PROJECT REPORT ON
SENTIMENTAL ANALYSIS USING BERT

By

A.Viswa Bhaskar	422111
Abhijeet	422101
k.Lohith	422173

Under the guidance of

Dr.K.Himabindu

Department of Computer Science and Engineering

NATIONAL INSTITUTE OF TECHNOLOGY ANDHRA PRADESH

TADEPALLIGUDEM-534101,INDIA

APRIL-2024

DECLARATION

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

A.Viswa Bhaskar	K.Lohith	Abhijeet
422111	422173	422101
Date:	Date:	Date:

CERTIFICATE

It is certified that the work contained in the this is titled “SENTIMENTAL ANALYSIS USING BERT” by “A.Viswa Bhaskar, bearing Roll No: 422111”, “K.Lohtith, bearing Roll No: 422173” and “Abhijeet, bearing Roll No: 422101” has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Dr.k.Himabindu

Computer Science and Engineering
N.I.T. Andhra Pradesh
April 2024

Date:

Place:Tadepalligudem

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany the successful completion of any task would be incomplete without the mention of people who made it possible, whose constant guidance and encouragement crowned our efforts with success. It is a pleasant aspect that I have now the opportunity to express my gratitude for all of them. We owe our sincere gratitude to our project guide Dr.K.Himabindu , Department of Computer Science, National Institute of Technology, Andhra Pradesh, who took keen interest and guided us all along, till the completion of our project work by providing all the necessary information.

We avail ourselves of this proud privilege to express our gratitude to all the faculty of the department of Computer Science and Engineering at NIT Andhra Pradesh for emphasizing and providing us with all the necessary facilities throughout the work.

We offer our sincere thanks to all our friends, fellow mates and other persons who knowingly or unknowingly helped us to complete this project.

A.Viswa Bhaskar	K.Lohith	Abhijeet
422111	422173	422101
Date:	Date:	Date:

List of Figures

List of Symbols and Abbreviations

Notation	**Meaning**
i.e.,	that is
etc.,	etcetera
NLP	Natural language processing
BERT	Bidirectional Encoder Representations from Transformers

ABSTRACT

The paper focuses on a practical perspective of using BERT (a version of transformer which can read the text in forward and backward directions), for the sentiment analysis in the domain of natural language processing. The presentation of the research problem in this project is caused by the necessity of the usage of precision methods of sentiment analysis . We are fixing this by introducing the BERT design and explaining the main steps to be taken while fine-tuning BERT for the sentiment analysis tasks. Here we will also touch the data preprocessing . Performing practical experiments on baseline sentiment analysis datasets, we realized that bert models fine-tuned show better results than machine learning methods and other deep learning system architecture areas. Such findings provide evidence that the BERT model is able to correctly classify emotions, which is a major contribution to the progress of sentiment analysis algorithms. The relevance of this project reposes in its practical implications, as the application of text sentiment detection model based on BERT models will intensify our aptitudes in accurate identifying of sentiment from the text thereby making us capable of interpreting public opinion, customer feedback and social media trends in a more comprehensive manner. This on top of it describes the success cases of deep learning algorithms that attempt to solve challenging NLP tasks, thus enriching our knowledge base regarding their real-world performance.

Table of Contents

CHAPTER-1

INTRODUCTION

Natural language processing (NLP) domain is now filled with the revolutionary BERT that changes the way of sentiment analysis. The bidirectionality of BERT makes it the first model to really analyze all connections and relations between words in a sentence, letting it conduct the analysis simultaneously. Through the whole of such approach, it allows BERT to build a far more extensive understanding of context and meaning in a sentence. BERT takes into account the whole sentence providing the machine to detect sarcasm and irony which are hard for a computer to differentiate because of its complexity. This advanced methodology makes BERT highly effective for tasks such as evaluating sentiment.

Fine-tuning helps deepens BERT further and gives it the ability to determine sentiment in certain domains. For illustration, BERT can be trained to determine the sentiment in the reviews of the clients, posts on social media, or even the financial news articles. Through the specific domain fine-tuning of BERT, the algorithm takes into account the peculiarities of language and sentiment expressions which is characteristic only for the separate domain. For example, BERT is able to dissect social media posts to tell the difference between a sarcastic post compared to those used emojis or non-literal language. This explains why BERT is the most useful in areas where sentiments can be intricate and multidimensional.

Consequently, BERT has become the standard-bearer of sentiment analysis, beating the former-top performing models in a wide range of domains. By virtue of the fact that it is able to catch even the surface meanings and the deep contextual information within a text, BERT could be the best tool for any language-related task that requires a great command of language. That is why BERT can be especially useful in areas where sentiment can be complex and multi-leveled like social networks analysis including sarcastic and ironic messages.

CHAPTER-2

LITERATURE SURVEY

Sentiment analysis has evolved significantly from its initial methods such as bag-of-words and TF-IDF, moving through various stages of machine learning approaches to more complex neural network-based models like LSTM.These developments aimed at improving the understanding of text through sequential data processing. The introduction of Transformers model marked a significant shift in NLP, focusing on self-attention mechanisms that allow for parallel processing and more effective handling of these dependencies. Building on this, BERT developed by Devlin et al. (2018), introduced a method for pre-training a language representation model on a large corpus in a deeply bidirectional way, significantly enhancing the model's understanding of language context.The efficiency of BERT in sentiment analysis has been demonstrated in numerous studies following its release. For instance, Sun et al. (2019) showed that BERT outperforms previous state-of-the-art models on various sentiment analysis benchmarks through its fine-tuning process, which adapts its pre-trained representations to specific tasks like sentiment classification. Further comparative studies by Xu et al. (2019) highlighted BERT's superiority over earlier neural models like LSTM,particularly in processing efficiency and accuracy across diverse datasets and languages.

However, challenges remain with BERT, particularly regarding its resource requirements for training and its occasional struggles with highly domain-specific language. These issues have sparked ongoing research aimed at refining BERT's architecture for greater efficiency and broadening its applicability to more specialized text data.

The self explaining structures improve NLP's this is introduced in 2020 by Zijun Sun, Chun FanF, Qinghong Han. These models not only give correct predictions but also explicit reasons owing to which these decisions are made. Indeed, they can be very beneficial during the process of sentiment analysis, text classification, as well as language generation. They design a trustworthy environment and increase the cooperation of between machine learning systems and human users through explanation of predictions and categorization. The literature thus positions BERT not only as a breakthrough in sentiment analysis but also as a focal point for future developments aimed at enhancing linguistic comprehension in NLP.

CHAPTER-3

METHODOLOGY

3.1 PROBLEM STATEMENT

Traditional sentiment analysis methods often miss the mark due to limitations in capturing context. This report investigates whether BERT Base Uncased, a powerful transformer model, can achieve superior sentiment analysis accuracy by considering context more effectively. Specifically, we explore if BERT Base Uncased can Outperform traditional methods in accuracy, Handle complex sentiment, By demonstrating BERT's effectiveness, this research paves the way for improved understanding of emotions within text, impacting applications like customer service, social media analysis, and marketing.

3.2 MODEL ARCHITECTURE

3.2.1 BERT MODEL

BERT (Bidirectional Encoder Representations from Transformers) appeared in 2018 and was a revolutionary transformer-based model developed by Google AI and radically transformed the area of Natural Language Processing (NLP). And it can be said that this appearance of BERT paved the way for this achievement. This best features in this approach is evinced when it meticulously deals with language comprehension and utilizes both forward and backward context information as an imperative element for the whole translation process.

This report focuses on BERT Base Uncased, a widely used version of the model. "Uncased" refers to the fact that it doesn't distinguish between uppercase and lowercase letters. While this might seem like a limitation, BERT Base Uncased has proven highly effective in various NLP tasks and offers a good balance between accuracy and efficiency.

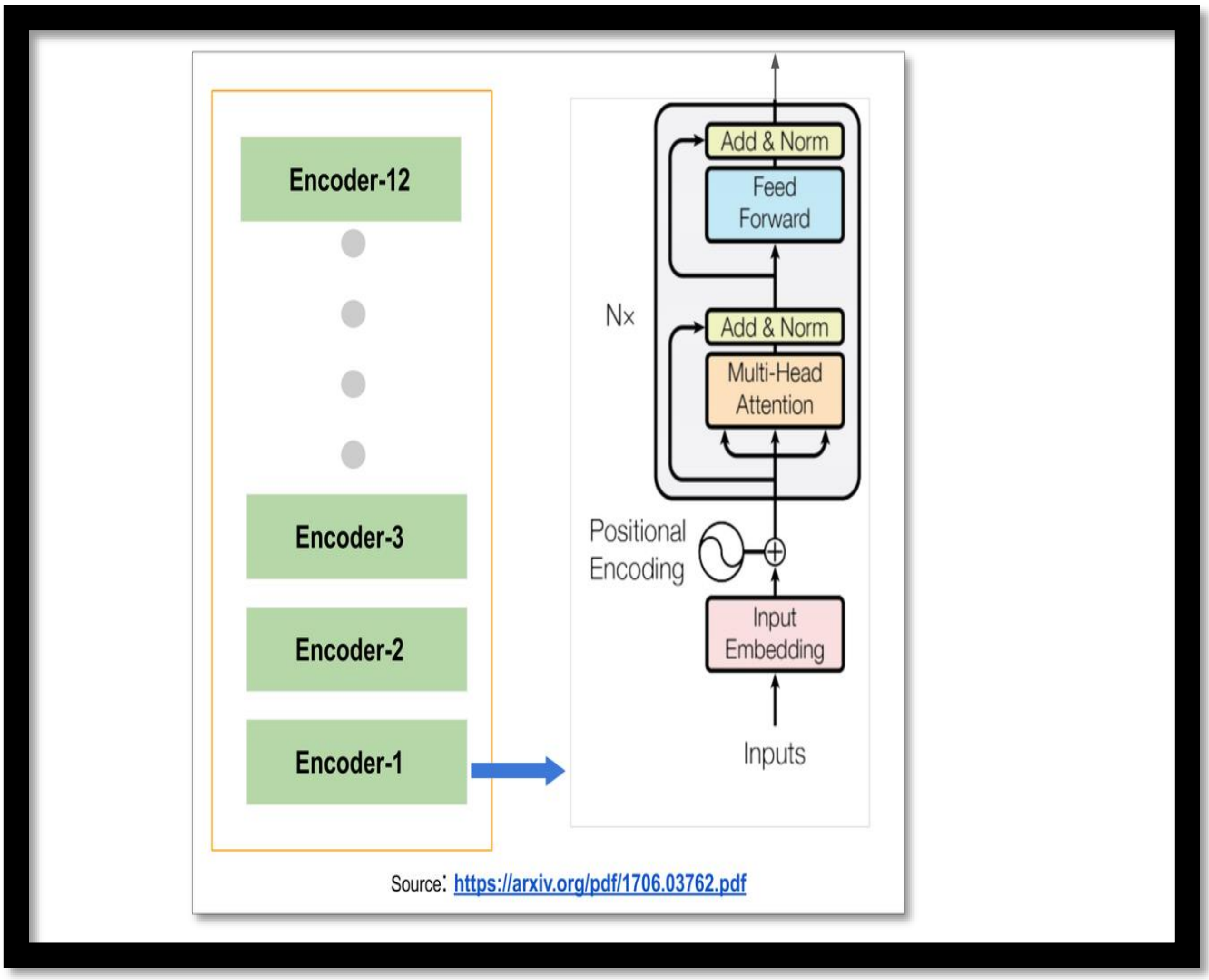


Fig.3.1

INPUT EMBEDDINGS

In the BERT model, input text is first converted into tokens, each of which is represented through a distinct set of embeddings that encapsulate different types of information: Token Embeddings, Segment Embeddings, and Positional Embeddings. Token Embeddings are learned representations for each token in the model's vocabulary, allowing BERT to understand token-specific meanings. Segment Embeddings are used when BERT processes pairs of sentences, as seen in tasks like question answering; these embeddings help the model differentiate between the two distinct sentences in its input. Positional Embeddings are critical because, unlike recurrent neural networks, the Transformer architecture does not inherently process sequence data in order. Thus, Positional Embeddings encode the position of each token within the sentence, providing the model with context about the sequence of words. This multi-faceted embedding approach enables BERT to effectively parse and interpret a wide array of language-based inputs.

MULTIHEAD ATTENTION:

The multi-head self attention in Transformer model such as BERT is the most complex system responsible for the processing of each part of input sequence and it integrates the contextual information effectively.

Queries, Keys, and Values:

Each input token is projected into three vectors:

- Query vectors (Q)
- Key vectors (K)
- Value vectors (V)

These vectors are generated by transforming the input embeddings X using learned weight matrices W^Q, W^K, and W^V, respectively:

$Q = XW^Q$, quad $K = XW^K$, quad $V = XW^V$

Calculating Attention Scores:

Attention Scores = QK^T / sqrt{d_k}

In this place the dk represents the dimensionality of the key vectors and by multiplication with (sqrt{d_k})process gradients stabilization takes place during training.

Softmax Normalization:

The raw scores are then passed through a softmax function to normalize them into a probability distribution:

Softmax Scores = softmax{QK^T / sqrt{d_k}}

This step ensures that the scores sum to one, turning them into weights that define the importance or relevance of each token in the sequence relative to the query token.

Output Computation:

The output of the attention mechanism for each token is then computed as a weighted sum of the value vectors, where the weights are defined by the softmax-normalized attention scores:The final output attention layer fulfills a similar role where the single output consists of a weighted sum of all tokens' value vectors, computed according to softmax-normalized attention scores as the weights.

Result = Softmax(Scores).V

This non-contiguous-type concatenation, although very effective in other aspects, does not enjoy this effective summarization behaviour of repeatable transformer layers that output tokens to which the sum of attention scores is higher.

Multi-Head Attention:

In multi-head attention, the above process is paralleled across \(N\) different attention 'heads', each with its own set of weight matrices $W^Q{}_n$, $W^K{}_n$, and $W^V{}_n$:

$Q_n = X \times W^Q{}_n$, $K_n = X \times W^K{}_n$, $V_n = X \times W^V{}_n$

$\text{Head}_n = \text{softmax}\ \{Q_n K_n{}^T / \text{sqrt}\{d_k\}\}\ V_n$

The outputs of each head are then concatenated and once again linearly transformed:

Multi-Head Output = $\text{Concat}(\text{Head}_1, \text{Head}_2, \ldots\ldots, \text{Head}_N) \times W^O$

where W^O is another learned weight matrix.

This multi-head setup allows the model to capture various aspects of the input information from different representational spaces, enhancing its ability to interpret complex contextual relationships within text. This layered and repetitive attention process is fundamental to BERT's performance across diverse NLP tasks.

FEED FORWARD Network:

Within each Transformer block of the BERT model, the feed-forward network (FFN) serves as a critical component for processing the sequence of embeddings produced by the multi-head self-attention mechanism. Each FFN is applied identically and independently to each position in the sequence, which means the same neural network is applied to each token's output from the attention mechanism, ensuring that the model can maintain its ability to handle variable-length input sequences.

Feed-Forward Network Architecture:

The forward network is composed of two linear transformation of ReLU (Rectified Linear Unit) activations in between.

$FNN(x) = max(0, xW1 + b1)(W2 + b2)$.

Here x is the information given (input) to the feed-forward units of the attention sub-layer or from another block of feed-forward layer. W1 and W2 are the weight matrices and b1 and b2 are the bias vectors for the line the transformations respectively. The ReLU activation function, which is signified by (max 0, z), adds nonlinearity to the model, so the model can learn more complex features of the data.

Layer Normalization and Residual Connections:

Likewise, the multi-head self-attention and feed-forward networks of each sub-layer of the Transformer block are enclosed in residual connection with a layer normalization afterwards.

Output = Layer Norm (x + Sublayer(x))

The input to this layer is word embedding, and the output is either self-attention block or feed-forward network. is about a new function x+, that refers to a residual connection which actually solves the issue of the vanishing gradient problem by letting gradients pass through the networks directly. Following the add, layer normalization is performed which normalizes the data throughout the features, making sure mean and variance are consistent, therefore training is easy.

Layer normalization is computed as:

$LN(x_i) = \gamma(x_i - \mu / \sigma) + \beta$

Here, x_i is the input, μ and σ are the mean and standard deviation of the inputs, and γ and β are learnable parameters of the normalization, adjusting the scale and shift of the normalization process, respectively.

This combination of feed-forward networks, residual connections, and layer normalization ensures that each Transformer block can effectively learn from the inputs, adjust based on the complexity of the model, and maintain stable training across deep networks. These elements are essential for the high performance of models like BERT on a variety of complex language understanding tasks.

Output:

For classification tasks, the output from the BERT model is typically taken from the output corresponding to the "[CLS]" token, the first token in the sequence. This token is specially designated for classification tasks. The output vector of this token is passed through additional layers (e.g., a linear layer for classification) depending on the specific task.

Training:

BERT's training involves two main tasks:

Masked Language Model (MLM): The goal of this undertaking is purely to scramble the tokens of the input wording such that the model can predict the original values of the masked words (as determined by the tokens' context).

Next Sentence Prediction (NSP): The model classifies the given pairs of sentences into the ones where the second sentence is grammatically connected to the previous sentence in the original text and the ones where the connection is unclear or non-existent.

These pre-training tasks enable BERT to effectively understand language context and relationships, making it versatile for various downstream NLP tasks.

3.2.2 MODEL COMPONENTS

BERT Model (self.bert):

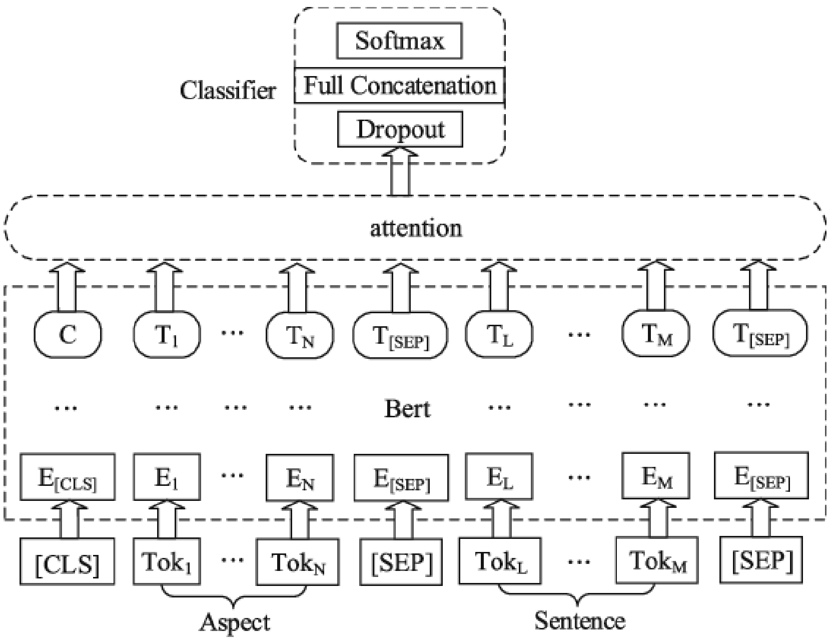


Fig.3.2

Purpose: Acts as the feature extractor by converting input text into contextualized embeddings.

Operation: Processing input tokens by the BERT model happens through multiple layers of transformer blocks. Each block applies self-attention, which corresponds to the modification of the representation of each word by gathering all the information from other words in the sentence. This enables the model to consider context not only from either the preceding or the following words but from both effectively (hence "bidirectional").

Self-Attention: Each word is transformed into queries (Q), keys (K), and values (V) using learned matrices. The attention scores are computed as softmax((QK^T)/sqrt(d_k)) x V, where d_k is the dimension of the keys. This formula determines how much each word in a sentence contributes to the representation of another word.

Output: BERT outputs two main types of embeddings: the last hidden state (sequence of embeddings for each token) and the pooled output (pooler_output). The pooler_output typically takes the embedding of the first token (the [CLS] token), which is expected to contain the aggregate representation useful for classification tasks.

Fully Connected Layer (self.fc):

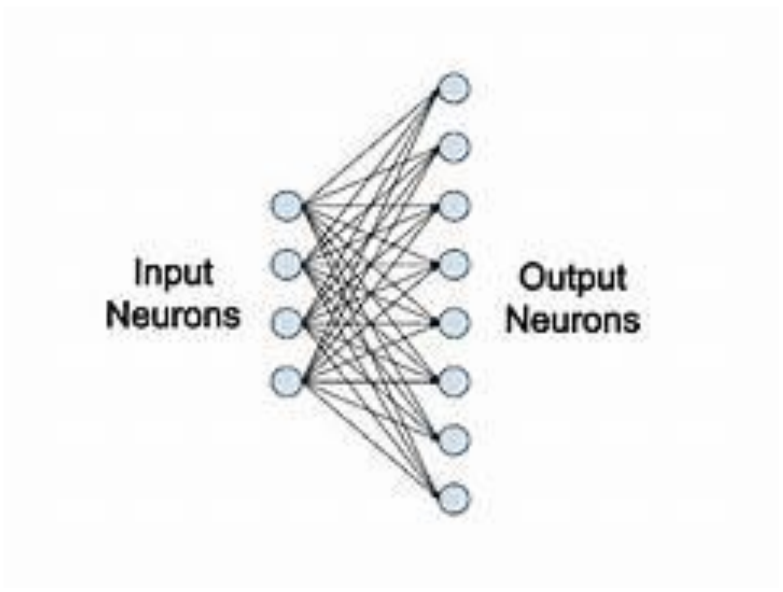


Fig.3.3

Purpose: Maps the high-dimensional output from BERT to the number of classes in the target dataset.

Operation: This is a linear transformation that projects the BERT pooled output to a vector whose length equals the number of classes.

Math Involved:

The linear layer computes $y = Wx + b$, where x is the input vector (BERT pooled output), W is a weight matrix, and b is a bias vector. Each element of the output vector y corresponds to a class score.

Attention layer (self.attention_layer):

It is initialized using nn.linear .

The input size (in_features) is set to 'self.bert.config.hidden_size' , which corresponds to the hidden size of the BERT model.

The output size (out_features) is set to 1, indicating that it outputs a single attention score for each token.

During the forward pass of the model, the self.attention_layer receives the hidden states of the tokens as input.

It applies a linear transformation to the hidden states, effectively mapping them to a single attention score for each token.

The output of this layer represents the attention scores for each token in the input sequence.

The attention scores produced by the linear layer are typically passed through an activation function to ensure non-linearity and interpretability.In this case, the sigmoid activation function (nn.Sigmoid()) is applied to the output of self.attention_layer.The sigmoid function squashes the attention scores to the range [0, 1], allowing them to represent probabilities.

Forward Method

Inputs:

input_ids: Tensor containing sequences of token ids for each text sample.

attention_mask: Tensor that specifies which tokens are actual data and which are padding.

token_type_ids: Tensor indicating the segment of each token (used in tasks like question answering where inputs are composed of two separate sequences).

Outputs:

The forward method feeds the inputs through the BERT model and then passes the pooler_output to the fully connected layer. The output is a vector of logits corresponding to the class scores.

3.2.3 Training Components

Classifier Initialization:

Optimizer (optimizer):

Adam Optimizer: A process during which the weights of the network are tuned upward or downward. The procedures are carried out in accordance with the gradients of the loss. Adam was an adaptive learning rate algorithm, therefore it distracted individual learning rates for different parameters.

Math: Adam combines the features of two other directions of the performance improvement with respect to stochastic gradient descent.Specifically

Increasing the learning performance that adjusts for sparse gradients problems is possible via maintaining per-parameter learning rate in AdaGrad.

Root Mean Square Propagation (RMSProp) also makes use of variable and specific learning rates for parameters which are picked based on the average of a few of the recent ascent speed of the gradients (i.e. using moving averages of the squares of the gradients to normalize the gradient).

Loss Function (criterion):

Cross-Entropy Loss: commonly used in the events of multi-label classification problems. It deals with the accuracy of a classifier whose production phases a value probability between zero and one

Math: Cross entropy loss has a negative likelihood calculation of the logarithm of the correct class, which mathematically can be shown as -sum (y_i * log_2(p_i)) for the i-th class label, which is an indicator with binary values, 0 or 1, and p_i is the predicted probability of the class.

CHAPTER-4

EXPERMENTAL SETUP AND RESULTS

4.1 DATASET OVERVIEW:

The Stanford Sentiment Treebank is a corpus, which buildings provide fully modeling of parse trees that help for a complete study of sentiment compositional effects in a language. The corpus is formed on top of Pang and Lee’s (2005) dataset which is comprised of 11,855 while sentences were extracted from the movie reviews. Tested by first parsing with the Stanford parser and then collecting just a distinct 215,154 phrases out of these parse trees, which were further annotated by 3 human judges. The binary classification task run on the SST-2 or SST binary refers to full sentences that have been parsed, which include either negative or somewhat negative, or somehow positive or positive and neutral sentences discarded.

It is in the form of Dataset Dict which is already having train , test and validation splits in itself .It has three features i.e label ,Sentence, idx . Here the Sentence is the raw text , label is the int value(0 or 1) representation of the sentiment of the sentence and the idx is the index value of the sentence in the test train split .The train data size is 67349, and test and validation sizes are 872 and 1821 respectively .

4.2 SYSTEM WORKFLOW

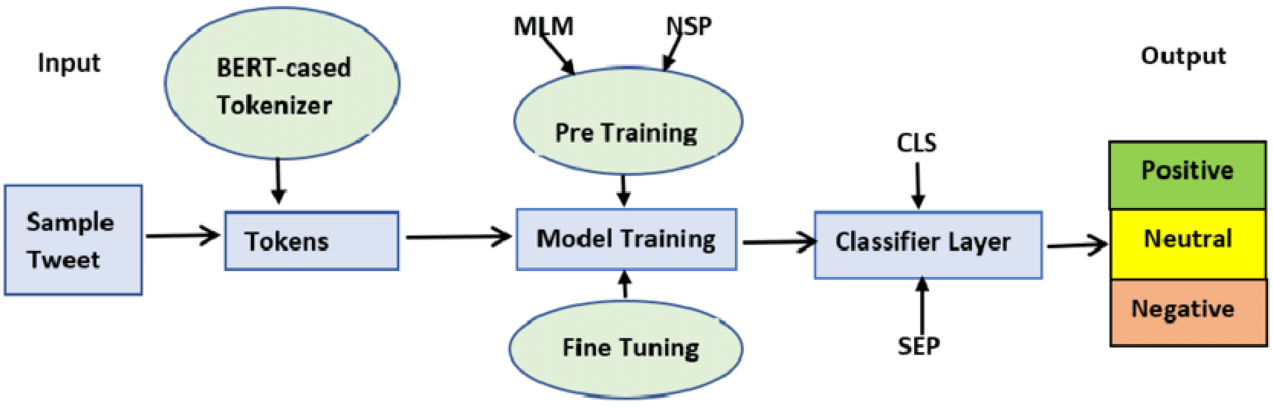


Fig.3.4

Here the sample input text is passed to the model's tokenizer which converts it into tokens including special tokens like CLS and SEP and then these are converted into token_ids ,attention masks and segment_ids and is set to Pytorch format and data loaders are created and passed to the model and the pooler output is collected and passed through the fully connected linear (classification layer) and the corresponding predicted sentiment is produced .

4.3 DATA PREPROCESSING

TOKENIZATION :

Tokenization and In particular, fixed-length text tokens models (e.g., BERT) are undoubtedly the lexical units on which all language models are based.

The fact that tokenization is a part of the premise of Natural language processing models eye was opened widely, particularly when the tokenization of the fixed sequence is taken into account, like when we think of BERT models.

As far as the basic preprocessing applies, tokenization paves the way to natural language processing and particularly gets the models 100% involved which do their job with regular sequence of tokens like BERT.

Tokenization is in itself no surprise because it’s the fundamental step any NLP task includes, which requires using token in the form of a fixed sequence as BERT series does.

The tokenization process generally involves the following steps:

Subword Tokenization: Each word is further broken down into subwords or pieces using a predefined vocabulary or tokenizer. This step allows BERT to handle rare or unseen words by representing them as combinations of more common subwords. Popular tokenization algorithms like WordPiece or Byte-Pair Encoding (BPE) are often used for this purpose.

Special Tokens: Special tokens are added to the tokenized sequence to denote the beginning and end of a sentence, as well as to indicate padding or unknown tokens. In the case of BERT, a special [CLS] token is also inserted at the beginning of the sequence to represent the entire input text for classification tasks.

Padding and Truncation: Finally, the tokenized sequences are padded or truncated to ensure a fixed length (512 tokens), as BERT requires inputs of uniform size. Padding entails the addition of padding tokens, at the conclusion of sequences with longer sequences being trimmed to fit within the models maximum length.

The BERT tokenizer is responsible, for performing the tokenization.

FORMATTING:

Preparing text data, for BERT involves formatting sequences into the input structure. This crucial step includes converting tokens to token IDs and generating attention masks among components.

Conversion to Token IDs: Once the text has been tokenized into words or subwords, each token is converted into a numeric ID. These IDs correspond to the token’s position in BERT’s predefined vocabulary. This vocabulary contains a fixed list of subwords, learned from a large corpus of text data, where each subword has a unique identifier. By converting tokens to these IDs, the textual input is transformed into a numerical format that the model can process.

Attention Masks: Since BERT processes input data in fixed-size batches, sequences in the same batch must have the same length. This is achieved through padding shorter sequences with a special [PAD] token. However, these padding tokens should not influence the model’s understanding of the actual sequence content. To handle this, an attention mask is created for each sequence, indicating which tokens are real and which are padding. The attention mask has the same length as the input sequence, where each position contains a binary value — typically 1 for real tokens and 0 for [PAD] tokens. This mask tells the model to only pay attention to the positions with a value of 1 during the self-attention calculations within the Transformer layers.

Segment IDs: For tasks involving pairs of sentences (e.g., question answering), BERT also requires segment IDs to distinguish between the two different sentences. A segment ID is a sequence of binary values (usually 0s and 1s), which indicates whether a token belongs to the first sentence or the second sentence in paired sentence tasks.

Formatting: The final formatting step involves combining these elements—token IDs, attention masks, and optionally segment IDs—into a structured format that BERT can process. Each element of the input data (a single training example) typically consists of a list of token IDs, a corresponding attention mask, and segment IDs if necessary. These elements are batched together into pytorch tensors, which can then be fed into the model during training or inference.

By properly formatting these sequences, BERT is able to efficiently process input data, focusing on meaningful content and effectively handling different types of NLP tasks. This structured approach to handling input data is essential for leveraging the full capabilities of the BERT model.

DATA LOADERS:

We Fix Batch_size as 64 and we make the data loaders for train data ,evaluation data and test data using pytorch which we give as input to our model while training , evalauation and testing .

4.4 IMPLEMENTATION DETAILS:

We Know that for hyper parameter tuning low learning rate is good for achieving faster and better result hence fix the lr as 1e-5 . we have kept the number of epochs as 3 as from 3 epochs the epochs vs accuarcy is nearly linear and the execution time is increasing hence for optimum result number of epochs is set as 3 . we have trained our model on SST-2 dataset which is widely used for text classification tasks and run on T4 Gpu which is provided by Google collab and it took around 35min for training the model .

4.5 EVALUATION METRICS

4.5.1 ACCURACY:

When using the BERT model for tasks such as sentiment analysis, document classification, or any other form of categorical data prediction, accuracy is often used to measure how well the model performs. In these contexts, accuracy is defined as the ratio of correctly predicted observations to the total observations. For a BERT model, this means evaluating how accurately the model can assign the correct label to text inputs based on its learned representations.

Mathematical Formula for Accuracy:

The formula for accuracy is straightforward and is expressed as:

Accuracy =(Number of Correct Predictions/Total Number of Predictions)

To break it down further:

Number of Correct Predictions: This is the count of instances where the predicted label matches the true label of the data point.

Total Number of Predictions: This is the count of all predictions made by the model (i.e., the size of the test dataset).

For this model we are getting the test accuracy as 0.9208

4.5.2 PRECISION SCORE:

Precision for this kind of classification models as BERT is the proportion of true positives that were actually correctly identified. It is one of the main indicators whenever the consequences of false positive predictions are significant.

Mathematical Formula for Precision:

Precision measures the ratio of true positives (TP) to the sum of true positives and false positives (FP).

Precision=TP/TP+FP

Where:

TP (True Positives): The number of accurate positive predictions from the model.

FP (False Positives): The number of wrong predictions in which the model predicted the class "positive" but the actual class was "negative".

For this model we are getting the Precision as 0.9255

4.5.3 RECALL:

Recall, also known as sensitivity or true positive rate, is a critical performance metric in the field of machine learning, especially for classification problems. It measures the model's ability to correctly identify all relevant instances or conditions from the actual positives available during testing. This metric is particularly vital in scenarios where missing a positive instance carries significant consequences.

In classification tasks, recall answers the question, "Out of all the actual positives, how many were correctly identified by the model?" It is a crucial measure when it is important to capture as many positives as possible, even if it means accepting more false positives. High recall is essential in situations where the cost of a false negative (failing to identify a positive) is much higher than the cost of a false positive.

Mathematical Formula for Recall:

Mathematical Formula for Recall

TP (True Positives): The number of times the model correctly classified instances labelled as positive.

FN (False Negatives): Erroneous classifications of positive examples as negative by the model.

Recall=TP / TP+FN

For this model we are getting the Recall as 0.9203.

4.5.4 F1 SCORE:

The F1 score is an evaluation metric mostly used in machine learning to measure accuracy, especially in cases where data is imbalanced class wise. It explicitly demonstrates the efficiency of the model as it harmonizes the precision and recall by their harmonic mean. In assessing the overall performance of a model, the F1 score can serve as a useful indicator to capture forecasts that involve false positives as well as false negatives.

The F1 score consists of two metrics: precision and recall, while their harmonic mean is taken at the same time. Accuracy (specificity) defines the proportion of correct identifications among the positive ones, recall (also known as sensitivity) – is the proportion of identifications among the correct positives.

F1=2 (Precision×Recall / Precision+Recall)

For this model we are getting the F1 Score as 0.9206

4.5.5 SELF EXPLAINABILITY:

There are some methods to make a model into self explainable one such technique is done below:

Attention Visualization: Many NLP models, especially those based on transformers like BERT, have attention mechanisms. These models can visualize which parts of the input text are most relevant for making predictions. The attention weights can be visualized as heatmaps, with brighter colors indicating higher attention.

Input Text: She feared seeing them happy again, knowing she really was nothing more than a disposable stand-in until they were able to be together again.

```
Predicted Probabilities:tensor([[0.9283, 0.0717]], device='cuda:0', grad_fn=<SoftmaxBackward0>)

Attention Weights: tensor([[0.5065, 0.5751, 0.5907, 0.5270, 0.5159, 0.5379, 0.5727, 0.6271, 0.5326,
        0.5352, 0.5313, 0.5353, 0.4994, 0.5603, 0.4638, 0.4078, 0.4896, 0.4992,
        0.3486, 0.5034, 0.3797, 0.4726, 0.4598, 0.3532, 0.5557, 0.5488, 0.5830,
        0.5489, 0.5595, 0.5193, 0.5856, 0.5125, 0.5137]], device='cuda:0',
       grad_fn=<SigmoidBackward0>)

Decoded Tokens: ['[CLS]', 'she', 'feared', 'seeing', 'them', 'happy', 'again', ',', 'knowing', 'she', 'really', 'was', 'nothing', 'more', 'than', 'a', 'di', '##sp', '##osa', '##ble', 'stand', '-', 'in', 'until', 'they', 'were', 'able', 'to', 'be', 'together', 'again', '.', '[SEP]']
```

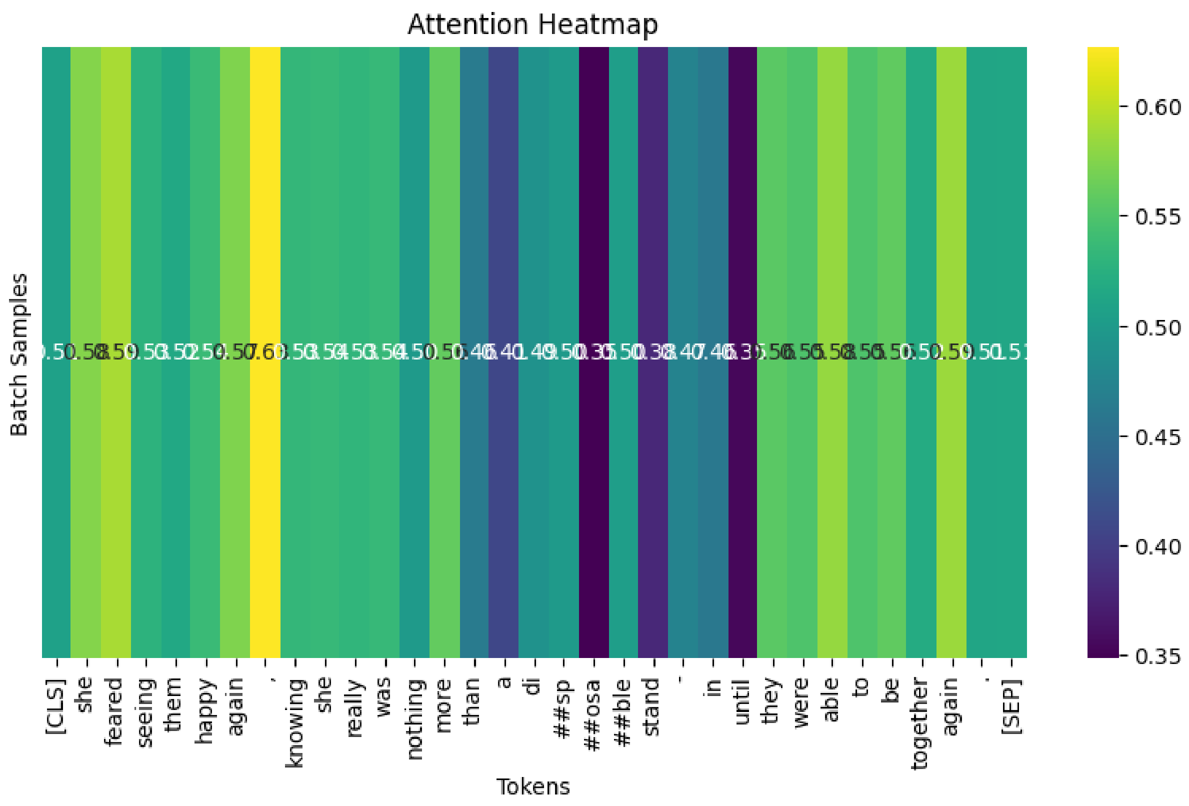


Fig.3.5

4.5.6 CONFUSION MATRIX:

A confusion matrix is a tool used for performance measurement in machine learning, specifically in classification tasks, but not limited to, to evaluate the performance of the classification model. It constitutes the prediction (actual) and the class of the dataset.

The confusion matrix is a square matrix that has actual classes in rows and predictable classes in columns. In the matrix cells, a number stands for the number of instances that pertain to correspondent actual and predicted categories.

The correct predictions represented at the main diagonal of the matrix are matched by the actual class and the predicted class. Discretionary elements in diagonal represent wrong predictions therefore these are situations where the predicted class differs from actual class.

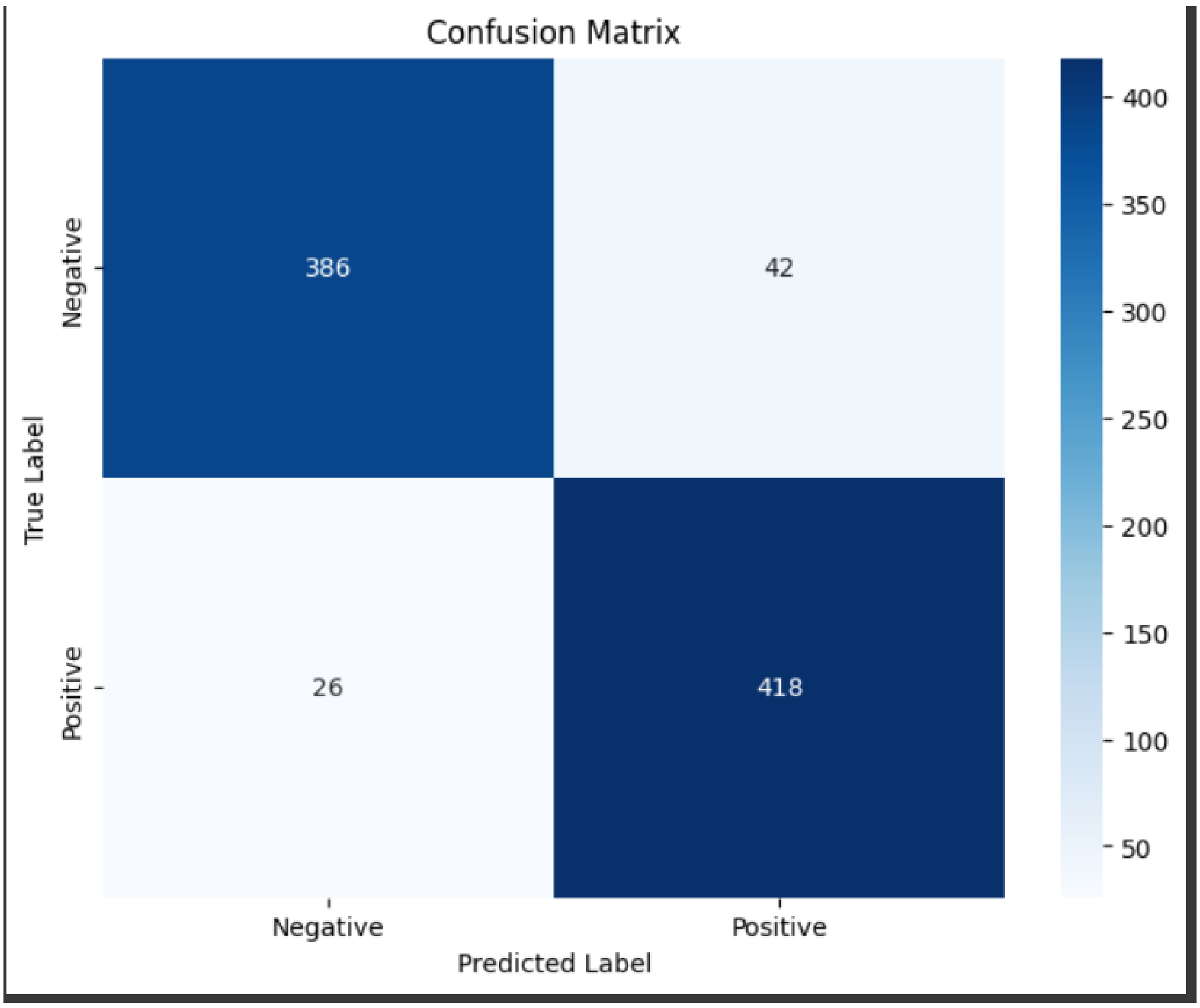

Confusion Matrix
True Label
Negative
Positive
386
42
26
418
Negative
Positive
Predicted Label
400
350
300
250
200
150
100
50

Fig.3.6

4.5.7 SOTA MODEL :

For the current State Of The Art model on sentimental analysis which was tested on the same SST-2 Dataset the results are shown below

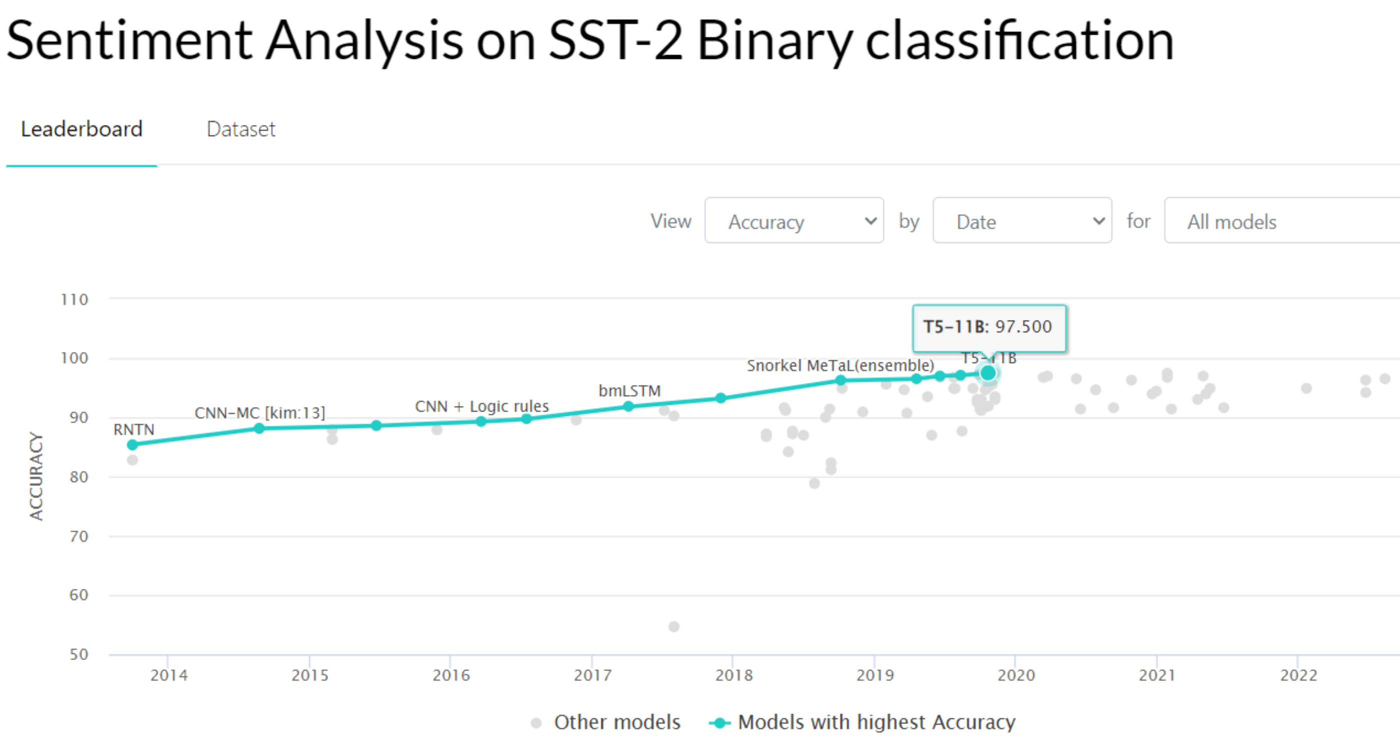


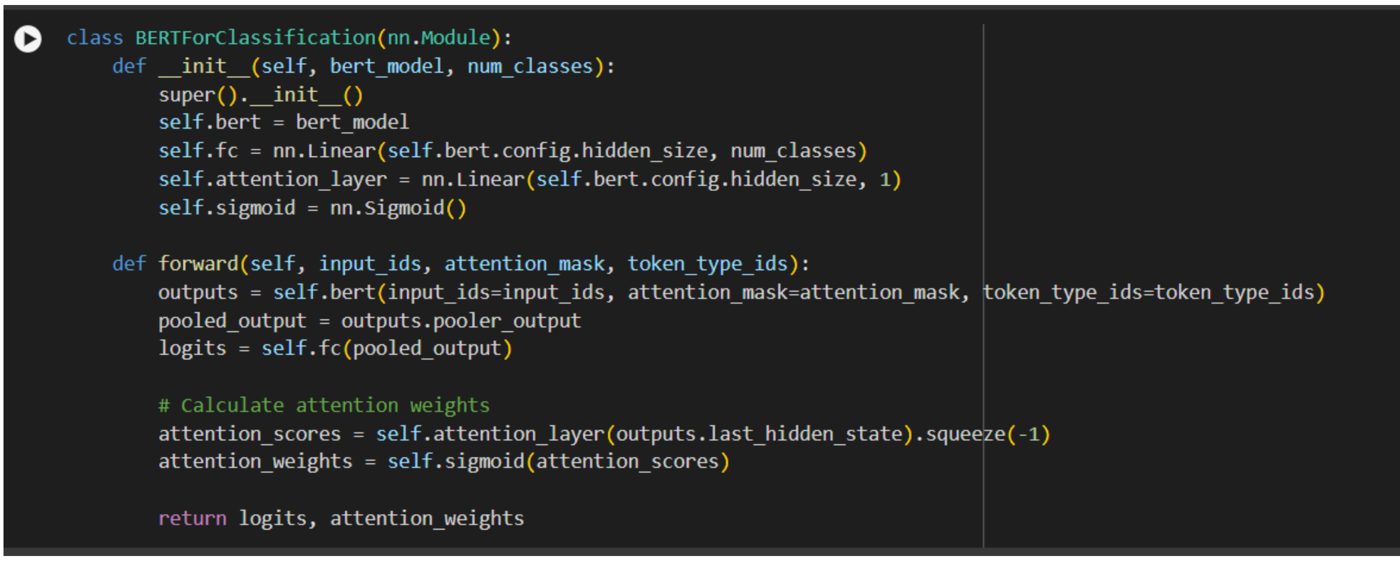
Fig.3.5

From:https://paperswithcode.com/sota/sentiment-analysis-on-sst-2-binary?tag_filter=4

As we can see that the accuarcy of the latest model which was build using transformers is 97.5%.

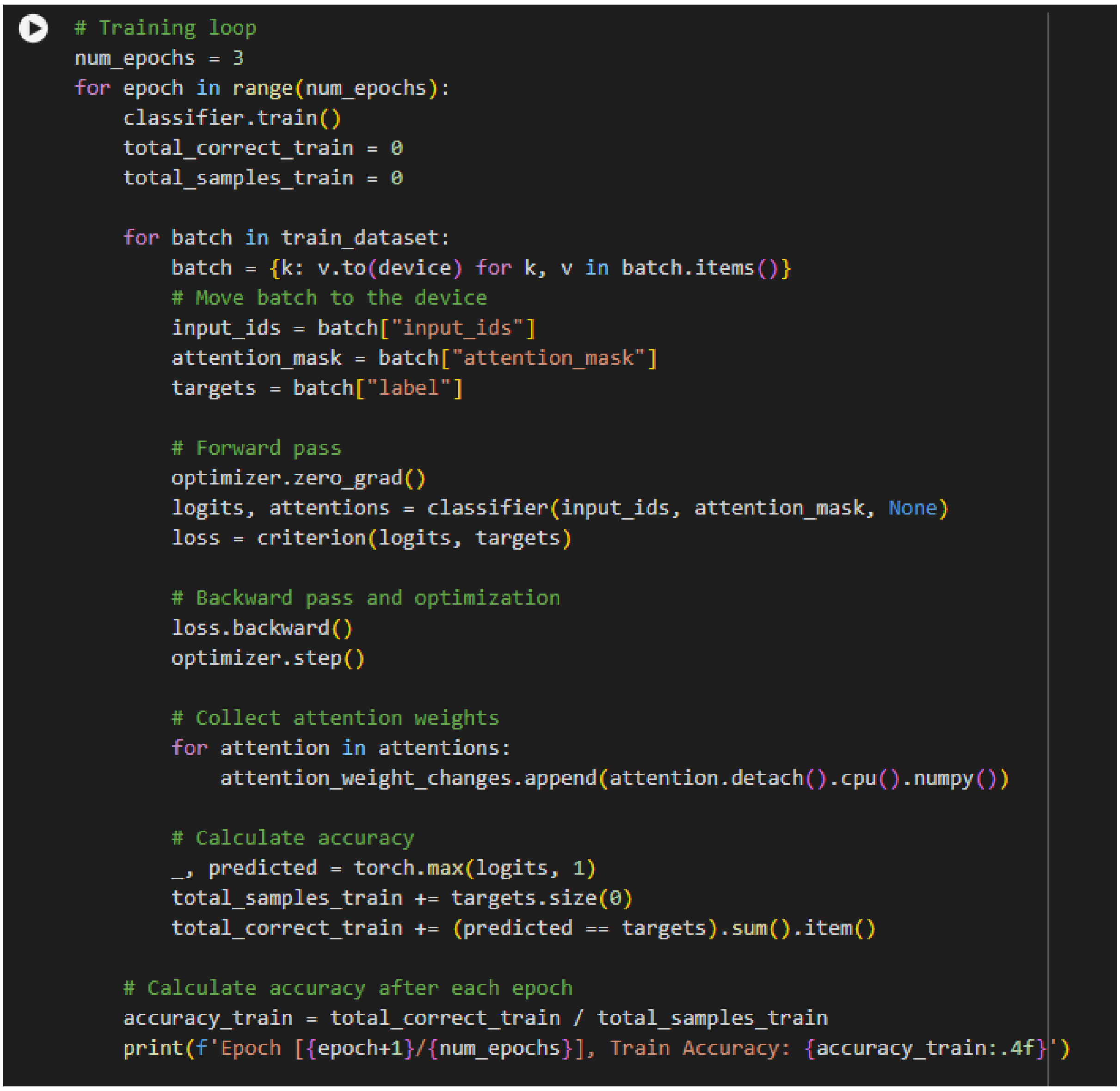
CODE SNAPSHOTS

MODEL:

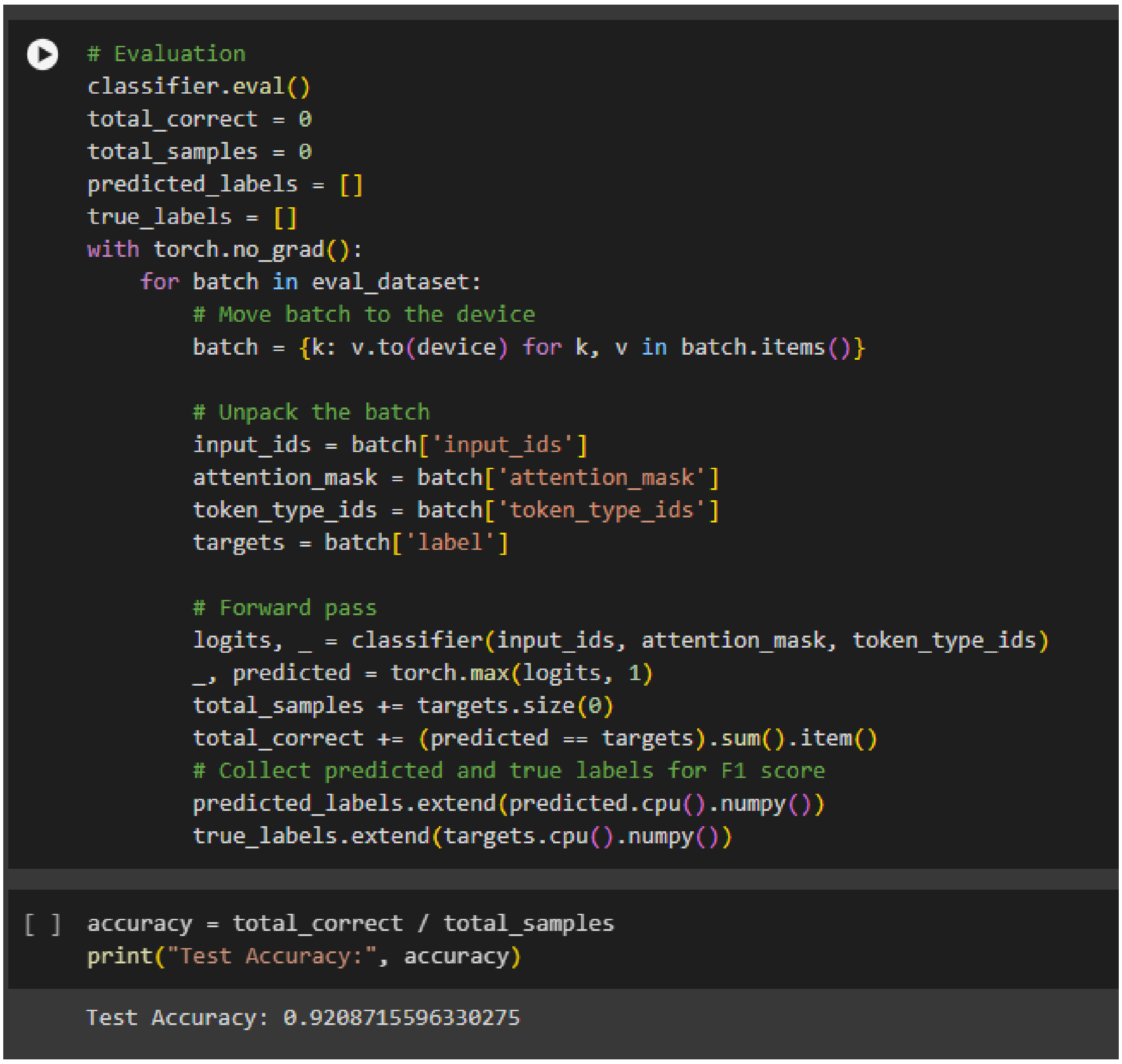


```python
class BERTForClassification(nn.Module):
    def __init__(self, bert_model, num_classes):
        super().__init__()
        self.bert = bert_model
        self.fc = nn.Linear(self.bert.config.hidden_size, num_classes)
        self.attention_layer = nn.Linear(self.bert.config.hidden_size, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, input_ids, attention_mask, token_type_ids):
        outputs = self.bert(input_ids=input_ids, attention_mask=attention_mask, token_type_ids=token_type_ids)
        pooled_output = outputs.pooler_output
        logits = self.fc(pooled_output)

        # Calculate attention weights
        attention_scores = self.attention_layer(outputs.last_hidden_state).squeeze(-1)
        attention_weights = self.sigmoid(attention_scores)

        return logits, attention_weights
```

TRAINING:


```python
# Training loop
num_epochs = 3
for epoch in range(num_epochs):
    classifier.train()
    total_correct_train = 0
    total_samples_train = 0

    for batch in train_dataset:
        batch = {k: v.to(device) for k, v in batch.items()}
        # Move batch to the device
        input_ids = batch["input_ids"]
        attention_mask = batch["attention_mask"]
        targets = batch["label"]

        # Forward pass
        optimizer.zero_grad()
        logits, attentions = classifier(input_ids, attention_mask, None)
        loss = criterion(logits, targets)

        # Backward pass and optimization
        loss.backward()
        optimizer.step()

        # Collect attention weights
        for attention in attentions:
            attention_weight_changes.append(attention.detach().cpu().numpy())

        # Calculate accuracy
        _, predicted = torch.max(logits, 1)
        total_samples_train += targets.size(0)
        total_correct_train += (predicted == targets).sum().item()

    # Calculate accuracy after each epoch
    accuracy_train = total_correct_train / total_samples_train
    print(f'Epoch [{epoch+1}/{num_epochs}], Train Accuracy: {accuracy_train:.4f}')
```

EVALUATION:



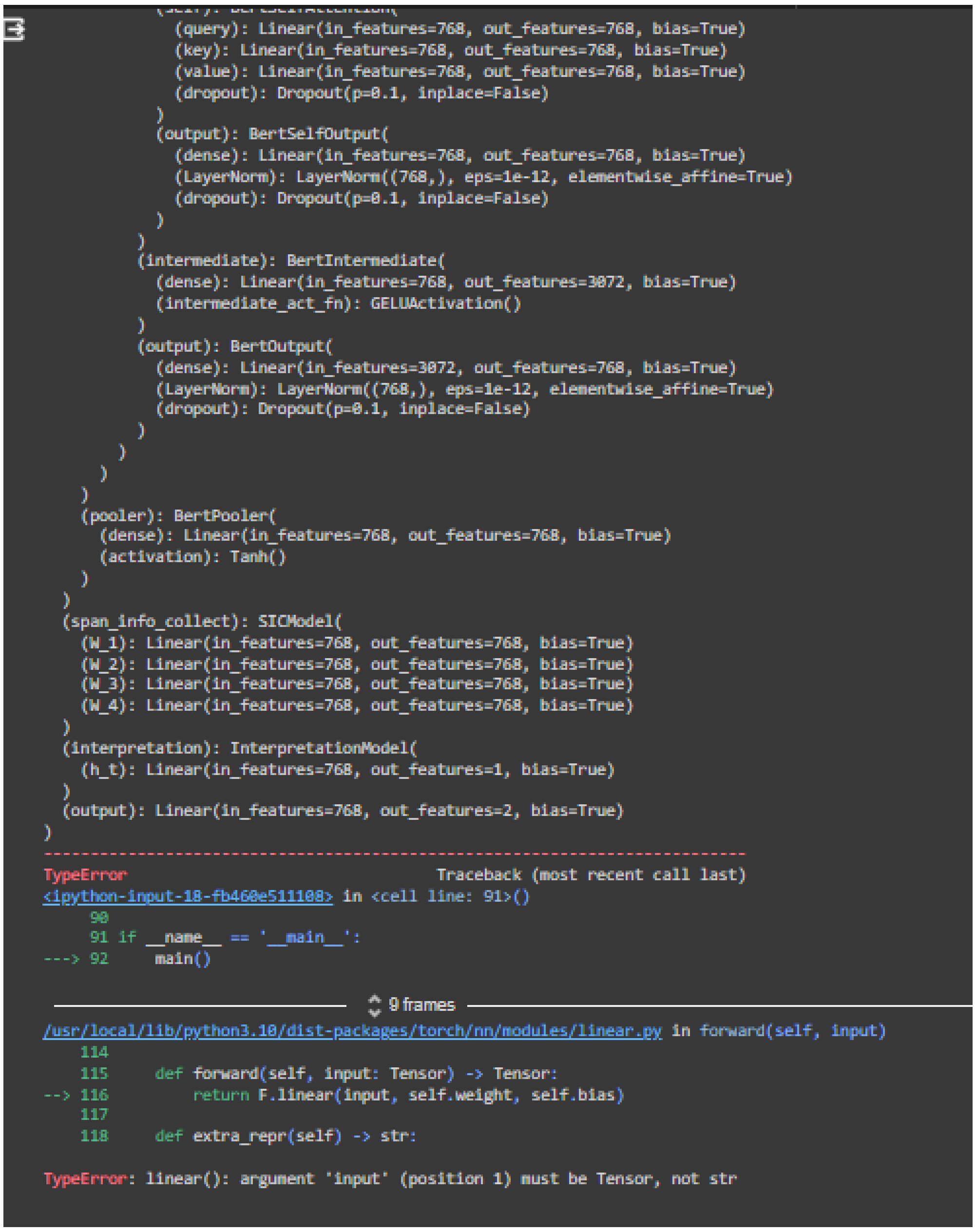
```
# Evaluation
classifier.eval()
total_correct = 0
total_samples = 0
predicted_labels = []
true_labels = []
with torch.no_grad():
    for batch in eval_dataset:
        # Move batch to the device
        batch = {k: v.to(device) for k, v in batch.items()}

        # Unpack the batch
        input_ids = batch['input_ids']
        attention_mask = batch['attention_mask']
        token_type_ids = batch['token_type_ids']
        targets = batch['label']

        # Forward pass
        logits, _ = classifier(input_ids, attention_mask, token_type_ids)
        _, predicted = torch.max(logits, 1)
        total_samples += targets.size(0)
        total_correct += (predicted == targets).sum().item()
        # Collect predicted and true labels for F1 score
        predicted_labels.extend(predicted.cpu().numpy())
        true_labels.extend(targets.cpu().numpy())
```

```
[ ] accuracy = total_correct / total_samples
    print("Test Accuracy:", accuracy)
```

```
Test Accuracy: 0.9208715596330275
```

SELF EXPAINING LAYER :

Adding a pure self explaining layer on the model was not suucessful as there were many errors, this will be considered as the future work of this project.



```
            (query): Linear(in_features=768, out_features=768, bias=True)
            (key): Linear(in_features=768, out_features=768, bias=True)
            (value): Linear(in_features=768, out_features=768, bias=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (output): BertSelfOutput(
            (dense): Linear(in_features=768, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (intermediate): BertIntermediate(
          (dense): Linear(in_features=768, out_features=3072, bias=True)
          (intermediate_act_fn): GELUActivation()
        )
        (output): BertOutput(
          (dense): Linear(in_features=3072, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
    )
  )
  (pooler): BertPooler(
    (dense): Linear(in_features=768, out_features=768, bias=True)
    (activation): Tanh()
  )
 )
 (span_info_collect): SICModel(
  (W_1): Linear(in_features=768, out_features=768, bias=True)
  (W_2): Linear(in_features=768, out_features=768, bias=True)
  (W_3): Linear(in_features=768, out_features=768, bias=True)
  (W_4): Linear(in_features=768, out_features=768, bias=True)
 )
 (interpretation): InterpretationModel(
  (h_t): Linear(in_features=768, out_features=1, bias=True)
 )
 (output): Linear(in_features=768, out_features=2, bias=True)
)
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-18-fb460e511108> in <cell line: 91>()
     90
     91 if __name__ == '__main__':
---> 92     main()

                                9 frames
/usr/local/lib/python3.10/dist-packages/torch/nn/modules/linear.py in forward(self, input)
    114
    115     def forward(self, input: Tensor) -> Tensor:
--> 116         return F.linear(input, self.weight, self.bias)
    117
    118     def extra_repr(self) -> str:

TypeError: linear(): argument 'input' (position 1) must be Tensor, not str
```

CHAPTER 5

CONCLUSION AND FUTURE WORK

The use of Bidirectional Encoder Representations from Transformers (BERT)for sentiment analysis makes advancement in the field of natural language processing (NLP). Leveraging the sophisticated architecture of the Transformer model, BERT excels in capturing deep contextual nuances within text, which enhances the accuracy and reliability of sentiment analysis applications across diverse industries. One of the critical strengths of BERT is its ability to understand the context by processing words in relation to all others in a text, thanks to its bidirectional training. This capability is crucial for accurately capturing sentiments, especially in complex scenarios where sentiments are not explicitly stated but implied through context and subtext. As a result, BERT can effectively handle nuanced linguistic features such as irony, sarcasm, and mixed emotions, which are often challenging for less advanced models. This makes BERT an invaluable tool for businesses and researchers aiming to gain deeper insights into consumer sentiment, market trends, and public opinion, thereby driving informed decision-making and strategy development.

FUTURE WORK: Make this model completely self explainable. The above model doesn't predict the sentiment of the text which consists of sarcasm hence we will create a model which detects the sarcasm first if its not there then this model will apply else another model whose accuracy is good with sarcasmic cases will be applied.

BIBLIOGRAPHY

[1] J. Selvaraj, "Explanation of BERT Model (NLP)," GeeksforGeeks, 2020. [Online]. Available: https://www.geeksforgeeks.org/explanation-of-bert-model-nlp/.

[2] A. Jain, "Introduction to BERT and Its Application in Sentiment Analysis," Medium, 2020. [Online]. Available: https://medium.com/analytics-vidhya/introduction-to-bert-and-its-application-in-sentiment-analysis-9c593e955560.

[3] M. M. Khan, "Why BERT has 3 Embedding Layers and Their Implementation Details," Medium, 2020. [Online]. Available: https://medium.com/@_init_/why-bert-has-3-embedding-layers-and-their-implementation-details-9c261108e28a.

[4] J. Alammar, "The Illustrated BERT, ELMo, and co.," 2018. [Online]. Available: https://jalammar.github.io/illustrated-bert/.

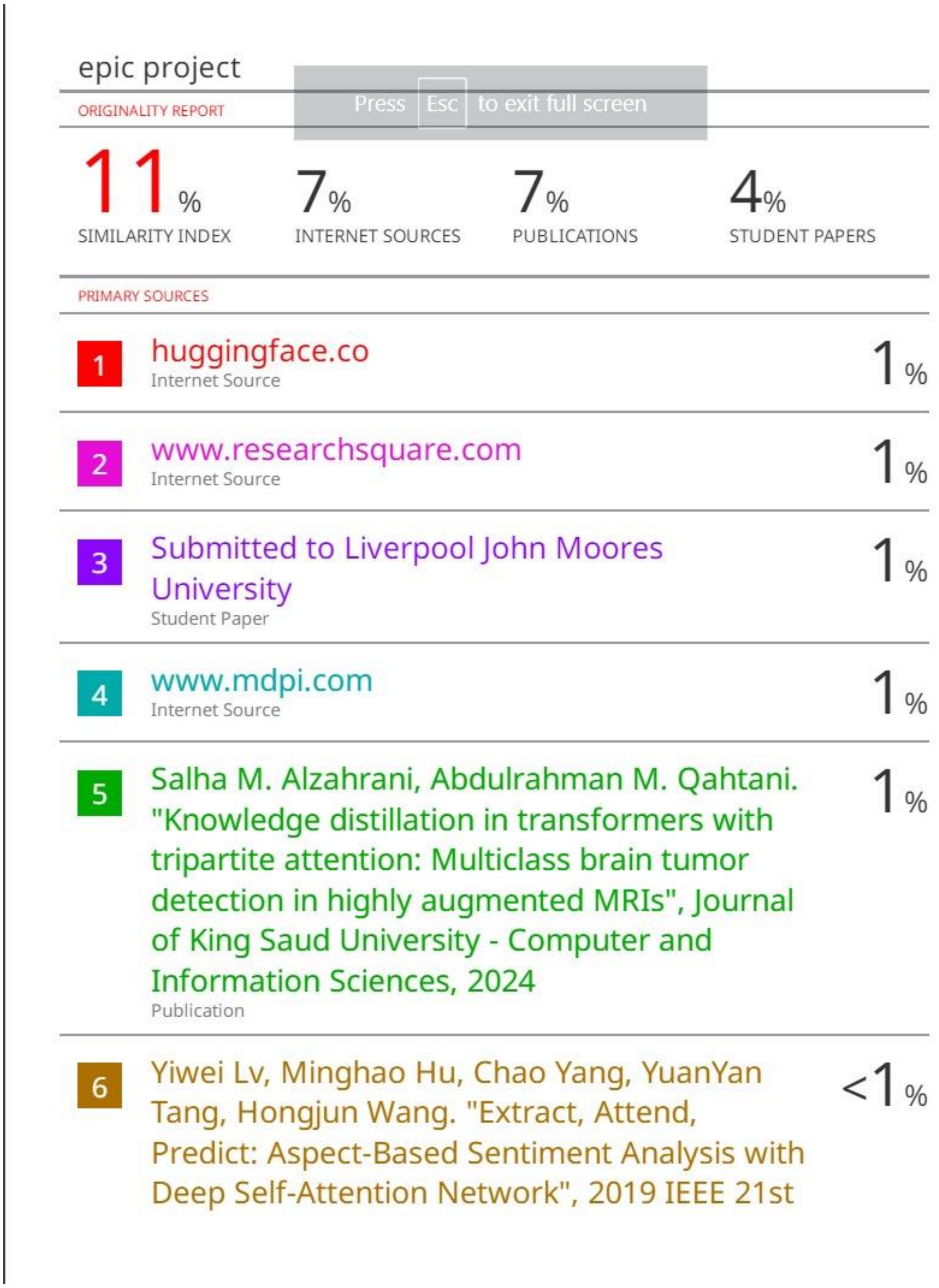
[5] J. Devlin et al., "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," arXiv:1810.04805 [cs], Oct. 2018, Accessed: May 4, 2024.Available: https://arxiv.org/abs/1810.04805.

[6] Hugging Face, "Training & Fine-Tuning," 2024. Available: https://huggingface.co/docs/transformers/training. [Accessed: May 4, 2024].

[7] Hugging Face, "stanfordnlp/sst2" Hugging Face Datasets, 2024. Available: https://huggingface.co/datasets/stanfordnlp/sst2. [Accessed: May 4, 2024].

[8]D. Jurafsky and J. H. Martin, "Speech and Language Processing," 3rd ed. Prentice Hall, 2024. Available: https://web.stanford.edu/~jurafsky/slp3/. [Accessed: May 4, 2024].

TURITIN SCORE



epic project

ORIGINALITY REPORT

Press Esc to exit full screen

11%	7%	7%	4%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	huggingface.co Internet Source	1%
2	www.researchsquare.com Internet Source	1%
3	Submitted to Liverpool John Moores University Student Paper	1%
4	www.mdpi.com Internet Source	1%
5	Salha M. Alzahrani, Abdulrahman M. Qahtani. "Knowledge distillation in transformers with tripartite attention: Multiclass brain tumor detection in highly augmented MRIs", Journal of King Saud University - Computer and Information Sciences, 2024 Publication	1%
6	Yiwei Lv, Minghao Hu, Chao Yang, YuanYan Tang, Hongjun Wang. "Extract, Attend, Predict: Aspect-Based Sentiment Analysis with Deep Self-Attention Network", 2019 IEEE 21st	<1%

	International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2019 Publication	
7	www.ijritcc.org Internet Source	<1 %
8	robots.net Internet Source	<1 %
9	Submitted to University of Derby Student Paper	<1 %
10	Submitted to Bournemouth University Student Paper	<1 %
11	www.nxtbook.com Internet Source	<1 %
12	Babeș-Bolyai University Publication	<1 %
13	export.arxiv.org Internet Source	<1 %
14	savioglobal.com Internet Source	<1 %
15	Dimitris Perdios, Adrien Besson, Marcel Arditi, Jean-Philippe Thiran. "A deep learning	<1 %

	approach to ultrasound image recovery", 2017 IEEE International Ultrasonics Symposium (IUS), 2017 Publication	
16	scholarspace.manoa.hawaii.edu Internet Source	<1 %
17	Submitted to Beijing Institute of Technology, Zhuhai Student Paper	<1 %
18	Gerrit Felsch, Naeim Ghavidelnia, David Schwarz, Viacheslav Slesarenko. "Controlling auxeticity in curved-beam metamaterials via a deep generative model", Computer Methods in Applied Mechanics and Engineering, 2023 Publication	<1 %
19	Maria Sayu Yamamoto, Khadijeh Sadatnejad, Toshihisa Tanaka, Md. Rabiul Islam, Frédéric Dehais, Yuichi Tanaka, Fabien Lotte. "Modeling Complex EEG Data Distribution on the Riemannian Manifold Toward Outlier Detection and Multimodal Classification", IEEE Transactions on Biomedical Engineering, 2024 Publication	<1 %
20	peerj.com Internet Source	<1 %
21	downloads.hindawi.com Internet Source	<1 %