautiful default styles and colour palettes to make statistical plots more attractive. It is built on the top of matplotlib library and also closely integrated to the data structures from pandas. Seaborn aims to make visualization the central part of exploring and understanding data. It provides dataset-oriented APIs, so that switching between different visual representations for same variables to better understand the dataset.

2.4.2 OPERATIONS USING SEABORN

- Used for data visualization and finding out the patterns.

2.5 MATPLOTLIB

2.5.1 INTRODUCTION

Matplotlib is one of the most popular Python packages used for data visualization. It is a cross-platform library for making 2D plots from data in arrays. Matplotlib is written in Python and makes use of numpy, the numerical mathematics extension of Python. Matplotlib along with NumPy can be considered as the open source of MATPLOTLIB.

Syntax: import matplotlib.pyplot as plt

Pyplot is a state-based interface to a **Matplotlib** module which provides a MATLAB-interface.

2.5.2 OPERATIONS USING MATPLOTLIB

Matplotlib is a popular Python library used for data visualization. Here are some common operations that can be performed using Matplotlib:

- Creating plots: To create a plot using Matplotlib, we can import the library and use the "plot" function to plot a graph.
- Customizing plots: Matplotlib provides many customization options to help us create professional-looking plots. We can add a title, labels for the x and y axes, and change the line style and colour.
- Plotting multiple lines: We can plot multiple lines on the same graph by calling the "plot" function multiple times.
- Creating different types of plots: Matplotlib supports a wide variety of plot types including scatter plots, bar graphs, and histograms. We can use the corresponding functions like "scatter", "bar", and "hist" to create these types of plots.

2.6 TORCH

2.6.1 INTRODUCTION

In Python, the torch package is a popular open-source machine learning library developed by Facebook's AI Research team. It is designed to work with Python. The torch package provides a variety of tools and modules for building and training neural networks, such as tensor operations, automatic differentiation, and a variety of optimization algorithms. The torch package is based on PyTorch, which is a Python-based deep learning framework that provides a tensor computation library and a dynamic computation graph that allows for efficient and flexible building of neural networks. The torch package in Python provides a similar set of features as PyTorch, including a wide range of neural network architectures, loss functions, and optimization algorithms. It also provides a variety of pre-trained models, including state-of-the-art models such as BERT and GPT-2, that can be used for various natural language processing tasks.

2.6.2 OPERATIONS USING TORCH

The torch package in Python provides a variety of tensor operations that can be used for building and training neural networks. Some of the key operations provided by the torch package are:

- **Tensor creation:** The torch package provides functions for creating tensors of different shapes and data types, including 1D, 2D, and higher-dimensional tensors.
- **Tensor indexing and slicing:** The torch package provides a variety of indexing and slicing operations that allow you to access and modify specific elements of a tensor. You can use indexing and slicing to extract rows, columns, and sub-tensors from a larger tensor.
- **Tensor arithmetic:** The torch package provides a variety of arithmetic operations that can be applied to tensors, including addition, subtraction, multiplication, and division.
- **Tensor operations:** The torch package provides a variety of tensor operations, including matrix multiplication, dot product, element-wise product, and other linear algebra operations. These operations are important for building neural networks and can be used to perform operations such as convolution and pooling.
- **Tensor broadcasting:** The torch package provides broadcasting, which allows for the automatic alignment of tensors of different shapes and sizes for element-wise operations. This makes it easier to perform arithmetic operations on tensors with different shapes and sizes.
- **Autograd:** The torch package provides automatic differentiation through the use of autograd. This allows for the computation of gradients of tensors with respect to some target variable. This is a key feature for training neural networks using backpropagation.

Overall, the torch package provides a comprehensive set of tensor operations that are essential for building and training neural networks. These operations are efficient and flexible, making it easier to build complex models in Python.

2.7 GC (GARBAGE COLLECTION)

2.7.1 INTRODUCTION

The gc module in Python provides automatic garbage collection features. It allows you to control and manage the behaviour of the garbage collector, which is responsible for freeing up memory that is no longer needed by the program. The gc module provides a powerful set of tools for managing memory in Python programs, and can be especially useful in situations where memory usage is a concern.

2.7.2 OPERATIONS USING GC

Using gc, the following operations can be performed.

- **Disable or enable garbage collection:** you can use the gc.disable() and gc.enable() functions to turn off and on the garbage collector, respectively.
- **Force a garbage collection:** you can use the gc.collect() function to force the garbage collector to run and free up any memory that is no longer needed.
- **Get information about the garbage collector:** you can use the gc.get_count() function to get the current number of objects tracked by the garbage collector, and the gc.get_threshold() function to get the current collection thresholds.
- **Set the collection thresholds:** you can use the gc.set_threshold() function to set the collection thresholds. This can be useful if you want to fine-tune the garbage collection behaviour to suit your specific needs.

2.8 COLLECTIONS

2.8.1 INTRODUCTION

The collections module in Python provides a set of specialized container data types that are more efficient and useful than the built-in data types. The collections module provides a useful set of tools for working with specialized container data types in Python. These containers can help you write more efficient and readable code, especially in situations where you need to perform frequent operations on large data sets.

2.8.2 OPERATIONS USING COLLECTIONS

Here are some of the container data types provided by the collections module:

- **Counter:** This container provides a dictionary-like object for counting the occurrences of elements in a list. For example, you can use a Counter object to count the frequency of each character in a string.
- **Deque**: This container provides a double-ended queue, which allows you to efficiently append and pop elements from either end of the queue.
- **defaultdict**: This container provides a dictionary-like object that automatically initializes missing keys with a default value. For example, you can use a defaultdict to count the occurrences of elements in a list without having to manually initialize the count for each element.
- **namedtuple**: This container provides a way to create named tuples, which are similar to regular tuples but have named fields. This can make it easier to work with complex data structures.
- **OrderedDict**: This container provides a dictionary-like object that remembers the order in which items were added. This can be useful in situations where the order of the items is important.

2.9 TEXTWRAP

2.9.1 INTRODUCTION

The textwrap module in Python provides a set of functions for formatting and wrapping text. The textwrap module provides a flexible set of tools for formatting and manipulating text in Python. It can be especially useful for generating output that needs to be neatly formatted and easy to read.

2.9.2 OPERATIONS USING TEXTWRAP

Here are some of the key features of the textwrap module:

- **Wrapping text:** The textwrap.wrap() function can be used to split a long string of text into a list of shorter lines. You can specify the maximum width of each line, as well as options for handling whitespace and line breaks.
- **Indenting text:** The textwrap.indent() function can be used to add a specified amount of indentation to each line of a string. This can be useful for formatting code or other types of structured text.

- **Filling text:** The textwrap.fill() function can be used to format a long string of text into a series of shorter lines that fit within a specified width. This function takes into account word boundaries and tries to avoid breaking words across lines.
- **Dedenting text:** The textwrap.dedent() function can be used to remove a consistent amount of indentation from each line of a string. This can be useful for processing indented text that needs to be aligned with the left margin.

2.10 STREAMLIT

2.10.1 INTRODUCTION

Streamlit is a popular Python library that allows developers to quickly build interactive web applications with minimal coding. It is designed to simplify the process of creating and deploying machine learning and data science projects to the web.

Some key features of Streamlit include:

- Easy to use: Streamlit provides a simple and intuitive API that makes it easy to create web applications with Python.
- Interactive widgets: Streamlit provides a range of interactive widgets that can be used to create sliders, dropdowns, checkboxes, and other user interface elements.
- Real-time updates: Streamlit automatically updates the web application in real-time as the user interacts with it. This means that any changes made by the user will be immediately reflected in the application.
- Fast deployment: Streamlit makes it easy to deploy your web application to the web. We can deploy your application on popular cloud platforms like Heroku, AWS, and Google Cloud Platform.

2.10.2 OPERATIONS USING STREAMLIT

Streamlit is a Python library that makes it easy to create interactive web applications with minimal coding.

Here are some common operations that can be performed using Streamlit:

- Building UI components: Streamlit provides a range of UI components that can be used to create interactive web applications. These include widgets like sliders, dropdowns, checkboxes, and text inputs.

- Displaying data: Streamlit makes it easy to display data in a variety of formats, including tables, charts, and maps.
- Handling user input: Streamlit allows you to handle user input easily using built-in functions like "text_input" and "checkbox". We can use these functions to get input from the user and perform actions based on their selections.
- Deploying to the web: Streamlit makes it easy to deploy your web application to the web using popular cloud platforms like Heroku, AWS, and Google Cloud Platform.

2.11 DEEP LEARNING ALGORITHM

2.11.1 INTRODUCTION

Deep learning is a subset of machine learning that uses artificial neural networks to model and solve complex problems. In deep learning, multiple layers of neural networks are stacked together to create a deep neural network that can learn and make predictions on large datasets. Deep learning has revolutionized many fields, including computer vision, natural language processing, speech recognition, and robotics. Some of the most popular deep learning models include convolutional neural networks (CNNs) for image classification, recurrent neural networks (RNNs) for natural language processing and speech recognition, and generative adversarial networks (GANs) for image and text generation.

Deep learning models typically require large amounts of labelled data for training, and can be computationally expensive to train on high-performance computing systems. However, with recent advancements in hardware and software, deep learning has become more accessible to researchers and developers, and has led to significant breakthroughs in many areas of artificial intelligence. Applications of deep learning include image and speech recognition, natural language processing, autonomous vehicles, drug discovery, recommendation systems, fraud detection, and more.

2.11.2 TRANSFORMER

A transformer is a deep learning model that adopts the mechanism of self-attention, differentially weighting the significance of each part of the input data. It is used primarily in the fields of natural language processing (NLP) and computer vision (CV). Transformer uses attention-mechanism.

ATTENTION-MECHANISM:

The attention-mechanism looks at an input sequence(input sentence) and decides at each step which other parts of the sequence are important. It sounds abstract, but let me clarify with an easy example: When reading this text, we always focus on the word you read but at the same time your mind still holds the important keywords of the text in memory in order to provide context.

For each input sequence that the encoder reads, the attention-mechanism takes into account several other inputs at the same time and decides which ones are important by attributing different weights to those inputs. The Decoder will then take as input the encoded sentence and the weights provided by the attention-mechanism and then the decoder gives us the desired output.

TRANSFORMER ARCHITECTURE:

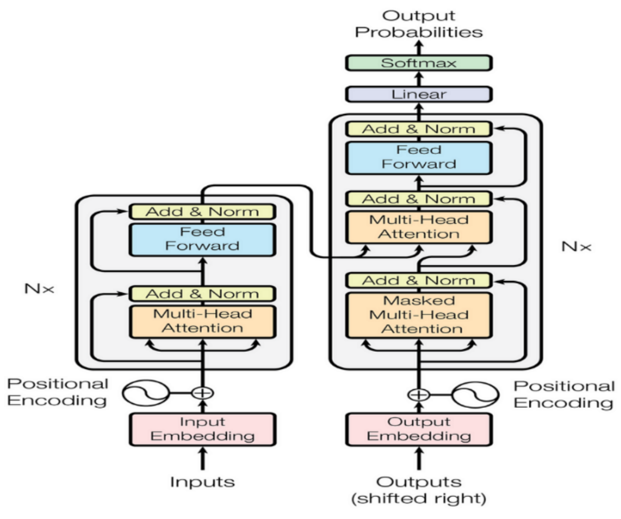


Figure 1: The Transformer - model architecture.

Figure 2.1 Transformer Architecture

The Encoder is on the left and the Decoder is on the right. Both Encoder and Decoder are composed of modules that can be stacked on top of each other multiple times, which is described by *Nx* in the figure. We see that the modules consist mainly of Multi-Head Attention and Feed Forward layers. The inputs and outputs (target sentences) are first embedded into an n-dimensional space since we cannot use strings directly.

MULTI-HEAD ATTENTION

One slight but important part of the model is the positional encoding of the different words. Since we have no recurrent networks that can remember how sequences are fed into a model, we need to somehow give every word/part in our sequence a relative position since a sequence depends on the order of its elements. These positions are added to the embedded representation (n-dimensional vector) of each word.

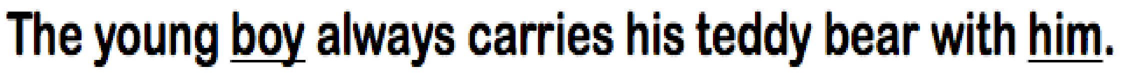

The young boy always carries his teddy bear with him.

Figure 2.2 Long-term dependencies in a sequence: if the model does not remember "boy" from the beginning, it might not know which pronoun to use at the end. Him, her, it?

In the self-attention layer, an input x (represented as a vector) is turned into a vector z via three representational vectors of the input: q(queries), k(keys) and v(values). These are used to calculate a score that shows how much attention that particular input should pay to other elements in the given sequence.

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Figure 2.3 Attention Formula

Q is a matrix that contains the query (vector representation of one word in the sequence), K are all the keys (vector representations of all the words in the sequence) and V are the values, which are again the vector representations of all the words in the sequence. For the encoder and decoder, multi-head attention modules, V consists of the same word sequence than Q. However, for the attention module that is taking into account the encoder and the decoder sequences, V is different from the sequence represented by Q.

EXAMPLE FOR CALCULATING SELF-ATTENTION:

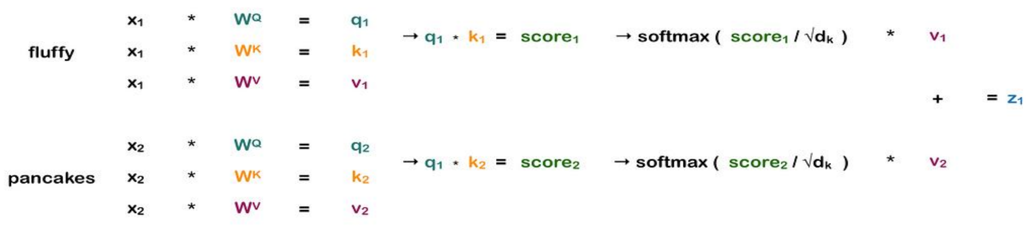

fluffy
x₁ * W^Q = q₁
x₁ * W^K = k₁
x₁ * W^V = v₁
→ q₁ · k₁ = score₁ → softmax (score₁ / √dₖ) * v₁
+ = z₁
pancakes
x₂ * W^Q = q₂
x₂ * W^K = k₂
x₂ * W^V = v₂
→ q₁ · k₂ = score₂ → softmax (score₂ / √dₖ) * v₂

Figure 2.4 Self-attention Calculation

Say we want to calculate **self-attention for the word "fluffy"** in the sequence "fluffy pancakes". First, we take the input vector *x1* (representing the word "fluffy") and multiply it with three different weight matrices Wq, Wk and Wv (which are continually updated during training) in order to get three different vectors: q1, k1 and v1. The exact same is done for the input vector *x2* (representing the word "pancakes"). We now have a query, key and value vector for both words.

- The **query** is the representation for the word we want to calculate self-attention for. So since we want to get the self-attention for "fluffy", we only consider its query, not the one of "pancakes". As soon as we are finished calculating the self-attention for "fluffy", we can also discard its query vector.
- The **key** is a representation of each word in the sequence and is used to match against the query of the word for which we currently want to calculate self-attention.
- The **value** is the actual representation of each word in a sequence, the representation we really care about. Multiplying the query and key gives us a score that tells us how much weight each value (and thus, its corresponding word) obtains in the self-attention vector. Note that the value is not directly multiplied with the score, but first the scores are divided by the square root of the *dk*, the dimension of the key vector, and softmax is applied.

The result of these calculations are one vector for each word. As a final step, these two vectors are summed up, and voilà, we have the self-attention for the word "fluffy".

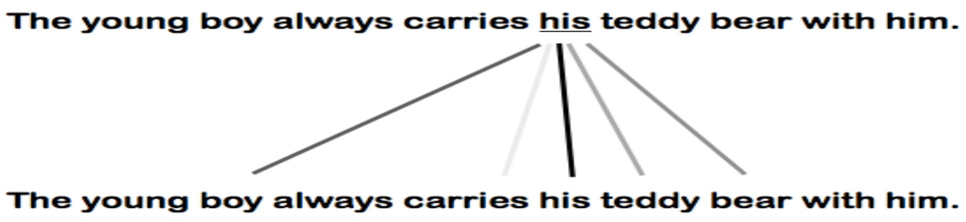

The young boy always carries his teddy bear with him.
The young boy always carries his teddy bear with him.

Figure 2.5 Self-attention for the word "his". The lines indicate how much attention the word "his" pays to other words in the sequence.

We have noticed that it is called **multi-head self-attention.** This is because the process above is carried out multiple times with different weight matrices, which means we end up with multiple vectors (called heads in the formulae below). These heads are then concatenated and multiplied with a weight matrix Wo. This means that each head learns different information about a given sequence and that this knowledge is combined at the end.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h) * W^O$$

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Figure 2.6 Multi-Head Self-attention Formula

As we saw in very first picture showing the Transformer architecture, self-attention layers are integrated in both the encoder and decoder. Which just had a look at what self-attention looks like in the encoder. The decoder, however, uses what is called **masked multi-head self-attention**. This means that some positions in the decoder input are masked and thus ignored by the self-attention layer. Why do they get masked? When predicting the next word of a sentence, the decoder should not know which word comes after the predicted word. Instead, only words up until the current positions should be known to the decoder. After all, when actually using the model to get real next-word predictions, the decoder cannot see future positions either. So by masking them during training, we don't allow the decoder to cheat.

The little black dog ___ [masked] [masked].

vs.

The little black dog ___ its tail.

Figure 2.7 In the first sentence (masked), the next word is far more difficult to predict than in the second sentence (unmasked). The words "its tail" make it clear the word to predict is probably "wiggled".

Feed-Forward Networks

Feed-Forward Networks further process he outputs of the self-attention layer before passing them on to the next encoder or decoder.

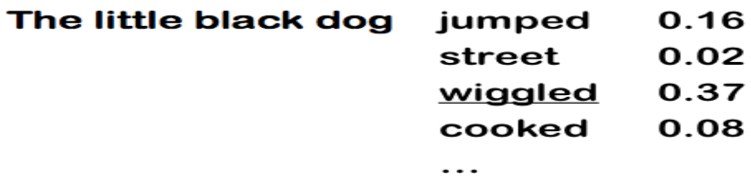
$$\text{FeedForwardNetwork}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

Figure 2.8 Feed-Forward Network Formula

Each feed-forward network consists of two linear layers with a ReLU function in between. The weights and biases W1, W2, b1 and b2 are the same across different positions in the sequence but different in each encoder and decoder.

Linear Layer and Softmax:

In order to get the output predictions, we somehow need to transform the output vector of the last decoder into words. So we first feed the output vector into a fully-connected linear layer and get a logits vector of the size of the vocabulary. We then apply the softmax function to this vector in order to get a probability score for each word in the vocabulary. We then chose the word with maximum probability as our prediction.



The little black dog	jumped	0.16
	street	0.02
	wiggled	0.37
	cooked	0.08
	...	

Figure 2.9 Probability Score

The answer is **wiggled** since it has maximum probability when compared to other word

2.11.3 BERT MODEL:

Sentiment analysis can be implemented using many classification algorithms like Linear Regression, Naive Bayes, Support Vector Machines, RNN derivatives LSTM and GRU. But there are many limitations for classification algorithms.

Some of the limitations of the classification algorithms are:

- **Context:** Sentiment analysis requires an understanding of the context in which the text was written. Classification algorithms typically analyse each document in isolation and do not take into account the broader context. This can lead to inaccurate results, especially when dealing with sarcasm or irony.
- **Subjectivity:** Sentiment analysis is inherently subjective, and the same text can be interpreted differently by different people. Classification algorithms rely on a fixed set of predefined categories, which may not capture the nuances of human emotions.
- **Limited Vocabulary:** Sentiment analysis is highly dependent on the accuracy of the lexicon used to identify sentiment. Classification algorithms rely on a limited set of

pre-defined words or phrases, which may not be comprehensive enough to capture all the nuances of human emotions.

- **Data Quality:** Sentiment analysis algorithms require high-quality data to produce accurate results. Poor-quality data can lead to incorrect conclusions, especially when the data is noisy, incomplete, or contains irrelevant information.
- **Multilingualism:** Sentiment analysis in multiple languages is challenging. Classification algorithms often require separate training data for each language, which can be time-consuming and expensive.

We can also use the word embeddings like Word2Vec and GloVe to implement the sentiment analysis.

WORD2VEC:

Word2Vec is a neural network-based algorithm for generating word embeddings, which are high-dimensional vectors that represent words in a continuous vector space. The algorithm is designed to capture the semantic relationships between words by encoding the distributional information of the words in large corpora of text.

Word2Vec works by training a shallow neural network on a large corpus of text. The algorithm treats each word in the corpus as a unique entity and generates a vector representation of each word based on the context in which it appears. The context of a word is defined by the words that appear around it in the text.

For example, consider the sentence: "**The cat chased the mouse.**" In this sentence, "cat" and "mouse" are closely related in meaning. Word2Vec would represent these words as similar vectors in the vector space. The algorithm would achieve this by training the neural network to predict the probability of a word appearing in the context of another word.

LIMITATIONS OF WORD2VEC:

- One of the main limitations of Word2Vec is that it is based on the distributional hypothesis, which assumes that words that appear in similar contexts have similar meanings. While this hypothesis is generally true, it can lead to inaccuracies in cases where two words have different meanings but are used in similar contexts.
- Another limitation of Word2Vec is that it can be computationally expensive to train the neural network on large corpora of text. Additionally, the quality of the resulting embeddings is highly dependent on the quality and size of the training data.

GLOVE:

GloVe (Global Vectors for Word Representation) is another popular algorithm for generating word embeddings. It also uses the distributional hypothesis to generate vector representations of words, but it differs from Word2Vec in the way it captures word co-occurrence statistics.

Unlike Word2Vec, which trains a neural network on a large corpus of text to predict the probability of a word appearing in the context of another word, GloVe generates word embeddings by performing a matrix factorization on the co-occurrence matrix of the words in the corpus.

For example, consider the sentence: "**The cat chased the mouse.**" GloVe would represent the words "cat" and "mouse" as similar vectors in the vector space based on their co-occurrence in the sentence. However, unlike Word2Vec, GloVe would also capture the relationship between the word "cat" and the context "chased the," and the relationship between the word "mouse" and the context "the chased."

LIMITATIONS OF GLOVE:

- One of the main limitations of GloVe is that it requires a large amount of memory to store the co-occurrence matrix, which can be challenging for very large corpora. Additionally, the quality of the resulting embeddings is highly dependent on the quality and size of the training data.
- Another limitation of GloVe is that it is less effective at capturing syntactic relationships between words than it is at capturing semantic relationships. This is because it does not take into account the order in which words appear in the text.

Since the classification algorithms, Word2Vec and GloVe models have many limitations, We are using new technology or model, called BERT, which overcomes the limitations of the classification algorithms, Word2Vec and GloVe models.

BERT:

BERT stands for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers. BERT is based on the transformer architecture. BERT is pre-trained on a large corpus of unlabelled text including the entire Wikipedia (that's 2,500 million words!) and Book Corpus (800 million words). This pre-training step is half the magic behind BERT's success. This is because as we train a model on a large text corpus, our model starts to pick up the deeper and intimate understandings of how the language works. BERT is a **"deeply bidirectional"** model. Bidirectional means that BERT learns information from both the left and the right side of a token's context during the training phase. The bidirectionality of a model is important for truly understanding the meaning of a language. Let's see an example to illustrate this. There are two sentences in this example and both of them involve the word "bank":

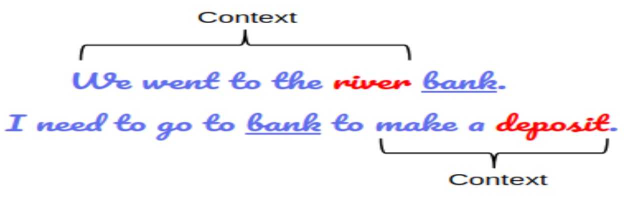

Context
We went to the river bank.
I need to go to bank to make a deposit.
Context

Figure 2.10 BERT captures both the left and right context

BERT's ARCHITECTURE:

The BERT architecture builds on top of Transformer. We currently have two variants available:

- BERT Base: 12 layers (transformer blocks), 12 attention heads, and 110 million parameters
- BERT Large: 24 layers (transformer blocks), 16 attention heads and, 340 million parameters

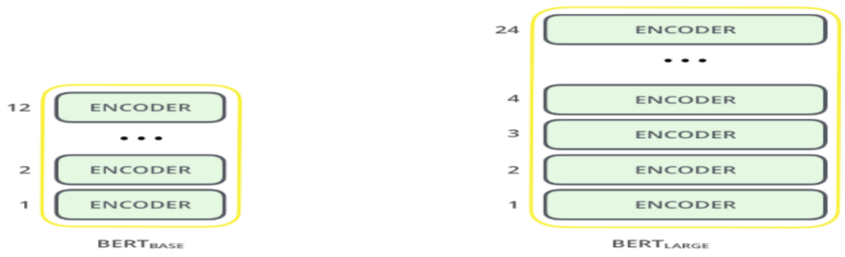


Figure 2.11 BERT Architecture

TEXT PRE-PROCESSING (USING BERT):

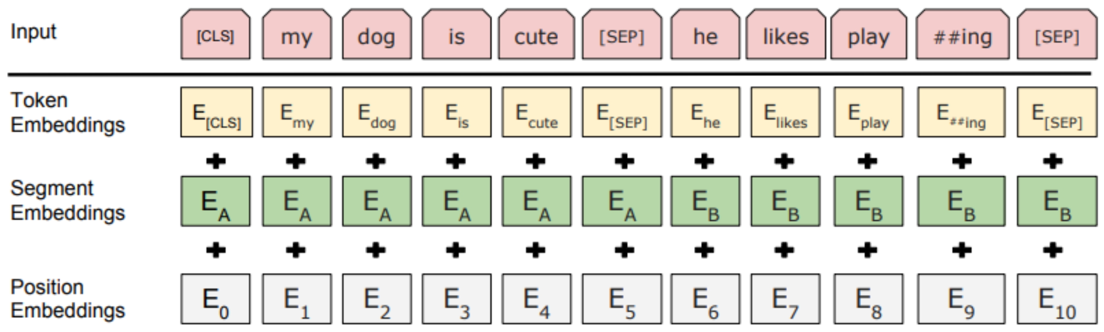

Input
[CLS]
my
dog
is
cute
[SEP]
he
likes
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$ E_{my} E_{dog} E_{is} E_{cute} $E_{[SEP]}$ E_{he} E_{likes} E_{play} $E_{\#\#ing}$ $E_{[SEP]}$
Segment Embeddings
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
Position Embeddings
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10}

Figure 2.12 Embeddings in text pre-processing

- **Position Embeddings**: BERT learns and uses positional embeddings to express the position of words in a sentence. These are added to overcome the limitation of Transformer which, unlike an RNN, is not able to capture "sequence" or "order" information.
- **Segment Embeddings**: BERT can also take sentence pairs as inputs for tasks (Question-Answering). That's why it learns a unique embedding for the first and the second sentences to help the model distinguish between them. In the above example, all the tokens marked as EA belong to sentence A (and similarly for EB).
- **Token Embeddings**: These are the embeddings learned for the specific token from the WordPiece token vocabulary.

3. SOFTWARE SYSTEM DESIGN

System design is the process of designing the elements of a system such as the architecture, modules and components, the different interfaces of those components and the data that goes through that system. System Analysis is the process that decomposes a system into its component pieces for the purpose of defining how well those components interact to accomplish the set requirements.

The purpose of the System Design process is to provide sufficient detailed data and information about the system. The purpose of the design phase is to plan a solution of the problem specified by the requirement document. This phase is the first step in moving from problem domain to the solution domain. The design of a system is perhaps the most critical factor affecting the quality of the software, and has a major impact on the later phases, particularly testing and maintenance.

The design activity is often divided into two separate phase-system design and detailed design. System design, which is sometimes also called top-level design, aims to identify the modules that should be in the system, the specifications of these modules, and how they interact with each other to produce the desired results.

A design methodology is a systematic approach to creating a design by application of set of techniques and guidelines. Most methodologies focus on system design. The two basic principles used in any design methodology are problem partitioning and abstraction. Abstraction is a concept related to problem.

3.1 PROCESS FLOW DIAGRAM

A process flowchart is a graphical representation of a business process through flowchart. It's used as a means of getting a top-down understanding of how a process works, what steps it consists of, what events change outcomes, and so on.

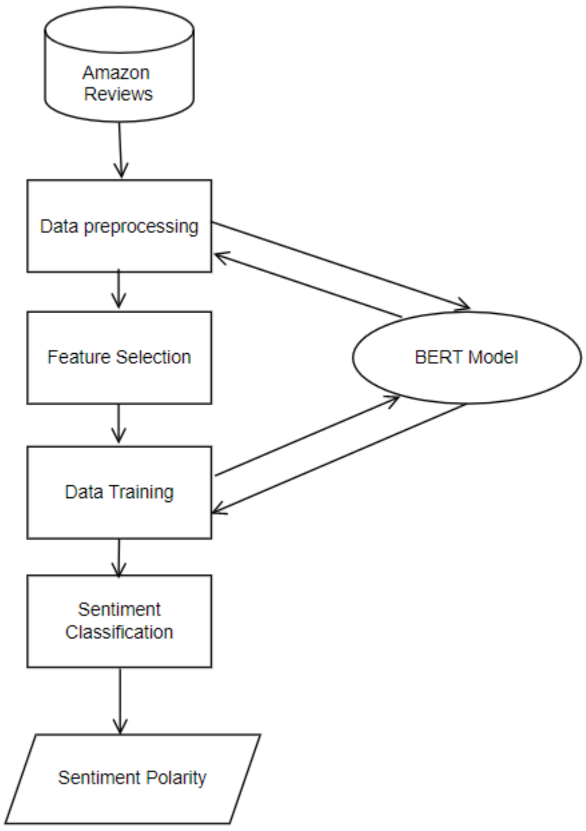


Figure 3.1 Process Flow Diagram

3.2 CLASS DIAGRAM

The class diagram can be used to show the classes, relationships, interface, association, and collaboration. UML is standardized in class diagrams. The main purpose to use class diagrams are:

- This is the only UML which can appropriately depict various aspects of OOPs concept.
- Proper design and analysis of application can be faster and efficient.
- Each class is represented by a rectangle having a subdivision of three compartments name, attributes and operation.
- There are three types of modifiers which are used to decide the visibility of attributes and operations.
- + is used for public visibility (for everyone)
- # is used for protected visibility (for friend and derived).
- – is used for private visibility (for only me)

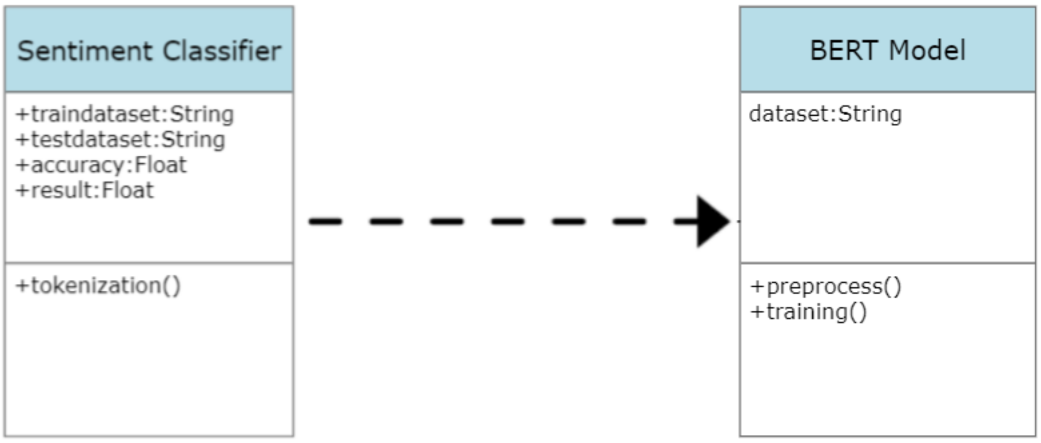


Figure 3.2 Class Diagram

3.3 INTERACTION DIAGRAMS

From the term Interaction, it is clear that the diagram is used to describe some type of interactions among the different elements in the model. This interaction is a part of dynamic behaviour of the system. This interactive behaviour is represented in UML by two diagrams known as Sequence diagram and Collaboration diagram. The basic purpose of both the diagrams are similar. Sequence diagram emphasizes on time sequence of messages and collaboration diagram emphasizes on the structural organization of the objects that send and receive messages. The purpose of interaction diagrams is to visualize the interactive behaviour of the system. Visualizing the interaction is a difficult task. Hence, the solution is to use different types of models to capture the different aspects of the interaction. Sequence and collaboration diagrams are used to capture the dynamic nature but from a different angle.

The purpose of interaction diagram is –

- To capture the dynamic behaviour of a system.
- To describe the message flow in the system.
- To describe the structural organization of the objects.
- To describe the interaction among objects.

The main purpose of both the diagrams are similar as they are used to capture the dynamic behaviour of a system.

However, the specific purpose is more important to clarify and understand. Sequence diagrams are used to capture the order of messages flowing from one object to another. Collaboration diagrams are used to describe the structural organization of the objects taking part in the interaction. A single diagram is not sufficient to describe the dynamic aspect of an entire system, so a set of diagrams are used to capture it as a whole. Interaction diagrams are used when we want to understand the message flow and the structural organization. Message flow means the sequence of control flow from one object to another.

3.3.1 SEQUENCE DIAGRAM

A sequence diagram is a type of UML (Unified Modeling Language) diagram that represents interactions between objects or components in a system or process. It is used to model the dynamic behaviour of a system, specifically the flow of messages between objects over time.

Sequence diagrams are useful for visualizing the flow of control in a system, identifying potential sources of errors or bottlenecks, and verifying that the system meets the requirements of stakeholders. They are commonly used in software development to design and document complex systems, as well as in business process modeling to represent workflows and interactions between stakeholders.

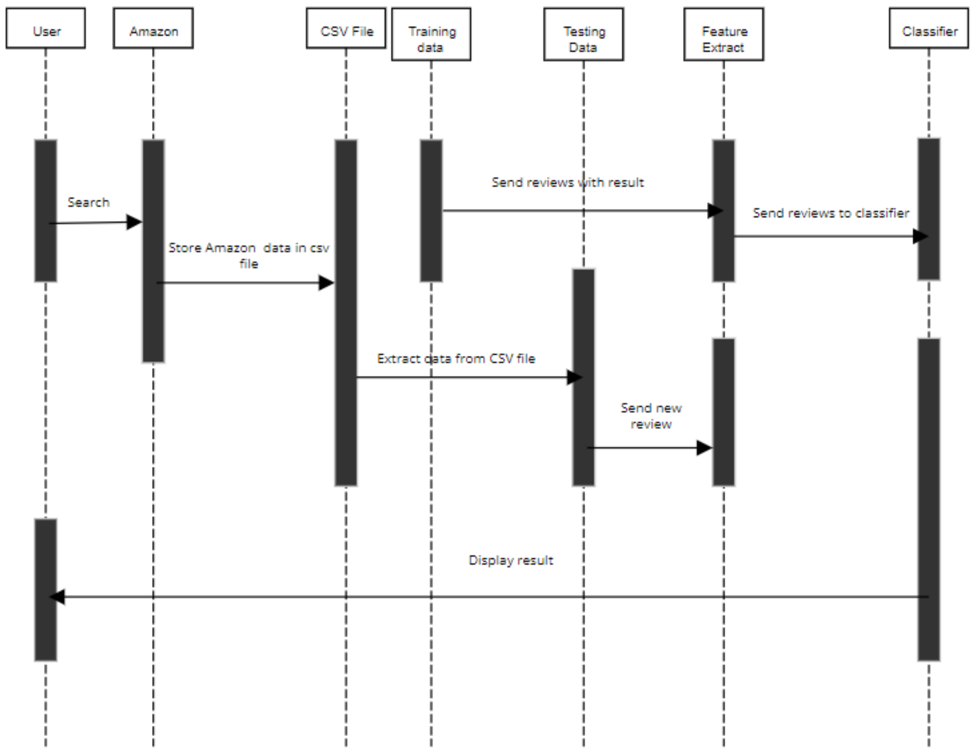


Figure 3.3 Sequence Diagram

3.3.2 COLLABORATION DIAGRAM

The second interaction diagram is the collaboration diagram. It shows the object organization as seen in the following diagram. In the collaboration diagram, the method call sequence is indicated by some numbering technique. The number indicates how the methods are called one after another. We have taken the same order management system to describe the collaboration diagram. Method calls are similar to that of a sequence diagram. However, difference being the sequence diagram does not describe the object organization, whereas the collaboration diagram shows the object organization. To choose between these two diagrams, emphasis is placed on the type of requirement. If the time sequence is important, then the sequence diagram is used. If organization is required, then collaboration diagram is used.

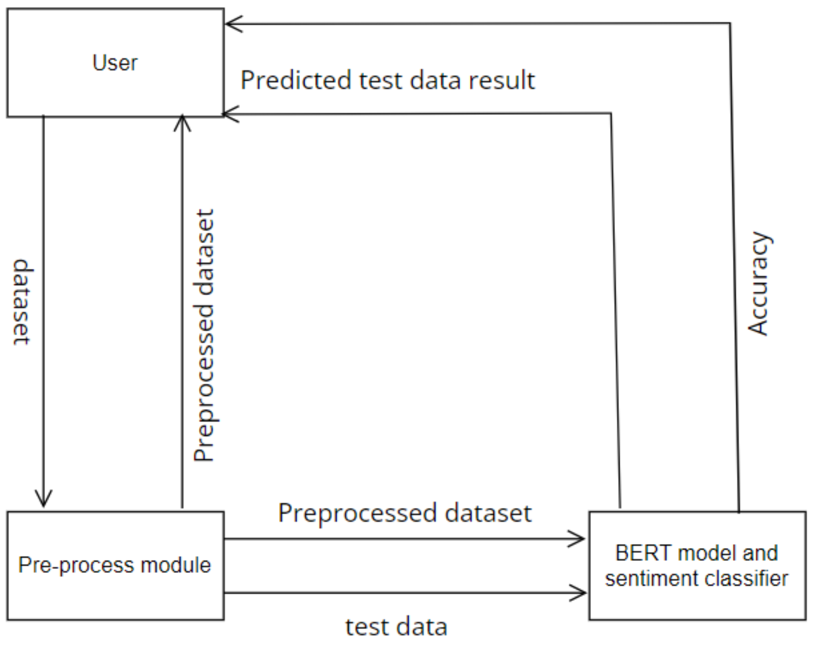


Figure 3.4 Collaboration Diagram

3.4 ACTIVITY DIAGRAM

Activity diagram is defined as a UML diagram that focuses on the execution and flow of the behaviour of a system instead of implementation. It is also called object-oriented flowchart. Activity diagrams consist of activities that are made up of actions which apply to behavioural modeling technology. Activity diagrams are used to model processes and workflows.

Activity Diagram Notations:

- Initial states: The starting stage before an activity takes place is depicted as the initial state
- Final states: The state which the system reaches when a specific process ends is known as a Final State
- State or an activity box: It represents the set of actions that are to be performed.
- Decision box: It is a diamond shape box which represents a decision with alternate paths. It represents the flow of control.

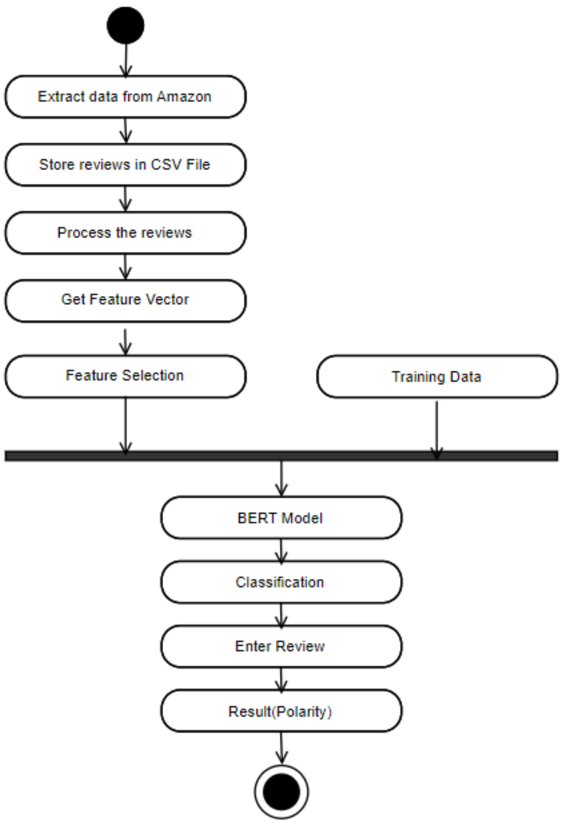


Figure 3.5 Activity Diagram

3.5 USE CASE DIAGRAM

Use case diagram is used to represent the dynamic behaviour of a system. It encapsulates the system's functionality by incorporating use cases, actors, and their relationships. It models the tasks, services, and functions required by a system/subsystem of an application. It depicts the high-level functionality of a system and also tells how the user handles a system.

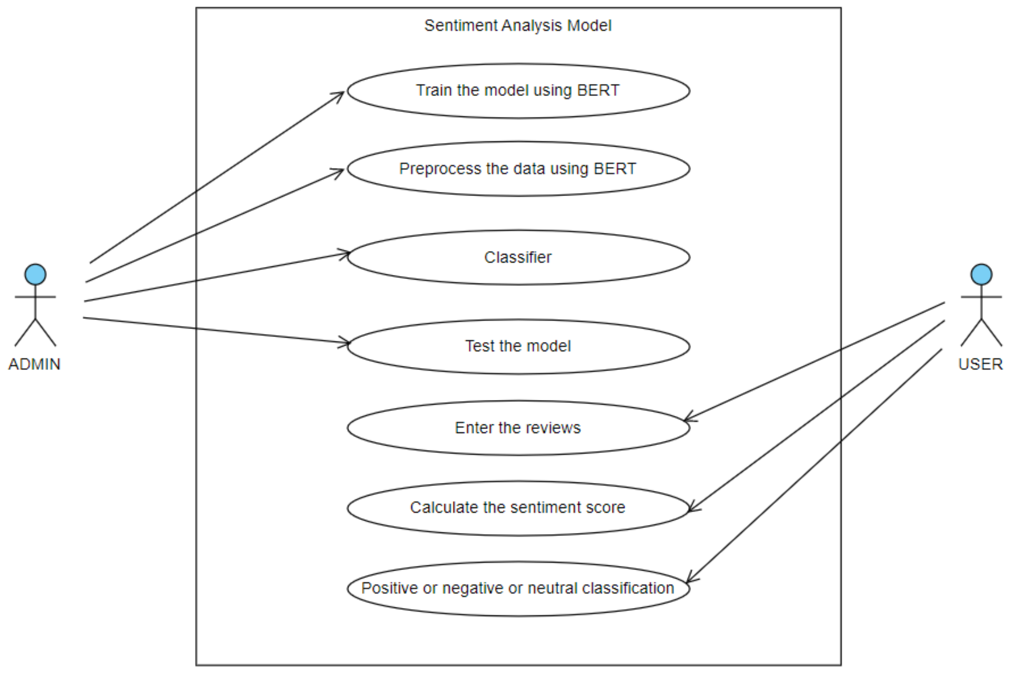

Sentiment Analysis Model
Train the model using BERT
Preprocess the data using BERT
Classifier
Test the model
Enter the reviews
Calculate the sentiment score
Positive or negative or neutral classification
ADMIN
USER

Figure 3.6 Use-case Diagram

4. SRS DOCUMENT

SRS is a document created by a system analyst after the requirements are collected from various stakeholders. SRS defines how the intended software will Interact with hardware, external interfaces, speed of operation, response time of system, portability of software across various platforms, maintainability, speed of recovery after crashing, Security, Quality, Limitations etc.

The requirements received from clients are written in natural language. It is the responsibility of the system analyst to document the requirements in technical language so that they can be comprehended and useful by the software development team. The introduction of the software requirement specification states the goals and objectives of the software, describing it in the context of the computer-base system. The SRS includes an information description, functional description, behavioural description, validation criteria.

The purpose of this document is to present the software requirements in a precise and easily understood manner. This document provides the functional, performance, design and verification requirements of the software to be developed.

After requirement specifications are developed, the requirements mentioned in this document are validated. Users might ask for illegal, impractical solutions or experts may interpret the requirements incorrectly. This results in a huge increase in cost if not nipped in the bud.

4.1 FUNCTIONAL REQUIREMENTS

A functional requirement defines a function of a system or it's component. A function can be described as set of inputs, the behaviour, and outputs. It also depends upon the type of software, expected users and the type of system where the software is used.

Functional requirements of our project are:

- Data Collection: The system should be able to collect a large volume of text data from various sources, such as social media platforms, news websites, customer reviews, etc.
- Text Pre-processing: The system should be able to pre-process the collected data, including text cleaning, tokenization, stop-word removal, stemming, and other techniques to prepare the text for analysis.
- Sentiment Analysis: The system should be able to perform sentiment analysis on the pre-processed text data, which involves identifying the sentiment expressed in the text, such as positive, negative, or neutral.
- Visualization: The system should be able to visualize the sentiment analysis results in various formats, such as charts, graphs, and tables, to help stakeholders understand the sentiment trends and patterns.

4.2 NON-FUNCTIONAL REQUIREMENTS

NON-FUNCTIONAL REQUIREMENT (NFR) specifies the quality attribute of a software system. They judge the software system based on Responsiveness, Usability, Security, Portability.

Non-functional requirements are called qualities of a system, there are as follows:

- Performance-The average response time of the system is less.
- Reliability - The system is highly reliable.
- Operability - The interface of the system will be consistent.
- Efficiency - Once user has learned about the system through his interaction, he can perform the task easily.
- Understandability-Because of user friendly interfaces, it is more understandable to the users.

4.3 MINIMUM SOFTWARE REQUIREMENTS

Tools: Google Colab, Pycharm

Programming Language: Python

Operating System: 64-bit Windows

4.4 MINIMUM HARDWARE REQUIREMENTS

Processor: Intel core i3 or above.

RAM: 8 GB

Hard disk Space: 1 TB

5. OUTPUT

5.1 SYSTEM IMPLEMENTATION

5.1.1 INTRODUCTION

Sentiment analysis is the process of using natural language processing and machine learning techniques to identify and extract subjective information from text. It is an increasingly popular application of artificial intelligence and has numerous real-world applications such as understanding customer feedback, analysing social media sentiment, and monitoring online reputation. Implementing a sentiment analysis system requires a clear understanding of the problem you are trying to solve and the specific data sources and technologies that will be used. The implementation process typically involves collecting and pre-processing data, developing and training a sentiment analysis model, integrating the model into your application or system, and deploying and monitoring the system to ensure it provides accurate sentiment analysis. Continuous monitoring and improvement of the system are also necessary to maintain its accuracy and usefulness over time. In our project, we will follow these steps to build a sentiment analysis system that can accurately classify text as positive, negative, or neutral.

5.1.2 PROJECT MODULES

1. **Data collection:** This module involves gathering the data needed to perform sentiment analysis. This may involve web scraping, API calls, or working with pre-existing datasets.
2. **Data pre-processing:** Once the data is collected, it needs to be cleaned and prepared for analysis. This may involve removing stop words, stemming or lemmatizing words, and dealing with missing or irrelevant data.
3. **Feature extraction:** This module involves selecting and extracting features from the pre-processed data. Some common feature extraction techniques include bag-of-words, TF-IDF, and word embeddings.
4. **Model selection:** There are many different machine and deep learning algorithms that can be used for sentiment analysis, including logistic regression, support vector machines, neural networks, Bi-LSTM, BERT. This module involves selecting the most appropriate algorithm for the task at hand.

5. **Model training:** Once the model is selected, it needs to be trained on the pre-processed data. This involves splitting the data into training and testing sets and using the training set to optimize the model parameters.
6. **Model evaluation:** After the model is trained, it needs to be evaluated on the testing set to determine how well it performs. This may involve metrics such as accuracy, precision, recall, and F1 score.
7. **Model optimization:** Based on the evaluation results, the model may need to be fine-tuned or optimized to improve its performance.
8. **Deployment:** Once the model is optimized, it can be deployed to make predictions on new data. This may involve integrating the model into a web application or other software platform.
9. **Monitoring and maintenance:** After the model is deployed, it needs to be monitored to ensure that it continues to perform well over time. This may involve periodic retraining or updating of the model as new data becomes available.

5.2 SOURCE CODE

# Installing Dependencies

```
!pip install transformers

import transformers
from transformers import BertModel, BertTokenizer, AdamW, get_linear_schedule_with_warmup
import torch
import gc
gc.collect()
torch.cuda.empty_cache()

import numpy as np
import pandas as pd
import seaborn as sns
from pylab import rcParams
import matplotlib.pyplot as plt
from matplotlib import rc
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
from collections import defaultdict
from textwrap import wrap

from torch import nn, optim
from torch.utils.data import Dataset, DataLoader
import torch.nn.functional as F

%matplotlib inline
%config InlineBackend.figure_format='retina'


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

RANDOM_SEED = 42
np.random.seed(RANDOM_SEED)
torch.manual_seed(RANDOM_SEED)

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
device
```

#Read the Data

```
df = pd.read_csv("reviews.csv")
df.head()

df.shape

df.info()
```

#We have a balanced dataset with respect to reviews

```
def group_sentiment(rating):
  #rating = int(rating)
  if rating <= 2:
    return 0        # Negative sentiment
  elif rating == 3:
    return 1        # Neutral Sentiment
  else:
    return 2        # positive Sentiment

df['sentiment'] = df.rating.apply(group_sentiment)

class_names = ['negative', 'neutral', 'positive']
```

#Data Preprocessing

Deep Learning models don't work with raw text. We need to convert text to numbers (of some sort). BERT requires even more attention. Here are the requirements:

- Add special tokens to separate sentences and do classification
- Pass sequences of constant length (introduce padding)
- Create array of 0s (pad token) and 1s (real token) called attention mask

The Transformers library provides a wide variety of Transformer models (including BERT). It also includes prebuilt tokenizers that solves most of the load required for pre-processing

```
PRE_TRAINED_MODEL_NAME = 'bert-base-cased'

tokenizer = BertTokenizer.from_pretrained(PRE_TRAINED_MODEL_NAME)
```

#Using a simple text to understand the tokenization process:

```
sample_txt = 'Best place that I have visited? Iceland was the most beautiful and I consider my
self lucky to have visited Iceland at such an early age.'
#sample_txt = 'When was I last outside? I am stuck at home for 2 weeks.'

tokens = tokenizer.tokenize(sample_txt)
token_ids = tokenizer.convert_tokens_to_ids(tokens)

print(f'  Sentence: {sample_txt}')
print(f'\n  Tokens: {tokens}')
print(f'\n  Token IDs: {token_ids}')   # Each token has a an unique ID for the model to unsers
tand what we are referring to.

len(tokens)
```

#Special Tokens

#[SEP] - marker for ending of a sentence

```
tokenizer.sep_token, tokenizer.sep_token_id
```

[CLS] - we must add this token to the start of each sentence, so BERT knows we're doing classification.

```
tokenizer.cls_token, tokenizer.cls_token_id

tokenizer.pad_token, tokenizer.pad_token_id

tokenizer.unk_token, tokenizer.unk_token_id
```

#All of the above work can be done using the encode_plus() method

```
encoding_test = tokenizer.encode_plus(
  sample_txt,
  max_length=32,           # sequence length
  add_special_tokens=True, # Add '[CLS]' and '[SEP]'
  return_token_type_ids=False,
  pad_to_max_length=True,
  return_attention_mask=True,
  return_tensors='pt',  # Return PyTorch tensors(use tf for tensorflow and keras)
)

encoding_test.keys()


print(' length of the first sequence is :  ', len(encoding_test['input_ids'][0]))
print('\n The input id\'s are : \n', encoding_test['input_ids'][0])
print('\n The attention mask generated is : ', encoding_test['attention_mask'][0])
```

```
tokenizer.convert_ids_to_tokens(encoding_test['input_ids'].flatten())
```

#Choosing Sequence Length

```
df.loc[df.summary.isnull()]


df = df[df['summary'].notna()]


token_lens = []
for text in df.summary:
    tokens_df = tokenizer.encode(text, max_length=512
    token_lens.append(len(tokens_df))
```

Most of the reviews seem to contain less than 128 tokens, but we'll be on the safe side and choose a maximum length of 160.

```
MAX_LEN = 160

class GPReviewDataset(Dataset):

  def __init__(self, reviews, targets, tokenizer, max_len):
    self.reviews = reviews
    self.targets = targets
    self.tokenizer = tokenizer
    self.max_len = max_len

  def __len__(self):
    return len(self.reviews)

  def __getitem__(self, item):
    review = str(self.reviews[item])
    target = self.targets[item]

    encoding = self.tokenizer.encode_plus(
      review,
      add_special_tokens=True,
      max_length=self.max_len,
      return_token_type_ids=False,
      pad_to_max_length=True,
      return_attention_mask=True,
      return_tensors='pt',
    )

    return {
      'review_text': review,
      'input_ids': encoding['input_ids'].flatten(),
      'attention_mask': encoding['attention_mask'].flatten(),
      'targets': torch.tensor(target, dtype=torch.long)
    }
```

#Splitting into train and validation sets

```
df_train, df_test = train_test_split(df, test_size=0.2, random_state=RANDOM_SEED)
df_val, df_test = train_test_split(df_test, test_size=0.5, random_state=RANDOM_SEED)


df_train.shape, df_val.shape, df_test.shape
```

#Creating data loaders to feed as input to our model. The below function does that.

```
def create_data_loader(df, tokenizer, max_len, batch_size):
  ds = GPReviewDataset(
    reviews=df.summary.values,
    targets=df.sentiment.values,
    tokenizer=tokenizer,
    max_len=max_len
  )

  return DataLoader(
    ds,
    batch_size=batch_size,
  )

BATCH_SIZE = 8

train_data_loader = create_data_loader(df_train, tokenizer, MAX_LEN, BATCH_SIZE)
val_data_loader = create_data_loader(df_val, tokenizer, MAX_LEN, BATCH_SIZE)
test_data_loader = create_data_loader(df_test, tokenizer, MAX_LEN, BATCH_SIZE)


data = next(iter(train_data_loader))
data.keys()

print(data['input_ids'].shape)
print(data['attention_mask'].shape)
print(data['targets'].shape)
```

#Sentiment Classification with BERT

#We'll use the basic BertModel and build our sentiment classifier on top of it. Let's load the model:

```
bert_model = BertModel.from_pretrained(PRE_TRAINED_MODEL_NAME)
```

```
model_test = bert_model(
  input_ids=encoding_test['input_ids'],
  attention_mask=encoding_test['attention_mask']
)
model_test.keys()


last_hidden_state=model_test['last_hidden_state']
pooled_output=model_test['pooler_output']


last_hidden_state.shape

bert_model.config.hidden_size

pooled_output.shape
```

#We have used all this knowledge to create a sentiment classifier that uses the BERT model:

```
class SentimentClassifier(nn.Module):

  def __init__(self, n_classes):
    super(SentimentClassifier, self).__init__()
    self.bert = BertModel.from_pretrained(PRE_TRAINED_MODEL_NAME)
    self.drop = nn.Dropout(p=0.3)
    self.out = nn.Linear(self.bert.config.hidden_size, n_classes)

  def forward(self, input_ids, attention_mask):
    returned = self.bert(
      input_ids=input_ids,
      attention_mask=attention_mask
    )
    pooled_output = returned["pooler_output"]
    output = self.drop(pooled_output)
    return self.out(output)


model = SentimentClassifier(len(class_names))
model = model.to(device)



input_ids = data['input_ids'].to(device)
attention_mask = data['attention_mask'].to(device)
```

```
print(input_ids.shape)      # batch size x seq length
print(attention_mask.shape) # batch size x seq length


F.softmax(model(input_ids, attention_mask), dim=1)
```

#**Training the model**

```
EPOCHS = 10

optimizer = AdamW(model.parameters(), lr=2e-5, correct_bias=False)
total_steps = len(train_data_loader) * EPOCHS

scheduler = get_linear_schedule_with_warmup(
  optimizer,
  num_warmup_steps=0,     # Recommended in the BERT paper.
  num_training_steps=total_steps
)

loss_fn = nn.CrossEntropyLoss().to(device)
```

#Helper function for training our model for one epoch:

```
def eval_model(model, data_loader, loss_fn, device, n_examples):
  model = model.eval()        # To make sure that the droupout and normalization is disabled d
uring the training.

  losses = []
  correct_predictions = 0

  with torch.no_grad():        # Back propogation is not required. Torch would perform faster.
    for d in data_loader:
      input_ids = d["input_ids"].to(device)
      attention_mask = d["attention_mask"].to(device)
      targets = d["targets"].to(device)

      outputs = model(
        input_ids=input_ids,
        attention_mask=attention_mask
      )
      max_prob, preds = torch.max(outputs, dim=1)

      loss = loss_fn(outputs, targets)

      correct_predictions += torch.sum(preds == targets)
      losses.append(loss.item())

  return correct_predictions.double() / n_examples, np.mean(losses)
```

#Helper function to evaluate the model on a given data loader:

```
def eval_model(model, data_loader, loss_fn, device, n_examples):
  model = model.eval()        # To make sure that the droupout and normalization is disabled d
uring the training.

  losses = []
  correct_predictions = 0

  with torch.no_grad():        # Back propogation is not required. Torch would perform faster.
    for d in data_loader:
      input_ids = d["input_ids"].to(device)
      attention_mask = d["attention_mask"].to(device)
      targets = d["targets"].to(device)

      outputs = model(
        input_ids=input_ids,
        attention_mask=attention_mask
      )
      max_prob, preds = torch.max(outputs, dim=1)

      loss = loss_fn(outputs, targets)

      correct_predictions += torch.sum(preds == targets)
      losses.append(loss.item())

  return correct_predictions.double() / n_examples, np.mean(losses)
```

Using these two helper functions, we can write our training loop. We'll also store the training history:

```
%%time

history = defaultdict(list)          # Similar to Keras library saves history
best_accuracy = 0

for epoch in range(EPOCHS):

  print(f'Epoch {epoch + 1}/{EPOCHS}')
  print('-' * 10)

  train_acc, train_loss = train_epoch(
    model,
    train_data_loader,
    loss_fn,
    optimizer,
    device,
    scheduler,
    len(df_train)
  )

  print(f'Train loss {train_loss} accuracy {train_acc}')

  val_acc, val_loss = eval_model(
    model,
    val_data_loader,
    loss_fn,
    device,
    len(df_val)
  )

  print(f'Val   loss {val_loss} accuracy {val_acc}')
  print()

  history['train_acc'].append(train_acc)
  history['train_loss'].append(train_loss)
  history['val_acc'].append(val_acc)
  history['val_loss'].append(val_loss)

  if val_acc > best_accuracy:
    torch.save(model.state_dict(), 'best_model_state.bin')
    best_accuracy = val_acc
```

#Evaluation

```
test_acc, _ = eval_model(
  model,
  test_data_loader,
  loss_fn,
  device,
  len(df_test)
)

test_acc.item()
```

#We'll define a helper function to get the predictions from our model

```
def get_predictions(model, data_loader):
  model = model.eval()

  review_texts = []
  predictions = []
  prediction_probs = []
  real_values = []

  with torch.no_grad():
    for d in data_loader:
      texts = d["review_text"]
      input_ids = d["input_ids"].to(device)
      attention_mask = d["attention_mask"].to(device)
      targets = d["targets"].to(device)

      outputs = model(
        input_ids=input_ids,
        attention_mask=attention_mask
      )
      _, preds = torch.max(outputs, dim=1)

      probs = F.softmax(outputs, dim=1)

      review_texts.extend(texts)
      predictions.extend(preds)
      prediction_probs.extend(probs)
      real_values.extend(targets)

  predictions = torch.stack(predictions).cpu()
  prediction_probs = torch.stack(prediction_probs).cpu()
  real_values = torch.stack(real_values).cpu()
  return review_texts, predictions, prediction_probs, real_values

y_review_texts, y_pred, y_pred_probs, y_test = get_predictions(
  model,
  test_data_loader
)
```

#Classification Report

```
print(classification_report(y_test, y_pred, target_names=class_names))
```

#Confusion Matrix

```
def show_confusion_matrix(confusion_matrix):
  hmap = sns.heatmap(confusion_matrix, annot=True, fmt="d", cmap="Blues")
  hmap.yaxis.set_ticklabels(hmap.yaxis.get_ticklabels(), rotation=0, ha='right')
  hmap.xaxis.set_ticklabels(hmap.xaxis.get_ticklabels(), rotation=30, ha='right')
  plt.ylabel('True sentiment')
  plt.xlabel('Predicted sentiment');

cm = confusion_matrix(y_test, y_pred)
df_cm = pd.DataFrame(cm, index=class_names, columns=class_names)
show_confusion_matrix(df_cm)
```

```
idx = 2
review_text = y_review_texts[idx]
true_sentiment = y_test[idx]
pred_df = pd.DataFrame({
  'class_names': class_names,
  'values': y_pred_probs[idx]
})

print("\n".join(wrap(review_text)))
print()
print(f'True sentiment: {class_names[true_sentiment]}')

review_text = "I love the book. Best book ever"
```

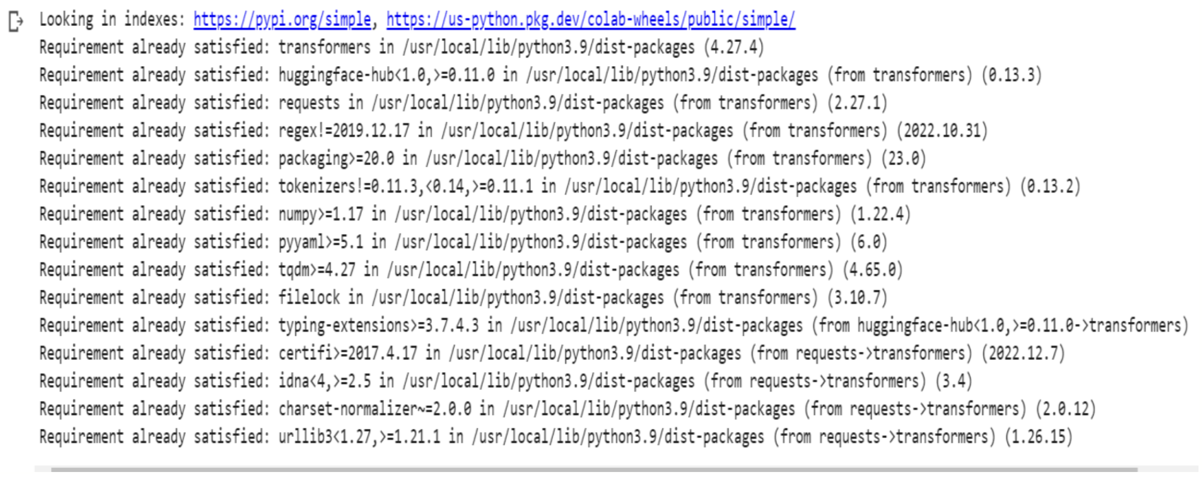
#Use the tokenizer to encode the text

```
encoded_review = tokenizer.encode_plus(
  review_text,
  max_length=MAX_LEN,
  add_special_tokens=True,
  return_token_type_ids=False,
  pad_to_max_length=True,
  return_attention_mask=True,
  return_tensors='pt',
)
```

#The predictions from our model

```
input_ids = encoded_review['input_ids'].to(device)
attention_mask = encoded_review['attention_mask'].to(device)

output = model(input_ids, attention_mask)
_, prediction = torch.max(output, dim=1)

print(f'Review text: {review_text}')
print(f'Sentiment  : {class_names[prediction]}')
```

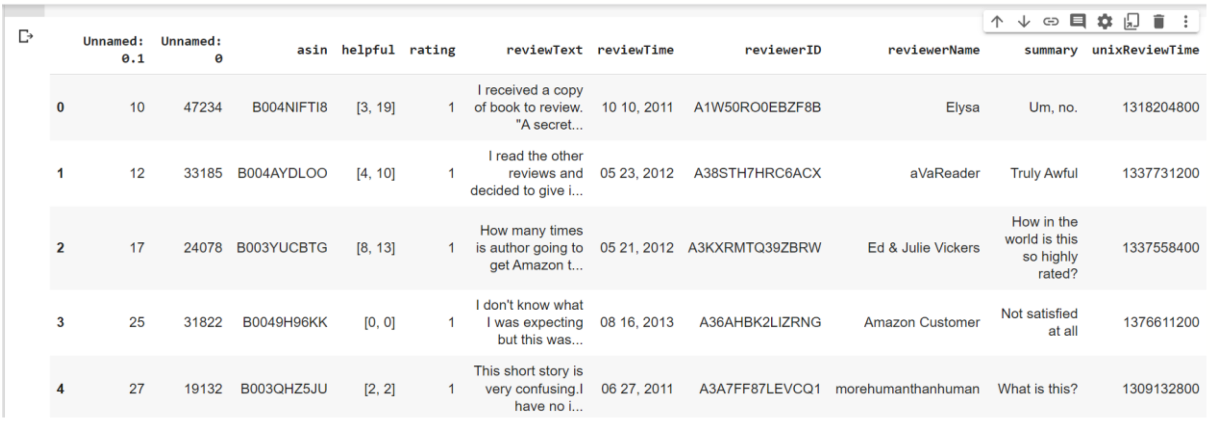
5.3 OUTPUT



```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: transformers in /usr/local/lib/python3.9/dist-packages (4.27.4)
Requirement already satisfied: huggingface-hub<1.0,>=0.11.0 in /usr/local/lib/python3.9/dist-packages (from transformers) (0.13.3)
Requirement already satisfied: requests in /usr/local/lib/python3.9/dist-packages (from transformers) (2.27.1)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.9/dist-packages (from transformers) (2022.10.31)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.9/dist-packages (from transformers) (23.0)
Requirement already satisfied: tokenizers!=0.11.3,<0.14,>=0.11.1 in /usr/local/lib/python3.9/dist-packages (from transformers) (0.13.2)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.9/dist-packages (from transformers) (1.22.4)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.9/dist-packages (from transformers) (6.0)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.9/dist-packages (from transformers) (4.65.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.9/dist-packages (from transformers) (3.10.7)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.9/dist-packages (from huggingface-hub<1.0,>=0.11.0->transformers)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.9/dist-packages (from requests->transformers) (2022.12.7)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.9/dist-packages (from requests->transformers) (3.4)
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.9/dist-packages (from requests->transformers) (2.0.12)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.9/dist-packages (from requests->transformers) (1.26.15)
```

Figure 5.1 Installation

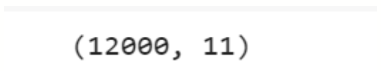
Figure 5.1 refers to the installation of all the libraries which are required for our project.



	Unnamed: 0.1	Unnamed: 0	asin	helpful	rating	reviewText	reviewTime	reviewerID	reviewerName	summary	unixReviewTime
0	10	47234	B004NIFTI8	[3, 19]	1	I received a copy of book to review. "A secret...	10 10, 2011	A1W50RO0EBZF8B	Elysa	Um, no.	1318204800
1	12	33185	B004AYDLOO	[4, 10]	1	I read the other reviews and decided to give i...	05 23, 2012	A38STH7HRC6ACX	aVaReader	Truly Awful	1337731200
2	17	24078	B003YUCBTG	[8, 13]	1	How many times is author going to get Amazon t...	05 21, 2012	A3KXRMTQ39ZBRW	Ed & Julie Vickers	How in the world is this so highly rated?	1337558400
3	25	31822	B0049H96KK	[0, 0]	1	I don't know what I was expecting but this was...	08 16, 2013	A36AHBK2LIZRNG	Amazon Customer	Not satisfied at all	1376611200
4	27	19132	B003QHZ5JU	[2, 2]	1	This short story is very confusing.I have no i...	06 27, 2011	A3A7FF87LEVCQ1	morehumanthanhuman	What is this?	1309132800

Figure 5.2 Reading the data

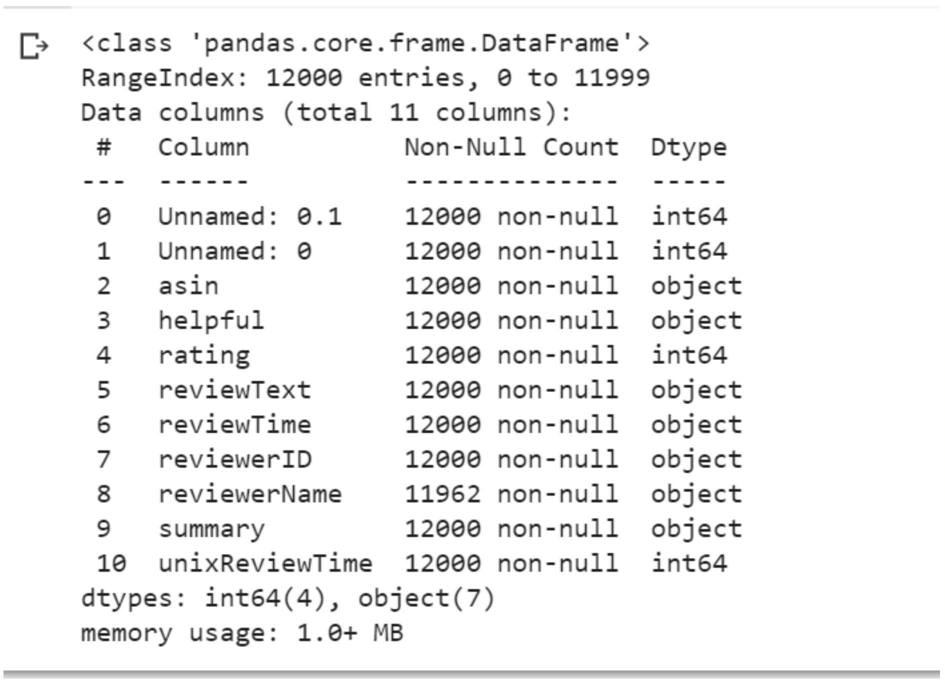
Figure 5.2 refers to reading of the csv File which is used in our project. The csv file which is used in our project is reviews.csv. The attributes present in that file are rating, reviewText, reviewTime, summary etc.



```
(12000, 11)
```

Figure 5.3 Shape of the Dataset

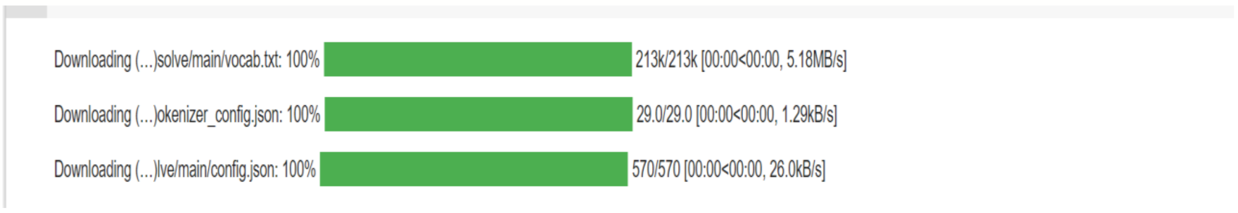
Figure 5.3 refers to the number of attributes and number of rows present in our dataset.



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12000 entries, 0 to 11999
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Unnamed: 0.1    12000 non-null  int64
 1   Unnamed: 0      12000 non-null  int64
 2   asin            12000 non-null  object
 3   helpful         12000 non-null  object
 4   rating          12000 non-null  int64
 5   reviewText      12000 non-null  object
 6   reviewTime      12000 non-null  object
 7   reviewerID      12000 non-null  object
 8   reviewerName    11962 non-null  object
 9   summary         12000 non-null  object
 10  unixReviewTime  12000 non-null  int64
dtypes: int64(4), object(7)
memory usage: 1.0+ MB
```

Figure 5.4 Attributes in the Dataset

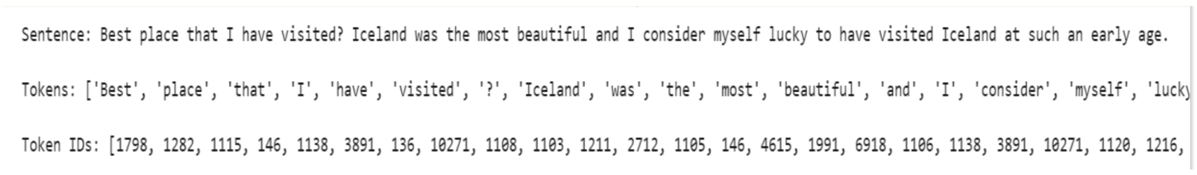
Figure 5.4 contains the name of the attributes present in our dataset along with the attributes datatype



```
Downloading (…)solve/main/vocab.txt: 100%          213k/213k [00:00<00:00, 5.18MB/s]
Downloading (…)okenizer_config.json: 100%          29.0/29.0 [00:00<00:00, 1.29kB/s]
Downloading (…)lve/main/config.json: 100%          570/570 [00:00<00:00, 26.0kB/s]
```

Figure 5.5 Downloading the bert-base-cased

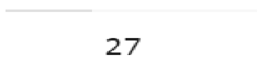
Figure 5.5 refers to the installation of the bert-base cased package in our system. It helps to find the polarity of the given sentence.



```
Sentence: Best place that I have visited? Iceland was the most beautiful and I consider myself lucky to have visited Iceland at such an early age.

Tokens: ['Best', 'place', 'that', 'I', 'have', 'visited', '?', 'Iceland', 'was', 'the', 'most', 'beautiful', 'and', 'I', 'consider', 'myself', 'lucky

Token IDs: [1798, 1282, 1115, 146, 1138, 3891, 136, 10271, 1108, 1103, 1211, 2712, 1105, 146, 4615, 1991, 6918, 1106, 1138, 3891, 10271, 1120, 1216,
```

Figure 5.6 Tokenization

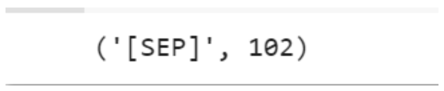
Figure 5.6 refers to the tokenization. The sentence which is taken as an example is tokenized and each token is given a specific ID called token ID.



```
27
```

Figure 5.7 Length of the token

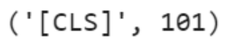
Figure 5.7 refers to the length of the tokens that we have tokenized. The sentence that we have tokenized contains 27 tokens.



```
('[SEP]', 102)
```

Figure 5.8 Special token SEP

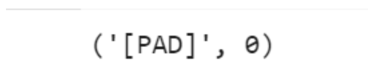
Figure 5.8 refers to the special token called SEP which represents the marker for ending of a sentence



```
('[CLS]', 101)
```

Figure 5.9 Special token CLS

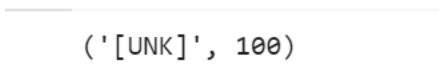
Figure 5.9 refers to the special token called CLS.CLS must be added to the start of each sentence, so BERT knows we're doing classification.



```
('[PAD]', 0)
```

Figure 5.10 Special token PAD

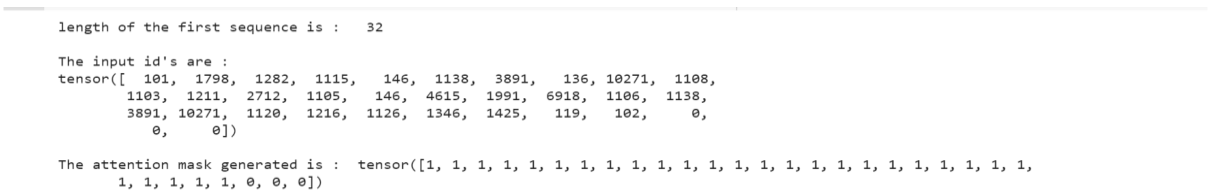
Figure 5.10 refers to the special token called PAD which is used for padding.



```
('[UNK]', 100)
```

Figure 5.11 Special token UNK

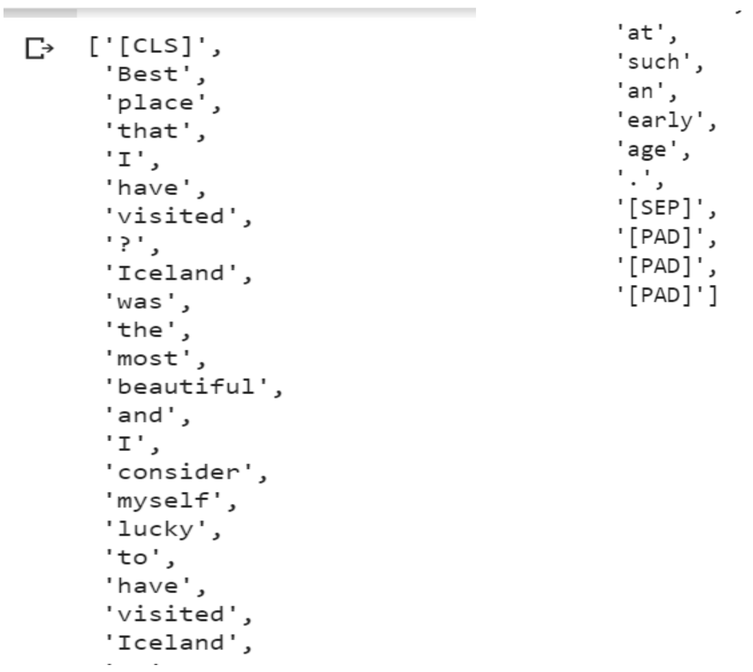
Figure 5.11 refers to the special token UNK. BERT understands tokens that were in the training set. Everything else can be encoded using the [UNK] (unknown) token.



```
length of the first sequence is :   32

The input id's are :
tensor([  101,  1798,  1282,  1115,   146,  1138,  3891,   136, 10271,  1108,
         1103,  1211,  2712,  1105,   146,  4615,  1991,  6918,  1106,  1138,
         3891, 10271,  1120,  1216,  1126,  1346,  1425,   119,   102,     0,
            0,     0])

The attention mask generated is :  tensor([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 0, 0, 0])
```

Figure 5.12 attention mask

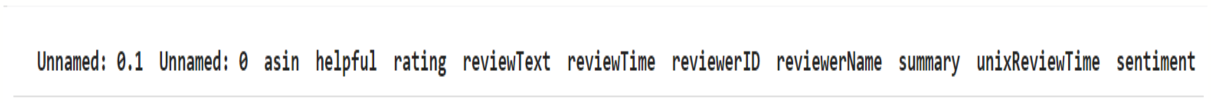
Figure 5.12 refers to the attention mask. The sequence of the attention mask is generated in the above figure.



```
['[CLS]',                  'at',
 'Best',                   'such',
 'place',                  'an',
 'that',                   'early',
 'I',                      'age',
 'have',                   '.',
 'visited',                '[SEP]',
 '?',                      '[PAD]',
 'Iceland',                '[PAD]',
 'was',                    '[PAD]']
 'the',
 'most',
 'beautiful',
 'and',
 'I',
 'consider',
 'myself',
 'lucky',
 'to',
 'have',
 'visited',
 'Iceland',
```

Figure 5.13 Inversing the Tokenization

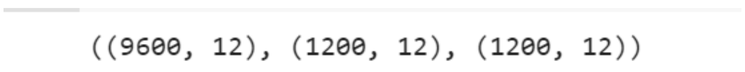
Figure 5.13 refers to the inverse of the tokenization. Inversing of the tokenization is done to have a look at the special tokens



```
Unnamed: 0.1  Unnamed: 0  asin  helpful  rating  reviewText  reviewTime  reviewerID  reviewerName  summary  unixReviewTime  sentiment
```

Figure 5.14 Null values check

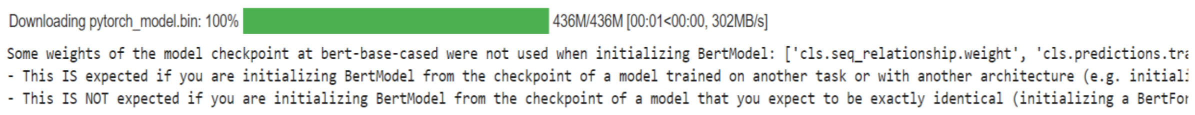
Figure 5.14 refers to the null values check. That means it checks for the attributes which contains the null values



```
((9600, 12), (1200, 12), (1200, 12))
```

Figure 5.15 Training and Testing Data

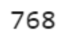
Figure 5.15 represents the training, testing data and validation data. 80 percent of the data is used for training and 20 percent of the data is used for testing.



```
Downloading pytorch_model.bin: 100%                436M/436M [00:01<00:00, 302MB/s]
Some weights of the model checkpoint at bert-base-cased were not used when initializing BertModel: ['cls.seq_relationship.weight', 'cls.predictions.tra
- This IS expected if you are initializing BertModel from the checkpoint of a model trained on another task or with another architecture (e.g. initiali
- This IS NOT expected if you are initializing BertModel from the checkpoint of a model that you expect to be exactly identical (initializing a BertFor
```

Figure 5.16 Loading the model

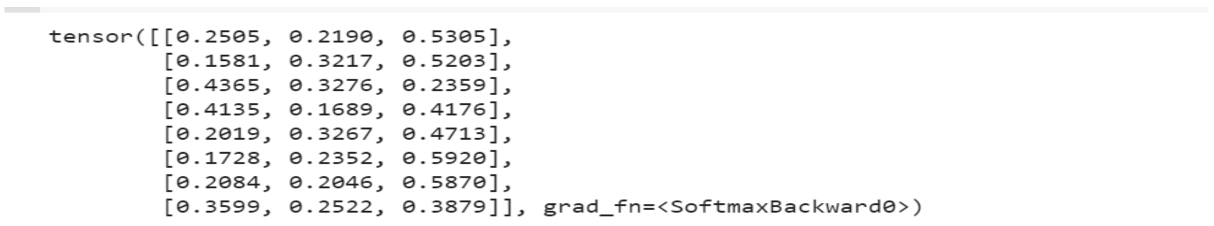
Figure 5.16 refers to the loading of our model. We'll use the basic BertModel and build our sentiment classifier on top of it.



```
768
```

Figure 5.17 The number of hidden units in the feedforward-networks

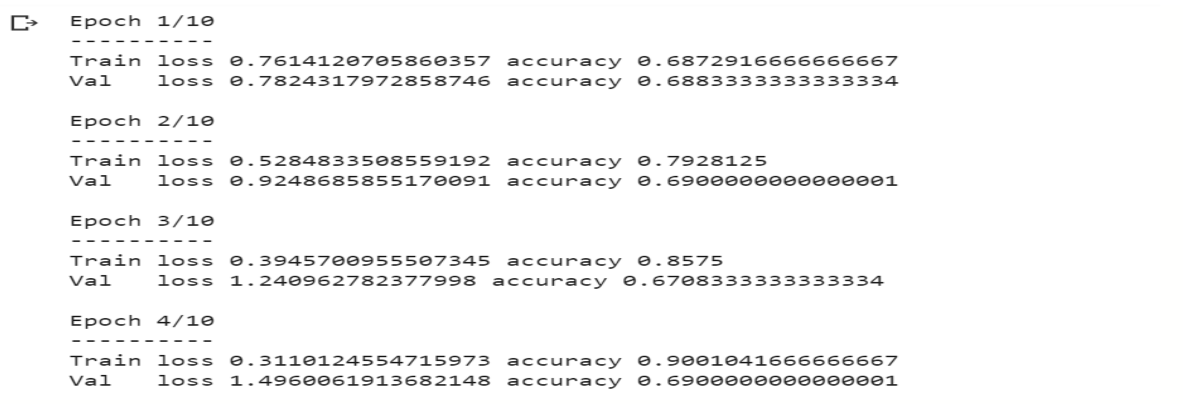
Figure 5.17 refers to the number of hidden units present in the feedforward networks. We have the hidden state for each of our 32 tokens (the length of our example sequence) and 768 is the number of hidden units in the feedforward-networks.



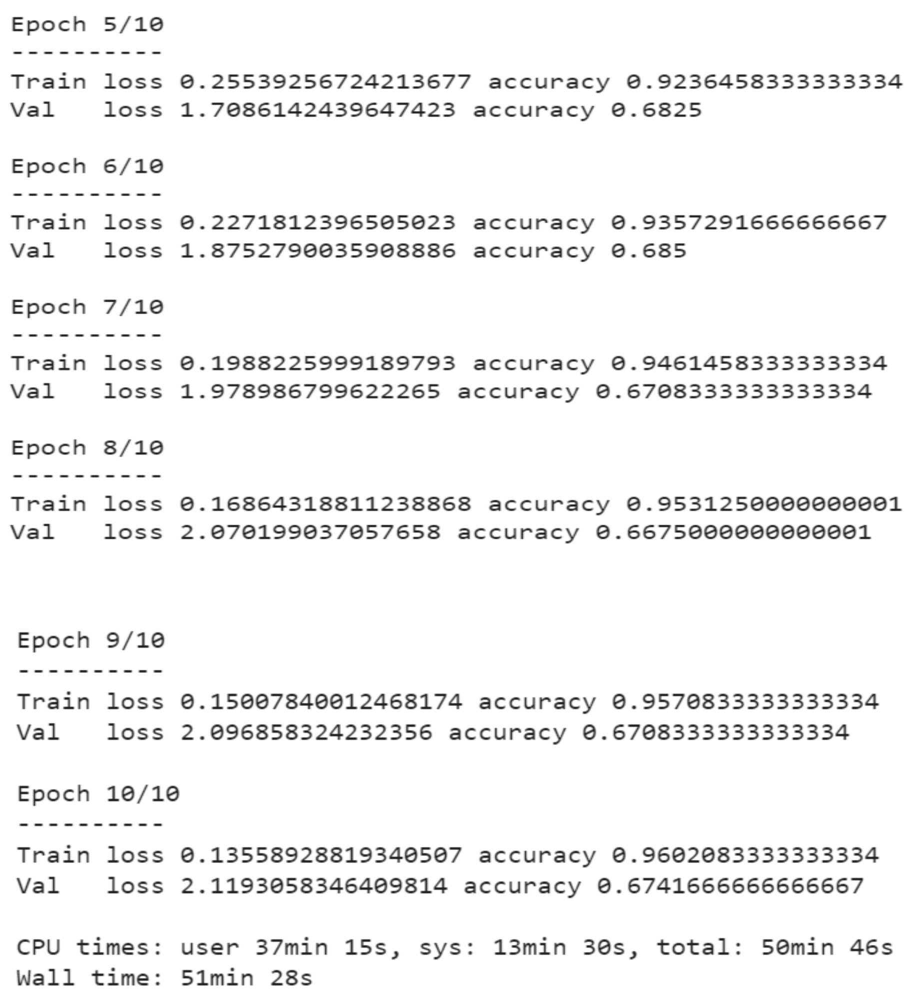
```
tensor([[0.2505, 0.2190, 0.5305],
        [0.1581, 0.3217, 0.5203],
        [0.4365, 0.3276, 0.2359],
        [0.4135, 0.1689, 0.4176],
        [0.2019, 0.3267, 0.4713],
        [0.1728, 0.2352, 0.5920],
        [0.2084, 0.2046, 0.5870],
        [0.3599, 0.2522, 0.3879]], grad_fn=<SoftmaxBackward0>)
```

Figure 5.18 Softmax Function

Figure 5.18 refers to the Softmax function. To get the predicted probabilities from our trained model, we'll apply the softmax function to the output obtained from the output layer.



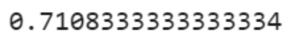
```
Epoch 1/10
----------
Train loss 0.7614120705860357 accuracy 0.6872916666666667
Val   loss 0.7824317972858746 accuracy 0.6883333333333334

Epoch 2/10
----------
Train loss 0.5284833508559192 accuracy 0.7928125
Val   loss 0.9248685855170091 accuracy 0.6900000000000001

Epoch 3/10
----------
Train loss 0.3945700955507345 accuracy 0.8575
Val   loss 1.240962782377998 accuracy 0.6708333333333334

Epoch 4/10
----------
Train loss 0.3110124554715973 accuracy 0.9001041666666667
Val   loss 1.4960061913682148 accuracy 0.6900000000000001
```



```
Epoch 5/10
----------
Train loss 0.25539256724213677 accuracy 0.9236458333333334
Val   loss 1.7086142439647423 accuracy 0.6825

Epoch 6/10
----------
Train loss 0.2271812396505023 accuracy 0.9357291666666667
Val   loss 1.8752790035908886 accuracy 0.685

Epoch 7/10
----------
Train loss 0.1988225999189793 accuracy 0.9461458333333334
Val   loss 1.978986799622265 accuracy 0.6708333333333334

Epoch 8/10
----------
Train loss 0.16864318811238868 accuracy 0.9531250000000001
Val   loss 2.070199037057658 accuracy 0.6675000000000001

Epoch 9/10
----------
Train loss 0.15007840012468174 accuracy 0.9570833333333334
Val   loss 2.096858324232356 accuracy 0.6708333333333334

Epoch 10/10
----------
Train loss 0.13558928819340507 accuracy 0.9602083333333334
Val   loss 2.1193058346409814 accuracy 0.6741666666666667

CPU times: user 37min 15s, sys: 13min 30s, total: 50min 46s
Wall time: 51min 28s
```

Figure 5.19 Epochs

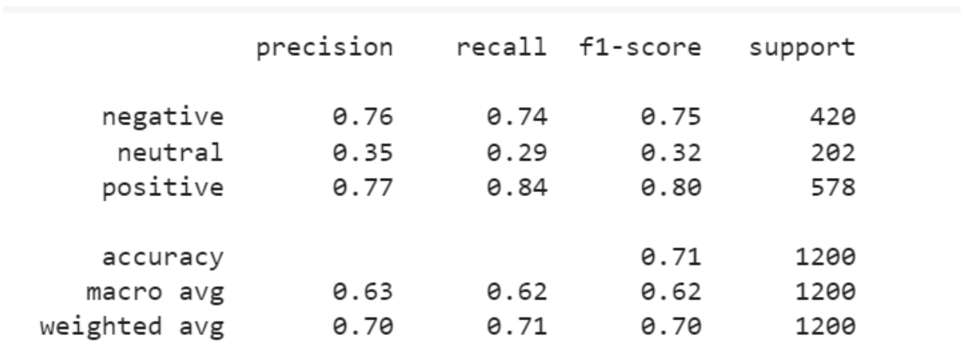
Figure 5.19 refers to the training of the data. Training of the data is known as Epoch. We have used 10 Epochs in our project. The accuracy obtained in the last epoch is 96 percent.



```
0.7108333333333334
```

Figure 5.20 Evaluation

Figure 5.20 refers to the evaluation of our model. We have evaluated our model on the testing data.



```
              precision    recall  f1-score   support

    negative       0.76      0.74      0.75       420
     neutral       0.35      0.29      0.32       202
    positive       0.77      0.84      0.80       578

    accuracy                           0.71      1200
   macro avg       0.63      0.62      0.62      1200
weighted avg       0.70      0.71      0.70      1200
```

Figure 5.21 Classification Report

Figure 5.21 depicts the classification report of our project. Classification report contains precision, recall, f1-score and support.

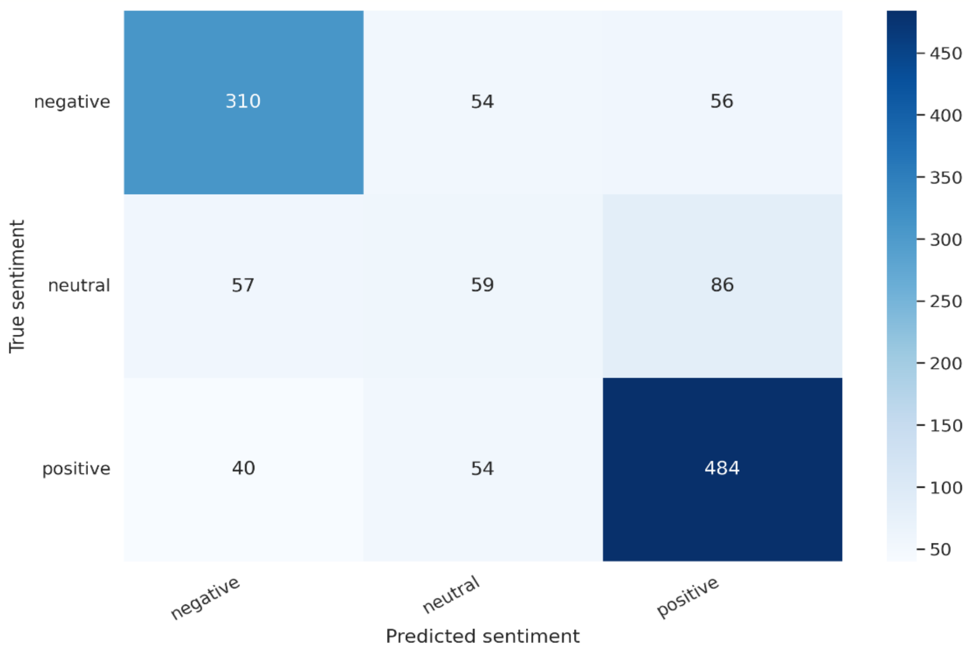
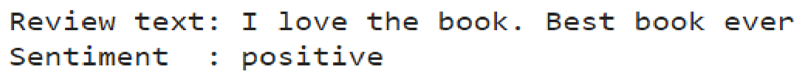


Figure 5.22 Confusion Matrix

Figure 5.22 represents the confusion matrix. A confusion matrix presents a table layout of the different outcomes of the prediction and results of a classification problem and helps visualize its outcomes.



```
Review text: I love the book. Best book ever
Sentiment  : positive
```

Figure 5.23 Sentiment Analysis of a text

Figure 5.23 represents the sentiment polarity of the given text. Since the given text contains two positive wors love and best, the sentiment of that sentence is positive.

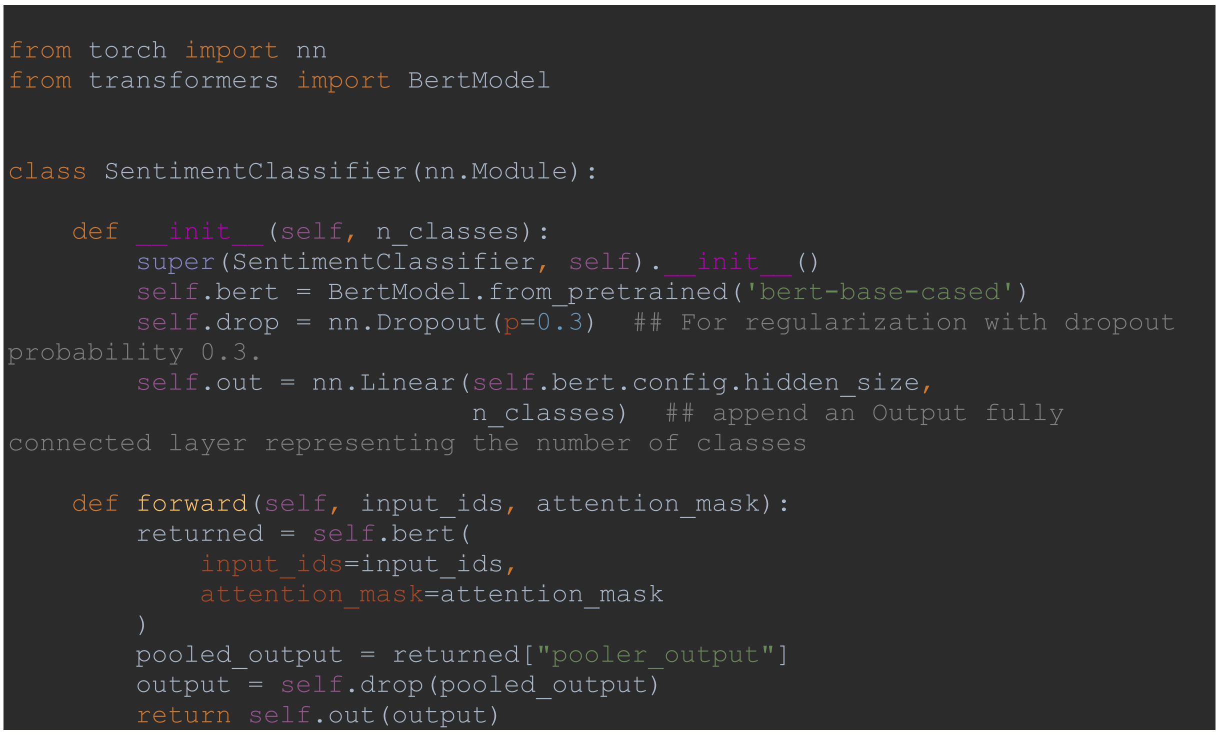
5.4 INTERFACE CODE WITH OUTPUT

app.py

```
import streamlit as st
import numpy as np
from transformers import BertTokenizer,BertModel
import torch
from module import SentimentClassifier
from PIL import Image

image = Image.open('img.png')
st.image(image, caption='Sentiment Analysis')

@st.cache_data

def get_model():
    tokenizer = BertTokenizer.from_pretrained('bert-base-cased')
    model=SentimentClassifier(3).to(device='cpu')

model.load_state_dict(torch.load('best_model_state.bin',map_location='cpu')
)
    return tokenizer,model


tokenizer,model = get_model()
class_names = ['negative', 'neutral', 'positive']

user_input = st.text_area('Enter Text to Analyze')
button = st.button("Analyze")
if button :
    encoded_review=tokenizer.encode_plus(user_input,
                                          max_length=160,
                                          add_special_tokens=True,
                                          return_token_type_ids=False,
                                          pad_to_max_length=True,
                                          return_attention_mask=True,
                                          return_tensors='pt',)
    input_ids = encoded_review['input_ids'].to(device='cpu')
    attention_mask = encoded_review['attention_mask'].to(device='cpu')
    output = model(input_ids, attention_mask)
    _, prediction = torch.max(output, dim=1)
    st.write(f'Review text: {user_input}')
    st.write(f'Sentiment  : {class_names[prediction]}')
```

Figure 5.1 app.py

module.py



```
from torch import nn
from transformers import BertModel


class SentimentClassifier(nn.Module):

    def __init__(self, n_classes):
        super(SentimentClassifier, self).__init__()
        self.bert = BertModel.from_pretrained('bert-base-cased')
        self.drop = nn.Dropout(p=0.3)   ## For regularization with dropout
probability 0.3.
        self.out = nn.Linear(self.bert.config.hidden_size,
                             n_classes)   ## append an Output fully
connected layer representing the number of classes

    def forward(self, input_ids, attention_mask):
        returned = self.bert(
            input_ids=input_ids,
            attention_mask=attention_mask
        )
        pooled_output = returned["pooler_output"]
        output = self.drop(pooled_output)
        return self.out(output)
```

Figure 5.2 module.py

img.png

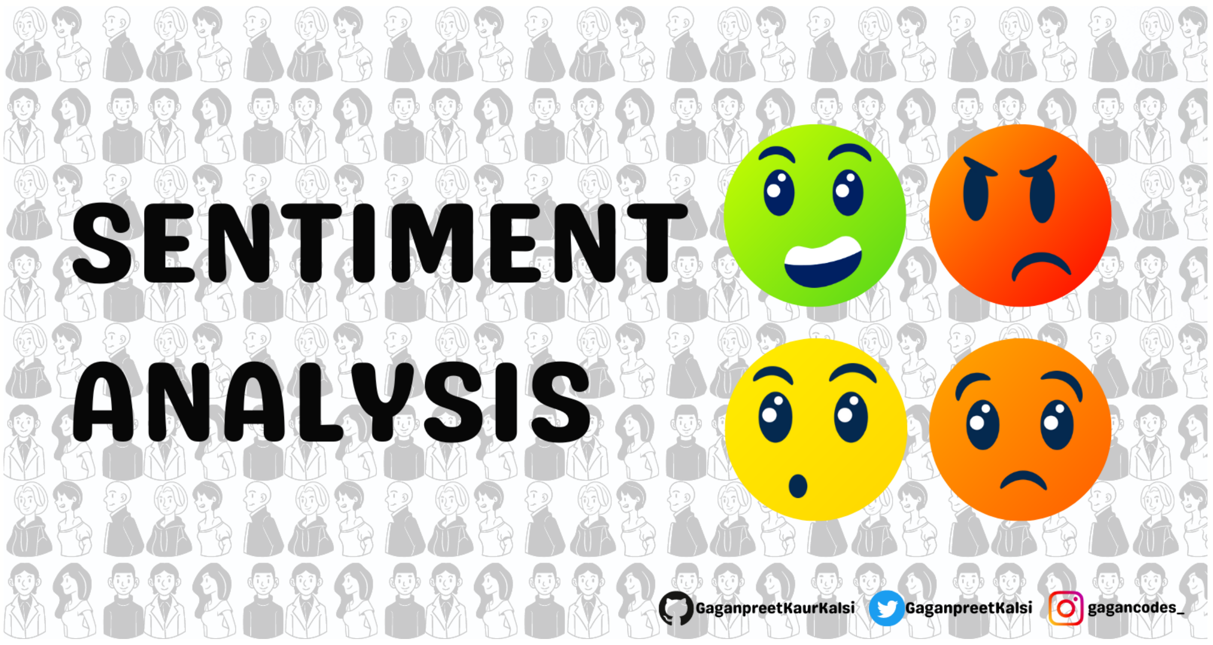
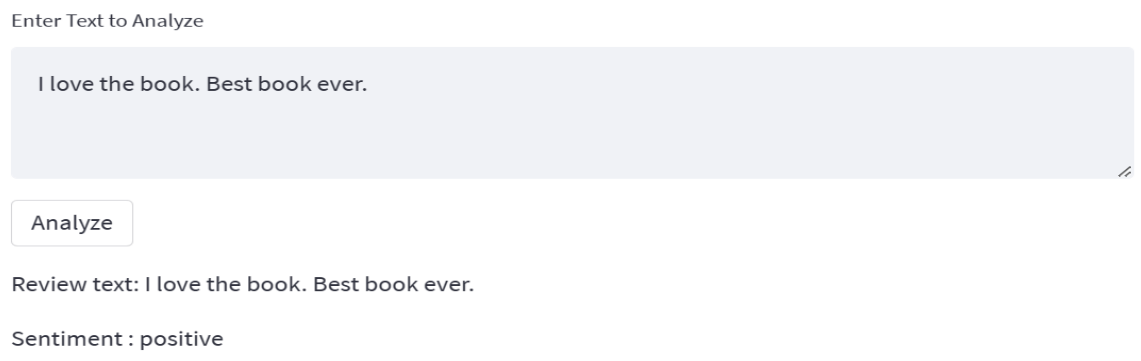


Figure 5.3 img.png

OUTPUT:


Enter Text to Analyze

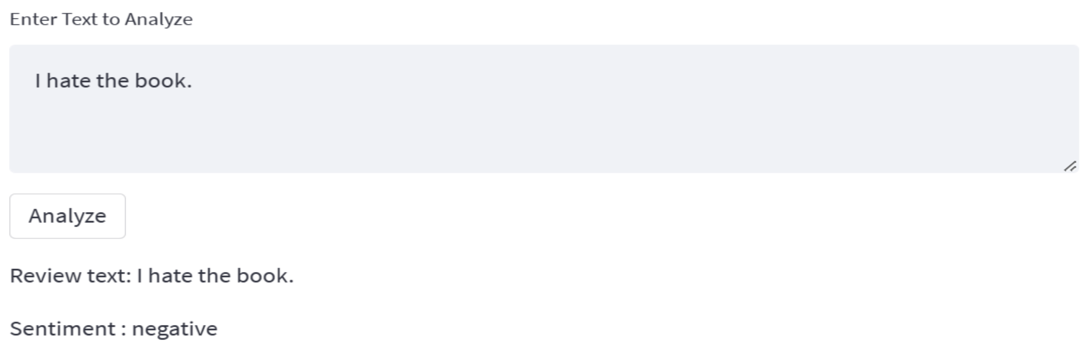
I love the book. Best book ever.

Review text: I love the book. Best book ever.

Sentiment : positive

Figure 5.4 Result-1

Figure 5.4 represents the sentiment polarity of the given text. Since the given text contains two positive wors love and best, so the sentiment of that sentence is positive.


Enter Text to Analyze

I hate the book.

Review text: I hate the book.

Sentiment : negative

Figure 5.5 Result-2

Figure 5.5 represents the sentiment polarity of the given text. Since the given text contains a negative word hate, the sentiment of that sentence is negative.

6. TESTING

Testing is the process of detecting errors. Testing performs a very critical role for quality assurance and for ensuring the reliability of software. The results of testing are used later on during maintenance also. Purpose of Testing: The aim of testing is often to demonstrate that a program works by showing that it has no errors. The basic purpose of testing phase is to detect the errors that may be present in the program. Hence one should not start testing with the intent of showing that a program works, but the intent should be to show that a program doesn't work. Testing Objectives: The main objective of testing is to uncover a host of errors, systematically and with minimum effort and time.

6.1 Testing Strategies

The testing strategies that we have used in our project are unit testing, integration testing, system testing.

Unit Testing

It focuses on smallest unit of software design. In this, testing an individual unit or group of inter related units will be done. It is often done by programmer by using sample input and observing its corresponding outputs. A unit may be an individual function, method, procedure, module, or object. It is a White Box testing technique that is usually performed by the developer.

Integration Testing

The testing of combined parts of an application to determine if they function correctly together is Integration testing. In this, the program is constructed and tested in small increments, where errors are easier to isolated and correct interfaces are more likely to be tested completely; and systematic test approach may be applied. This testing can be done by using two different methods

1. Top-Down Integration Testing.

2. Bottom-Up Integration Testing.

System Testing

System Testing is a type of software testing that is performed on a complete integrated system to evaluate the compliance of the system with the corresponding requirements. System testing detects defects within both the integrated units and the whole system. The result of system testing is the observed behaviour of a component or a system when it is tested. System Testing is a black-box testing.

7. CONCLUSION

Sentiment analysis is the process of using natural language processing, machine learning, and computational linguistics to identify, extract, and quantify subjective information from text. The objective of sentiment analysis is to determine the attitude, opinion, or emotional tone of a speaker or writer with respect to a specific topic, product, service, or brand. In our project, we have used the BERT model to implement the sentiment analysis. BERT model is pre-trained on a large corpus of text, including books, articles, and websites. Based on the results of the sentiment analysis project using BERT model, we will conclude that the BERT model was able to accurately classify the sentiment of the given text data. The project involved the use of state-of-the-art NLP techniques such as pre-training, fine-tuning and attention mechanisms to analyze the text and determine whether it expressed a positive, negative, or neutral sentiment. The BERT model is known for its ability to capture the context and meaning of words and sentences, and has been shown to outperform other models on various NLP tasks including sentiment analysis. Overall, sentiment analysis using BERT model has many practical applications in fields such as marketing, customer service, and social media analysis, and can provide valuable insights into public opinion and attitudes towards products, services, and events

8. FUTURE SCOPE

Pre-trained language models like BERT show clear advantages over classification algorithms, Word2Vec and GloVe models. Therefore, the future scope of our project is **RoBERTa.** RoBERTa stands for Robustly Optimized BERT Pretraining Approach. RoBERTa is a large - scale language model based on the BERT (Bidirectional Encoder Representations from Transformers) architecture. RoBERTa was specifically designed to improve upon BERT's pretraining approach. RoBERTa is trained on a large corpus of text data using an unsupervised, self-supervised learning approach. It is trained to predict the next word in a sequence of text, given all the previous words. This task is called masked language modeling, and it helps the model learn to represent words and their relationships to one another in a way that is useful for a wide range of natural language processing tasks. It has also been used in a variety of applications, including question-answering, sentiment analysis, and language generation.

9.REFERENCES

[1] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT) (Vol. 1, pp. 4171–4186). https://www.aclweb.org/anthology/N19-1423/

[2] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). RoBERTa: A Robustly Optimized BERT Pretraining Approach. arXiv preprint arXiv:1907.11692. https://arxiv.org/abs/1907.11692

[3] Zhang, Y., Sun, X., Yang, J., & Yang, Y. (2020). Chinese BERT: Chinese Pretrained Language Model. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations (EMNLP) (pp. 73–78). https://www.aclweb.org/anthology/2020.emnlp-demos.10/

[4] Sun, C., Qiu, X., Xu, Y., & Huang, X. (2020). Utilizing BERT for Aspect-Based Sentiment Analysis via Constructing Auxiliary Sentence. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (pp. 4406-4415). https://arxiv.org/abs/1903.09588

[5] Chen, Y., Yang, Z., Liu, C., & Sun, M. (2020). A Hierarchical Self-Attention Model for Aspect-Level Sentiment Analysis. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (pp. 5589-5598). https://www.aclweb.org/anthology/2020.emnlp-main.450.pdf

[6] Guo, W., Li, X., & Liu, Y. (2020). Fine-grained sentiment analysis with hierarchical attention-based BERT. Journal of Ambient Intelligence and Humanized Computing, 11(2), 905-916. https://link.springer.com/article/10.1007/s12652-019-01630-7

[7] Wang, Y., Huang, M., Zhao, L., & Zhu, X. (2020). Aspect-Level Sentiment Analysis with BERT-Reinforced Fusion Model. IEEE Access, 8, 214779-214788. https://ieeexplore.ieee.org/document/9218387