MULTI-CLASS TEXT CLASSIFICATION USING PRE-TRAINED BERT MODEL

by

Haider Zaman Khan

In the partial fulfillment of the requirements

for the degree
Master of Philosophy

Department of Electronics
Quaid-I-Azam University, Islamabad
2021-2023

Certificate

It is certified that the work presented in this dissertation is accomplished by Haider Zaman Khan under my supervision at Quaid-I-Azam University, Islamabad, Pakistan.

Supervisor:

_____

Dr. Mussrat Abbas
Associate Professor
Department of Electronics
Quaid-i-Azam University, Islamabad

Submitted through:

_____

Prof. Dr. Qaisar Abbas Naqvi
Chairman
Department of Electronics
Quaid-I-Azam University, Islamabad

Acknowledgements

Praise to ALLAH, the Cherisher and Sustainers of the world, the most Beneficent and Merciful. The Lord of Universe, Creator, Master and Sovereign, created man and bestowed upon him the faculties of learning speaking, understanding and granted him a kind of autonomy and appointed him as His vicegerent on the Earth. All respects are for His Holy Prophet Hazrat Muhammad (peace be upon him) who guided the world towards straight path so that humanity plunged out from ignorance and enlightened the human life with knowledge.

I wish to express my heartfelt gratitude and sincere appreciation to my supervisor and teacher Dr. Mussrat Abbas for his valuable advice, continued interest, enthusiasm and great efforts. His supervision encouraged me to improve my research capabilities with keen interest and passion. The thesis in this present form is only due to his full cooperation and support. I am grateful to Prof. Dr. Qaisar Abbas Naqvi, Chairman of the Department for providing me all the necessary facilities to complete my thesis work.

I am specially obliged to my senior Ms. Umamah Bint Khalid for her caring and helpful attitude. I could not express enough appreciation to my family, words could not do justice to what their support, appreciation and well wishes mean to me. My father and my mother who stood by me in every situation, their love and prayers took me to the place where I am now. My sibling and friends, who helped me through my tough time.

Haider Zaman Khan

Dedicated To my beloved Family and, Teachers

Abstract

Natural Language Processing (NLP) has gained immense popularity due to its fundamental application in automatically categorizing text documents into predefined groups or classes, enabling the extraction of valuable insights from unstructured text data. This research aims to create a robust and efficient text classification model using state-of-the-art techniques while minimizing time and resource requirements. Leveraging the Transformer framework, specifically the BERT model for NLP tasks, this work focuses on augmenting pre-trained BERT models with cross-validation techniques for text classification. The model is fine-tuned using the 20 Newsgroups dataset, comprising both preprocessed and raw datasets with 20 distinct classes each. The achieved accuracy rates of 92% for the preprocessed dataset and 90% for the raw dataset highlight the superiority of pre-trained BERT models in real-world text classification challenges. This research contributes by exploring the synergy of BERT models with cross-validation and fine-tuning strategies, aiming to outperform traditional baseline models in multiclass text classification tasks. It also includes 5-fold cross-validation for efficient performance of text classification, mitigating overfitting and class imbalance issues, resulting in better accuracy with low computational resources.

Contents

List of Figures

List of Tables

List of Acronyms

Acronym	**Description**
AI	Artificial Intelligence
ANN	Artificial Neural Network
ML	Machine learning
GAN	Generated Adversarial Networks
CNN	Convolutional Neutral Network
LSTM	Long Short-Term Memory
RNN	Recurrent Neutral Network
GRU	Gated Recurrent Unit
SVM	Support Vector Machine
KNN	K-Nearest Neighbor
NER	Named Entity Recognition
DL	Deep Learning
MLM	Masked Language Modeling
NSP	Next Sentences Prediction
NLP	Natural Language Processing
RAdam	Rectified Adam
GPT	Generated Pre-trained Transformer
BERT	Bidirectional Encoder Representations from Transformers

Chapter 1

Introduction

1.1 Artificial Intelligence

The term AI stands for Artificial Intelligence, which aims to build machines intelligent enough to think, learn, and act automatically. Machine learning and Deep Learning are two prominent branches of AI that focuses on algorithms which can learn from data without explicit programming [24]. Large datasets of pre-labelled samples train such algorithms. The algorithms are then used for predictions on unseen data. Due to the availability of enormous datasets and the rising processing capacity of computers, AI has grown in popularity over the past few years. Applications include algorithms that are used in computer vision to recognize objects in pictures and movies. Other examples include self-driving cars, facial recognition, and image analysis in medicine, all utilize this technology. AI algorithms are also used in natural language processing, i.e., to comprehend and analyze spoken as well as written words. Applications using this technique include question answering, spam screening, machine translation, text classification and so on. In order to recognize human speech, AI algorithms are used in speech recognition. Applications for this technology include voice assistants, dictation software and contact centers. New AI applications are constantly being created since the field is continually growing. As data availability and computer power both continue to increase, AI is likely to become considerably more important in the years ahead.

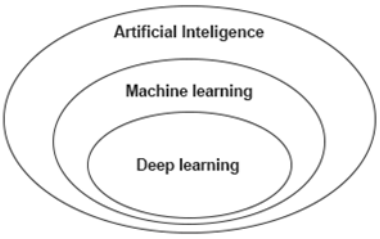

Figure 1.1: AI, ML and DL

With a focus on its uses, developments and difficulties, this thesis seeks to investigate the function of artificial intelligence in machine learning. It will go into the fundamental ideas and methods of machine learning, such as reinforcement, unsupervised and supervised learning. Additionally, it will highlight successful use cases and real-world examples of machine learning effects across diverse industries. The ethical implications of machine learning will also be rigorously examined in this thesis, with a focus on the necessity of justice, transparency and responsibility in its use.

1.2 Machine learning and Deep learning

1.2.1 Machine learning

A sub-branch of AI that focuses on creating algorithms or models that work like machines and make predictions or decisions without explicit programming for each task [4] [11]. It is a field dedicated to develop systems that can learn and improve through experience. Fundamentally, machine learning revolves around training the models using data. Instead of providing explicit instructions, machine learning algorithms learn patterns and linkages within the data to make predictions or initiate actions. These algorithms are designed to autonomously identify and extract significant features or patterns from the input data, enabling them to generalize same and make accurate predictions on new unseen data [4].
The following steps are commonly included in the machine learning process:

- **Data collection:** Relevant data is collected or generated to serve as input for the learning process.
- **Data preprocessing:** The collected data is cleaned, transformed, and prepared for model training. This step includes tasks like removing outliers, tackle missing values, and normalizing the data to a standard.
- **Model training:** The prepared data is used to train the learning model. During this phase, the model is presented with input data along with the desired output or target variable. The model adjusts its internal parameters based on the provided examples, optimizing its ability of accurate predictions on unseen data.
- **Model evaluation:** The trained model is examined against test set or validation data separated beforehand and not used during the training. This step evaluates performance of the model and its ability on unseen data. Various parameters like accuracy, precision, recall and F1 score are used to measure the performance.
- **Model deployment:** Once a satisfactory level of performance is achieved, the trained model can be deployed for real predictions or take actions on new, real-world data. This may involve fixing the model into a larger software system or creating an independent application or service that utilizes the model's predictions.

1.2.2 Deep Learning

Artificial neural network architectures are utilized in deep learning to effectively address complex real-world problems. Models are made to mimic human brain like arrangement for information processing and data analysis. They perform well in various kinds of positions and have the capacity to take in learning from mistakes, enabling them to continuously develop and adapt. The neurons that make up the layers of connected nodes that make up deep learning algorithms, analyzes data and render conclusions based on the inputs they receive. The output layer generates the final prediction or classification, whereas the input layer receives the raw data in the form of photos or text. The ability of the model to predict outcomes accurately is aided by the intermediate hidden layers that form body of the neural network and carry out different calculations and features extraction. In contrast to typical machine learning algorithms where features extraction is needed to be done manually, deep learning models are capable of autonomously learning and extracting characteristics from the input [11]. Deep learning models may now successfully complete tasks like picture identification, natural language processing, and audio recognition etc. where the input data is complicated and multidimensional. In different applications like image and speech recognition, driver-less cars and drug development, deep learning has shown outstanding results.

1.3 Natural Language Processing (NLP)

NLP is mainly concerned with as how computers can be made to understand human languages. Using computer methods, NLP processes and analyzes unstructured natural language data, including text, audio, and images with text. Its fundamental objective is to develop models that enable the extraction of useful information from human language for a variety of activities [1] [2]. Natural language processing has many difficulties due to its complexity and ambiguity, which necessitates precise context interpretation and, word and phrase meaning disambiguation by the models. Transformers and recurrent neural networks have become effective NLP tools in recent years [26] [14]. It has attained stunning performance in many applications and can learn intricate patterns in natural language data. The inability to fully comprehend deep learning models and the potential for bias in NLP models as a result of a lack of diversity in training data are problems that still need to be fixed. The usage of NLP could raise ethical issues like copyright. Particularly when NLP models are used to make decisions that have an impact on people's lives, like hiring or loan decisions, this might result in erroneous or discriminating results. NLP is a fast-developing field with a wider scope of potential applications and chances for creativity. The following are some of the most typical NLP related areas of interest.

- **Named Entity Recognition:** This entails locating and extracting particular entities from a text source, like names, places, and organizations. It can be applied to tasks like knowledge graph generation and information extraction.
- **Machine Translation:** A text translation from one language to another is done automatically in this. Due to the complexity of human language, machine translation can be problematic, although recent developments in NLP have improved its accuracy and efficiency.

- **Sentiment Analysis:** This involves predicting emotional standing of a given piece of text, such as contained in tweets or product reviews. Sentiment analysis can be used for tasks such as brand monitoring and customer feedback analysis [5].
- **Text Generation:** This involves creating new text in reference to a given input, such as a prompt or a set of keywords. Text generation can be used for tasks such as content creation and creative writing.
- **Speech Recognition:** This can be used for tasks such as virtual assistants and dictation software.
- **Question Answering:** Building systems that can understand and respond to questions posed by users, often requiring the comprehension and extraction of relevant information from text.
- **Text Classification:** This involves assigning a name or category to a particular text. Text classification is a technique used for a variety of tasks like spam detection, sentiment analysis, and topic classification.
- **Text Summarization:** Generating concise summaries or abstracts of longer text documents, capturing the key information and main points.
- **Information Extraction:** Automatically extracting structured information from unstructured text, such as extracting relationships between entities or extracting key facts from news articles.
- **Topic Modeling:** Identifying and extracting underlying themes or topics from a collection of documents, enabling organization and exploration of large text corpora.
- **Text Mining:** Applying data mining techniques to extract valuable insights, patterns, or trends from large volumes of text data.

These are only a few of the numerous NLP task kinds. The methodologies and algorithms needed for each task vary, and as new models and approaches are created, the discipline of NLP continues to evolve.

1.4 Text Classification

The fundamental job of text classification in NLP is automatically classifying and arranging massive amounts of textual input into predetermined categories or classes. Due to the increased accessibility of vast volumes of text-based data in recent years, including news articles, social media postings, emails, and product evaluations, among others, this task has grown more crucial [14]. An algorithm is trained on a labelled dataset made up of examples and their pre-tagged classes for text classification or categorization. The system uses this training data to generalize and categorize new, unseen text into one of the pre-defined groups. Finding pertinent details or patterns in the text that distinguish between the various groups is the major issue in text

categorization. Text classification may be used in e-commerce to classify user reviews, anticipate client preferences, and automatically categorize items [14]. It may be applied to finance to categorize financial news, gauge market sentiment, and spot fraudulent activity. It may be applied to the healthcare industry to organize patient information, identify medical issues, and forecast patient outcomes. Naive Bayes, Support Vector Machines, Decision Trees, Random Forests, and Neural Networks are the few well-known machine learning techniques for text categorization [14] [18]. CNN and RNN have recently exhibited outstanding results in text categorization tests [1]. These models can develop meaningful representations of the text data and are capable of capturing complicated linkages and dependencies in the text [30]. Overall, text categorization is a difficult and significant NLP job with several practical applications. Text categorization models are likely to become even more relevant for a variety of sectors and applications as machine learning and deep learning techniques advance, increasing their accuracy and scalability. Based on the nature of the classification and the amount of granularity of the categories, text classification task can be divided into four different scope levels. The following are the four levels of scope:

- **Document-level classification:** At this level of text categorization, whole documents or individual texts are categorized into predetermined groups or classes. A piece of news, for instance, may be divided into subcategories like politics, sports, entertainment, or business. Applications like content filtering, document management, and news aggregation frequently employ document-level categorization.

- **Sentence-level classification:** This level of text classification involves classifying individual sentences within a document into categories. A customer evaluation of a product, for example, may be divided at the phrase level into categories like product quality, customer service, and delivery time. Sentence-level categorization is frequently utilized in applications like sentiment analysis, opinion mining, and chatbots.

- **Entity-level classification:** This level of text classification involves identifying and categorizing people, organizations, and locations within the documents. For example, news articles could be classified at the entity level into categories such as politicians, sports teams, or cities. Entity-level classification is often used in applications such as information extraction, knowledge management, and search engines.

- **Aspect-level classification:** This level of text classification involves identifying and categorizing specific aspects or attributes of a product, service, or topic within a document. For example, a customer review of a hotel could be classified at the aspect level into categories such as room cleanliness, staff friendliness, or location convenience. Aspect-level classification is often used in applications such as opinion mining, customer feedback analysis, and product recommendation systems.

Each level of scope has its own unique challenges and requirements, and the appropriate level of classification depends on the specific task and application. Some text classification systems may combine multiple levels of classification to yield a more comprehensive analysis of the text data.

1.4.1 Text Classification Steps

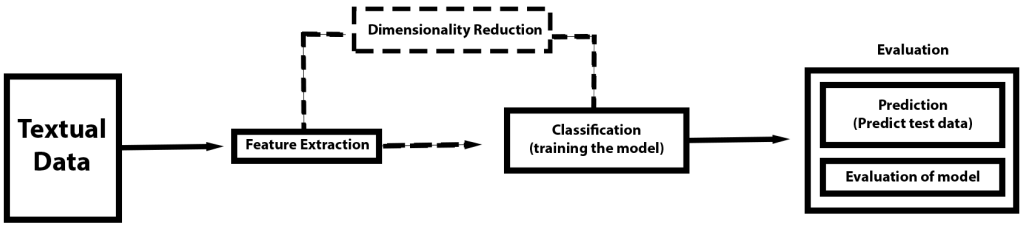

Figure 1.2: Text Classification Pipeline

A text classification pipeline is a series of steps that takes in raw text data and outputs the predicted class labels for that data. The pipeline typically includes the following steps:

- **Data preprocessing:** This step cleans and transforms the raw text data into a format suitable for analysis. Removing stop words, stemming or lemmatizing words and punctuations, and converting the text to a lowercase are all part of data preprocessing.
- **Feature extraction:** This step involves transforming the text data into vectors or numerical features that can be comprehended and processed by a machine learning algorithm. Such techniques include bag-of-words, term frequency-inverse document frequency (TF-IDF), word embeddings, and character n-grams etc [12].
- **Dimensionality reduction:** In order to increase effectiveness and performance of classification, this stage entails reducing dimensionality of the feature space. Because there are so many distinctive words or properties in the corpus, text data can have a very high degree of dimension. Principal Component Analysis (PCA), Singular Value Decomposition (SVD), and t-distributed Stochastic Neighbor Embedding (t-SNE) are few examples of dimensionality reduction methods that can be used to decrease the number of dimensions in the feature space while retaining the maximum of the original variance. This may result in shorter training times, higher generalization and easier model interpretation [1]. The right dimensionality reduction technique and the number of dimensions to keep need to be carefully chosen, though, as too much dimensionality reduction might cause data loss and poor performance.
- **Model training:** This step involves selecting an appropriate machine learning algorithm and training it on a labeled dataset of text data. This involves segregating the data into training and testing sets, selecting best-suited hyperparameters for the algorithm, and tuning the model to give optimum performance on the validation set.

- **Model evaluation:** This involves performance appraisal of the trained model on a test set of unlabeled text data. This may include metrics such as accuracy, precision, recall, and F1 score.

- **Model deployment:** Once the model has been optimally trained and evaluated, it can be deployed in a real-world production environment to classify new, unseen and unlabeled text data.

Each step in the text classification pipeline plays an important role in the overall performance of the system. The choice of preprocessing techniques, feature extraction methods, and machine learning algorithms will depend on the specific task and the characteristics of the text data. A well-designed and optimized text classification pipeline can provide valuable insights and automate decision-making processes in a variety of industries and domains.

1.4.2 Text classification Techniques

Text classification can be done either manually or automatically. Manual text classification requires a human annotator to analyze contents and assign the appropriate category, which is cumbersome, costly and error-prone. Automated text categorization uses machine learning approaches to sort out and categorize texts promptly, efficiently, and accurately.

- **Manual text classification:** It involves manually assigning categories or labels to a given set of text data. This approach can be time-consuming and resource-intensive, but it can also provide a high degree of accuracy and granularity in the classification process [13]. Manual text classification is often used when the categories or labels are highly specific and cannot be easily automated. For example, in legal or medical contexts, where specific terminology is used, manual classification may be necessary to ensure accuracy. To perform manual text classification, a team of human annotators is typically employed. The annotators are given a set of guidelines or rules to follow when assigning categories or labels to the text data. They may also be provided with examples of previously classified text data to help ensure consistency and accuracy. While manual text classification can be highly accurate, it can also be subject to human biases and inconsistencies. It is also not scalable for large datasets or for tasks where the categories or labels are not well-defined. For these reasons, automated text classification using machine learning algorithms is often preferred for larger datasets or when the categories or labels can be defined in advance.

- **Automatic text classification:** It involves using machine learning algorithms to auto assign categories or labels to text data. Automated text categorization is excellent for huge datasets and applications that need real-time classification since it may be very accurate and scalable [13]. Based on the quality and sufficiency of the training data and the selected machine learning technique, it may also be biased and prone to overfitting and other errors. To make sure that the model keeps functioning as intended, it is important to thoroughly examine and constantly monitor its performance throughout time. There are several applications for automatic text classification, including document classification, sentiment analysis and information retrieval, and it is generally a useful technique for

studying big amounts of text data [14]. We may anticipate seeing increasingly more advanced and precise text categorization algorithms in the future as natural language processing methods develop, fine-tune and advance further.

1.4.3 Text classification Approaches

There are different approaches to automatic text classification, which generally fall under three types of systems:

i. Rule-based systems
ii. Machine learning-based systems
iii. Hybrid systems

Rule-based systems

Rule-based systems, a form of machine learning strategy used in text categorization, employ a preset set of rules to make judgements or execute actions. These guidelines are frequently based on specific criteria, including the text's inclusion of particular words or phrases. The algorithm may assign the text to a specific category, for instance, if it contains particular words or phrases. These rules can be manually defined or generated using natural language processing techniques [28]. Since the rules are openly established, rule-based systems have the benefit of being very transparent and simple to comprehend. They might not be appropriate for managing more ambiguous or complex material, though, as the rules are set in stone and cannot be modified to fit into different settings or scenarios. A rule-based system is merely one technique to categorize text, and how important it is will depend on the issue at hand. To increase classification accuracy, other strategies, such machine learning algorithms, can be employed to automatically discover and retrieve patterns and features in the data. There are numerous instances of rule-based systems being employed in different applications. They are given below:

- **Spam filters:** Spam filters look for and eliminate unwanted emails using a set of established rules. These guidelines may include things like searching for specific keywords, looking at email headers, or looking for sources of recognized spam.

- **Chatbots:** Many chatbots produce responses to user inputs using rule-based systems. What's the weather like, say a user types? as an example. To recognize that the user is asking about the weather and produce the proper response, the chatbot may employ a set of rules [20].

- **Fraud detection:** Several fraud detection systems identify problematic transactions using rule-based approaches. A rule that flags a transaction for additional inquiry might be triggered, for instance, if it exceeds a specific dollar threshold, involves a novel or unusual location, or occurs outside of regular business hours.

- **Sentiment analysis:** A text might be categorized as having positive sentiment if a set of rules finds that the document contains more positive terms than negative ones [14].

Machine learning based systems

Machine learning-based systems employ statistical models and algorithms to automatically learn to classify text based on sample data. These systems can be trained on large datasets and adapt to new situations by updating their models based on new data. The most common machine learning algorithms used in text classification are decision trees [14], support vector machines, naive Bayes, logistic regression and neural network-based architectures. These algorithms can be further classified into supervised, unsupervised and semi-supervised learning methods. In text classification, the training data consists of labeled documents, where each document is assigned a category or label. The machine learning algorithm learns to recognize patterns in the text that are indicative of each category, and uses this knowledge to classify new unlabeled documents.

Hybrid systems

A hybrid system in machine learning is one that combines various machine learning algorithms or strategies to boost a system's overall performance. Text classification is one of the many uses for hybrid systems. Combining different classifiers to form an ensemble model is a typical strategy for developing a hybrid text classification system. Ensemble models generate predictions using a number of classifiers, which are then integrated to get a final prediction [31]. This can be done using a variety of strategies, including weighted voting, stacking, and boosting. Another approach for developing a hybrid text classification system involves combining different machine learning methods, such as a rule-based system with a deep learning model. When handling complicated jobs that call on a variety of knowledge or expertise, this strategy can be especially beneficial. The handling of unbalanced datasets is one specific difficulty in text categorization that hybrid systems can help with. For instance, a hybrid system might combine oversampling methods with an SVM classifier to enhance classification performance on unbalanced datasets [18].

1.5 Challenges and Limitation

Text classification has certain limitations that can impact its effectiveness. These limitations include text-ambiguation as the most challenging for correct interpretation, where same words or phrases reflect different meanings in different surrounding contexts. Text classification models trained on specific domains may not perform well on different domains. The language, vocabulary, and writing style can vary significantly across domains, making it challenging for models to accurately classify text outside their training domain. Text classification models heavily rely on training data to learn patterns and make accurate predictions. If the training set is limited or biased, the model's performance may suffer. Biases present in the data, such as underrepresentation of certain groups, can be perpetuated by the model. Text classification models may encounter words or phrases that were not present in their training data. These out of vocabulary (OOV) words sometimes pose challenges as the model may struggle to understand or classify these accurately. Some text classification approaches rely solely on matching keywords or simple rules. This approach may overlook the broader context and meaning of the text, leading to mis-

classification. Imbalanced datasets occur when certain classes have significantly more or fewer instances compared to others in the training data [31]. Models trained on imbalanced data may struggle to accurately classify minority classes. Deep learning models used for text classification are often regarded as black-box models. While these provide accurate predictions, they lack interpretability, making it difficult to understand how they arrived at a specific classification decision. Language constantly evolves, with new words, phrases, and slang emerging over time. Text classification models trained on outdated data may struggle but hard to understand and classify contemporary language accurately.

1.6 Motivation

Addressing the text classification problem, several steps are used to develop an effective solution. First, the objective of the text classification task and the categories or labels to be assigned to the text data need to be clearly defined. Then, a diverse and representative dataset for training and evaluation can be gathered. Preprocessing the text data includes removing noise, standardizing formats, and handling missing values. Feature extraction methods are then used to extract distinct features from the text data that can capture the representative characteristics of each category. This can involve techniques such as bag-of-words and word embeddings. Next, an appropriate text classification algorithm or model matching the specific requirements and characteristics of the subjected problem is chosen. After that, the dataset is divided into training and testing sets. The chosen model is trained on the training set, and the parameters are fine-tuned for optimized performance. The model is evaluated on the validation data using appropriate metrics like accuracy, precision, recall, or F1-score. The model can also be fine-tuned by experimenting with different hyperparameters, feature representations, or regularization techniques to improve classification performance. Once satisfied, the model is evaluated on a separate testing set to examine its generalization capability. The trained model is deployed in a production environment to classify new, unseen text data. Continuously monitoring the performance of the deployed model, gathering feedback, and addressing any issues or drifts in classification accuracy is essential. Regularly retraining or updating the model with new data ensures its relevance and effectiveness over time. By following these steps, a robust text classification solution can be built that effectively categorizes text data, enabling applications and insights in various fields like information retrieval, sentiment analysis, recommendation systems, and many more.

1.7 Thesis orgranization

Chapter 1 serves as the introduction to the thesis, providing an overview of the research topic, including Artificial Intelligence, Machine Learning, NLP and Text Classification. **Chapter 2** includes the literature review related to techniques to overcome performance issues in text classification. **Chapter 3** includes a comprehensive detail of Deep learning-based transformer architecture models and its popular variant, the BERT model. **Chapter 4** introduces a methodology that involves utilizing a pre-trained BERT model for conducting text classification on the

20newsgroup dataset. This methodology aims to evaluate the performance of BERT model in handling the classification task effectively. The results and discussions from the experiments carried out during the investigation are presented in **Chapter 5**. This chapter examines the results and evaluates them. **Chapter 6** provides closing thoughts and suggests directions for more study. It highlights the study's main conclusions, talks about its contributions to the field, and makes recommendations for possible directions for future research and development in the text categorization and natural language processing domains.

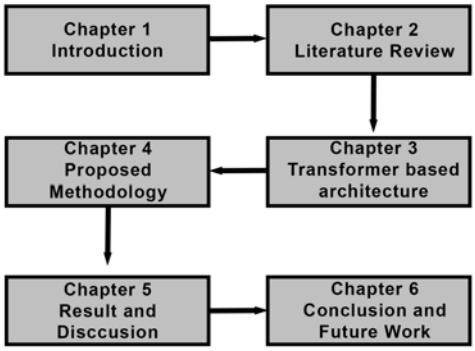

Figure 1.3: Thesis Structure

1.8 Chapter Summary

This chapter provides an overview of ML and its types, Natural language processing, Text Classification, and the important use of Text Classification in the world. It covers the advantages and disadvantages of text classification and use of text classification-based algorithm. Furthermore, the chapter shows the research's motivation objectives and the thesis organization.

Chapter 2

Literature review

Over the years, researchers have developed different approaches to address text classification, ranging from traditional machine learning algorithms to more recent deep learning techniques [14]. In this chapter, we examine the existing literature on text classification and explore some of the famous learning techniques that have shown remarkable success in this field [19].

2.1 Recurrent Neural Networks

RNN are networks commonly used in natural language processing and text classification. These are capable of processing sequential data while maintaining internal state or memory of previous inputs. This property makes these suitable for tasks like language modeling, speech recognition and machine translation. The recurrent unit, which accepts both the current input and the network's past state as inputs, is the fundamental building block of RNN. The output of the recurrent unit is transmitted together with a modified version of the prior state to the following unit in the sequence. As a result, the network may keep a running internal memory of prior inputs and utilize that data to forecast what the current input will be.

RNNs can combine with attention mechanisms and transfer learning to further enhance perfectness of text classification tasks. RNNs can however be computationally expensive to train and can suffer from issues such as vanishing and exploding gradients, which can make training difficult [16]. Nonetheless, RNNs remain a powerful and widely used technique for text classification and other natural language processing tasks. RNNs' ability to interpret sequences of varying length is one of its key advantages for natural language processing tasks. This is achieved by processing the input sequence through rotating one character at a time, keeping internal memory of previous characters. This lets the network to learn patterns and mutual dependencies within the sequence.

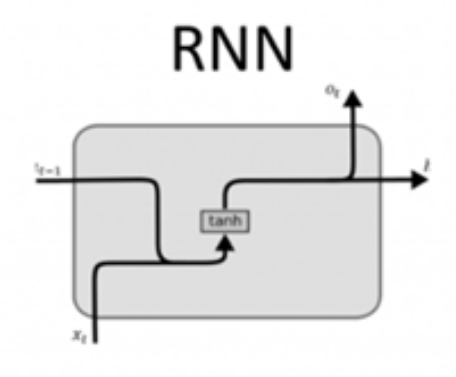

Figure 2.1: Recurrent Neutral Network

2.2 Long Short-Term Memory

LSTM stand out as a crucial advancement in the realm of RNN, particularly for tackling a persistent issue known as the vanishing gradient problem. This problem arises when RNN struggle to retain information over long sequences, hindering their ability to effectively capture dependencies in sequential data. LSTMs address this challenge by introducing specialized memory cells and arrangements within the network architecture. These components endow LSTMs with the remarkable ability to selectively retain or forget information as needed, even across lengthy input sequences. This capability makes LSTMs highly effective in capturing long-term dependencies as well as mitigating the challenges associated with modeling sequential data.

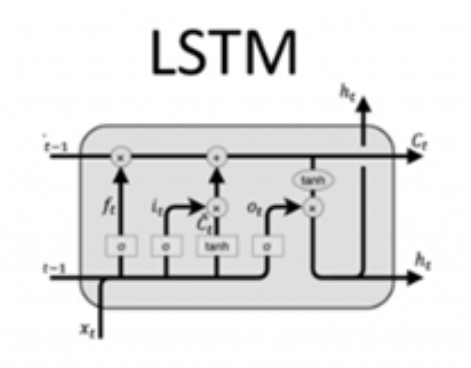

Figure 2.2: Long Short-Term Memory

LSTMs have been widely applied in text classification tasks and have demonstrated superior performance. For example, Zhang et al. (2015) utilized LSTM networks for text categorization, achieving significant improvements over traditional methods. The ability of LSTMs to capture and retain relevant information over extended contexts has made them a popular choice in

many NLP applications. Furthermore, researchers have explored variations of LSTM, such as bidirectional LSTMs, which process sequential data both in forward and backward directions, capturing dependencies from past and future contexts [4].

2.3 Gated Recurrent Unit

GRUs are another variant of RNNs that address limitations of traditional RNNs while offering a simpler architecture compared to LSTMs. GRUs employ gating mechanisms similar to LSTMs, enabling them to selectively update and reset their hidden states. This gating mechanism helps GRUs to effectively model dependencies in sequential data while maintaining computational efficiency.

Research studies have demonstrated the effectiveness of GRUs in text classification tasks. For instance, Chung et al. (2014) compared the performance of GRUs with LSTMs on various sequential learning tasks, including text classification, and found that GRUs achieve competitive results with fewer parameters. GRUs offer a balance between mustering of long-term dependencies and reducing computational efforts, making these a popular choice especially in text classification applications [1].

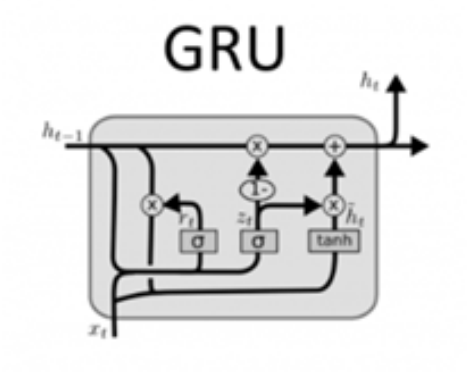

Figure 2.3: Gated Recurrent Unit

2.4 Convolutional Neural Network

CNNs are deep neural networks that excel at capturing spatial and local patterns in data. They utilize convolutional, pooling and fully connected layers to process and extract features from input data. CNNs are particularly effective in figuring out meaningful patterns such as edges, shapes, and textures in image analysis tasks [30] [28]. However, researchers have extended the application of CNNs to text classification tasks by treating text as a one-dimensional sequence of words or characters. CNNs for text classification operate on fixed-length input, such as

sentences or documents. The key idea is to apply convolutional filters over the input, capturing local patterns. These filters slide across the input and conduct element-wise multiplications and additions to generate feature maps. Pooling layers then act to optimize or reduce dimensionality of the features space, preserving only the salient ones. Finally, fully connected layers and softmax activation function make predictions based on the extracted features [28].

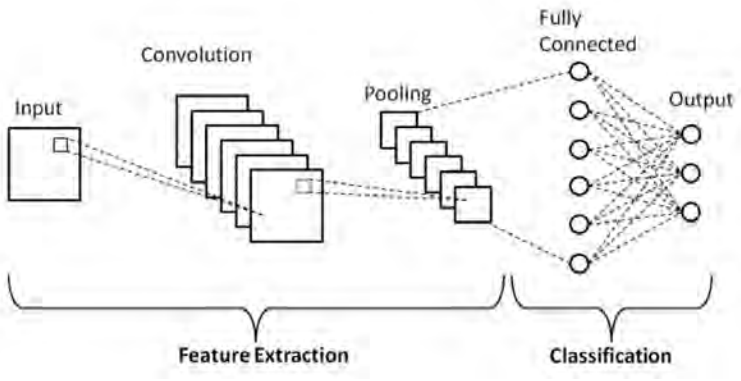

Figure 2.4: Convolutional Neutral Network

CNNs for text classification have demonstrated impressive performance, often rivaling or even surpassing other approaches. Moreover, researchers have explored different variations and enhancements to CNNs for text classification. For instance, Zhang et al. (2015) proposed a character-level CNN that operates on characters of the text, enabling the model to capture fine-grained linguistic information [31]. This approach proved effective for tasks such as sentiment analysis and text categorization.

2.5 Generated Adversarial Networks

This type of deep learning models termed as GAN have completely changed the way that generative modeling is done. Introduced by Ian Goodfellow and his colleagues in 2014 [22], GANs have earned significant attention due to their ability to synthesize data that nearly resembles the training data distribution. GANs are primarily composed of a generator and a discriminator. The generator produces synthetic data that closely mimic the training data. It takes as input a random noise vector and transforms same into a generated sample. The generator typically consists of fully connected or convolutional layers, and followed by activation functions that induce non-linearity. The generator generates samples which are identical to the original data [22]. The discriminator, on the other hand, functions as a binary classifier and seeks to distinguish between genuine and generated data. It takes real data and output from the

generator as inputs and learns to distinguish between the two. The discriminator is typically designed as a binary classifier with layers that retrieve features from the input and apply a sigmoid activation function to output a probability that indicate the likelihood of the input being real.

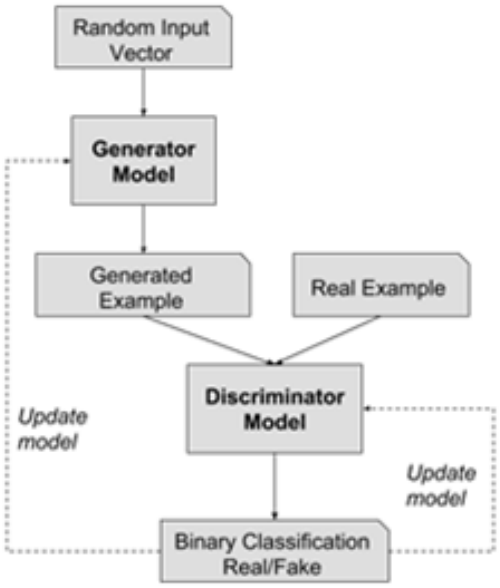

Figure 2.5: Generated Adversarial Networks

During training, GANs involve a game between the generator and discriminator. In order to trick the discriminator, the generator attempts to produce realistic samples, and the discriminator gains the ability to differentiate between created and actual samples. By reducing a loss function that represents their rivalry, this adversarial process drives learning [22]. Whereas the discriminator's loss depends on correctly classifying samples, the generator's loss is dependent upon deceiving it. GANs have demonstrated success in a number of domains, including text and image generation, yielding content that is contextually relevant and high-quality images [22].

Despite their remarkable achievements, GANs also come with certain challenges. One significant challenge is mode collapse, where the generator produces a limited variety of samples or fails to capture entire data distribution. Another challenge is training instability, where the generator and discriminator can stuck in a suboptimal equilibrium during training. Additionally, GANs require careful tuning of hyperparameters and substantial computational resources for training.

2.6 Autoencoders

Unsupervised neural network models called autoencoders are employed for tasks including feature extraction, data compression, and dimensionality reduction. The input data is first cod-

ified into a low-dimension latent space and then it is decoded to recover the original form. Autoencoders consists of three layers which are encoder, decoder and bottleneck layer. The encoder network compresses input data by reducing its dimensionality and capturing only essential features. The bottleneck layer acts as a bottleneck for information flow and represents the compressed latent space. The decoder network aims to reconstruct the original input by progressively increasing the dimensionality [23]. During training, the autoencoders reduce the difference or mismatch, between the original input and the reconstructed output, using a loss function like mean squared error approach. The network parameters are adjusted through optimization algorithms like gradient descent. Autoencoders offer advantages such as they can detect anomalies by reconstructing samples and identifying reconstruction errors. Autoencoders are also useful for data denoising through training on noisy data and generating clean samples.

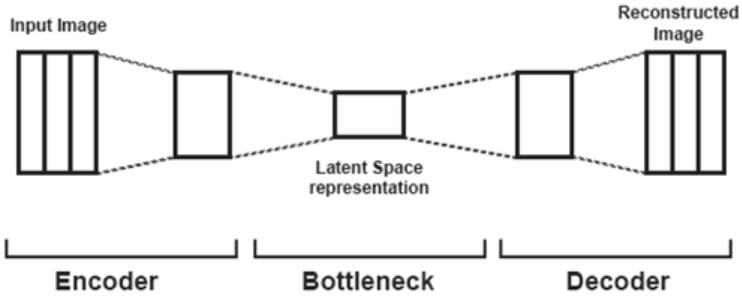

Figure 2.6: Autoencoders

Variational autoencoders (VAEs) extend autoencoders to generative models, enabling the generation of new samples from a learned latent space [29]. Autoencoders, however, have limitations. Information loss may occur during encoding-decoding, leading to the loss of fine-grained details or noise in the input. Interpretability can be challenging as the meaning of each dimension in the latent space may not be clear. Training large-scale autoencoders with complex architectures can be computationally expensive, and tuning hyperparameters requires careful experimentation.

2.7 DL models Challenges/Limitation

Deep learning, while powerful, faces several challenges and limitations that are crucial to consider [28]. One primary concern is the extensive data requirements. Deep learning models typically demand a vast amount of labeled data for effective training, which can be arduous and costly to acquire. Additionally, the computational resources needed for training large models are substantial, making it difficult for researchers or organizations with limited access to high-performance hardware to leverage deep learning effectively. Overfitting is another critical challenge in deep learning. Despite their complexity, deep learning models are susceptible to overfitting, where they perform well on the training data but could not generalize its ability for unseen data. Various regularization techniques have been proposed to mitigate overfitting, but it remains a persistent concern. The training time for deep learning models can be quite

long, particularly for complex architectures and large datasets. This slow training process can hinder rapid experimentation and development, making it important to consider computational efficiency in deep learning research. Although transfer learning has shown promise in leveraging pre-trained models for specific tasks, generalization to different domains remains a challenge. Ensuring that the knowledge captured in pre-trained models can be effectively transferred to new and diverse tasks requires ongoing investigation. Another critical concern is the susceptibility of deep learning models to adversarial attacks. These attacks involve introducing imperceptible changes to input data, causing the model to misclassify. Developing robust models that deter such attacks is an active area of research. Ethical considerations are paramount in deploying deep learning models. Issues related to privacy, bias, and fairness arise when deploying these models in real-world applications. Ensuring that deep learning models are ethically designed and deployed is essential to avoid harmful consequences. Despite their impressive performance, deep learning models are not yet fully understood in terms of how they learn and represent information. In some cases, traditional machine learning methods may still outperform deep learning, especially in domains with limited data availability. The reliance of deep learning on massive datasets can be a limitation when data is scarce.

2.8 Chapter Summary

In this chapter, the basics of DL models and their weaknesses are explored. DL enables unsupervised or automatic models, offering vast applications in healthcare, IoT, smartphones, and more. The challenges include overfitting and limiting generalization to unseen data. Despite these difficulties, DL holds immense potential to revolutionize various industries, enhancing professional landscape and transforming daily lives. By addressing its limitations responsibly and advancing research, DL opens up new possibilities for innovation and progress. Embracing its power unlocks a world where machines learn and adapt seamlessly, shaping a future where AI-driven technologies enrich our existence.

Chapter 3

Transformer based Architecture

3.1 Transformer

The Transformer algorithm is a DL based network configuration that was first introduced in 2017 by Vaswani et al [26]. These are termed as next generation of RNNs and LSTMs and have several distinct benefits to offer, like:

- **Parallel processing:** Improves performance and scalability metrics
- **Bidirectionality:** Allows better understanding of ambiguous words and coreferences [6].

In NLP and machine translation, Transformer model has over extended ability to process sequential input data, such as sentences or paragraphs. Unlike the traditional RNN, where it would process data in series and that too with 100 words maximum, exceeding words than 100 would let RNN forgot previous words it had trained on. The LSTM was introduced to replace the RNN which updated from 100 words maximum to 1000 words but exceeding 1000 words would also make LSTM forgot the previous words it has trained on. So that researcher introduced Transformers with self-attention mechanism. Self-attention mechanism allows to attend to parts of the input when making predictions [26]. Over the range of NLP tasks, Transformer architecture has produced cutting-edge results. It has also been modified and aligned for speech recognition and computer vision.

In the Transformer architecture, the input data consists of a sequence of tokens corresponding words or sub-word levels. Each token is represented by a vector that projects its meaning. The self-attention mechanism computes a weight for each token such that which reflects its importance in the overall context of the input. These weights compute a weighted sum of all the input vectors, and results in a context vector that captures meaning of the input sequence. One of the main advantages of Transformers is its ability to make parallel processing of input sequences of varying but unlimited size. This is because the self-attention mechanism allows the model to compute importance of all input tokens in parallel unlike RNNs that process sequentially one token at a time. This makes the Transformer much faster and more efficient than RNNs, especially for long input sequences. Multi-head attention is another key feature that allows the

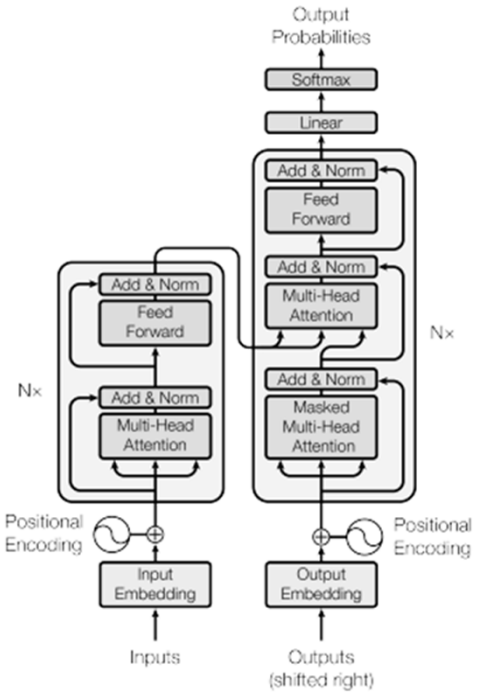

Figure 3.1: Transformer based Algorithm

model to attend to different parts of the input in parallel by dividing the input vectors into multiple subspaces or heads [27]. The attention weights are computed separately for each head, and the results are just concatenated to produce the output. This lets the model to capture and reflect all aspects of the input into a more meaningful output. The use of residual connections and layer normalization is also important for the performance of the Transformers. Overall, the Transformer algorithm is a highly effective tool for a wide variety of NLP tasks due to its ability to process input sequences in parallel using self-attention. Its use of multi-head attention, residual connections, and layer normalization make it a powerful neural network for processing sequential data.

An encoder and a decoder are the two fundamental parts in Transformers. The encoder processes the input sequence and produces trails of hidden representations that spell out meanings. The decoder utilizes these hidden details and generates a learned output sequence. The Encoder and Decoder blocks are explained to a detail as in the following sections.

3.1.1 Encoder

The encoder in Transformers is made up of a series of identical functional blocks or layers. Each such layer comprises of two sub-layers: a multi-head self-attention and a feedforward network sub-layer. Around each of these is a residual connection and a layer normalization. Each of the

sub-output layer is called LayerNorm (x + Sublayer (x)), where Sub-layer (x) is the function that the sub-layer itself implements. All sub-layers and the embedding layers give outputs with the dimension dmodel = 512.

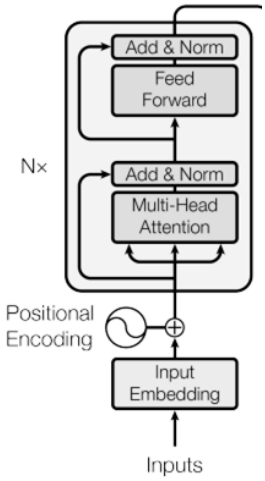

Figure 3.2: Encoder

The multi-head self-attention mechanism computes a weighted sum of the input tokens, where the weights are proportional to the similarity between each token and all other tokens in the input sequence. This enables to capture relationships between different parts of the input sequence [10]. The feedforward neural network is a standard fully connected neural network consisting of two linear transformations with a Rectified Linear Unit (ReLU) activation in between. It takes output from the self-attention sub-layer and passes same through a non-linear activation function to produce new set of hidden representations. The use of a feedforward network lets the model to learn complex non-linear relationships between the input tokens.

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2 \tag{3.1}$$

Each layer in the encoder also has residual connections and layer normalization. The residual connections allow the gradient to flow across the layers, which controls the vanishing gradient problem. The layer normalization stabilizes distribution of the input data and improves performance of the model. The output of the final layer in the encoder is a sequence of hidden representations that capture meaning of the input sequence. These hidden states are used as input to the decoder, which correspondingly generates the output sequence.

3.1.2 Decoder

The decoder in the Transformer configuration is meant for generating the output sequence in reference to the input through utilizing hidden states as maintained by the encoder. In addition,

it also has a series of identical functional blocks or layers consisting of a feedforward network, a multi-head self-attention, and a masked multi-head attention mechanism as each sub-layer.

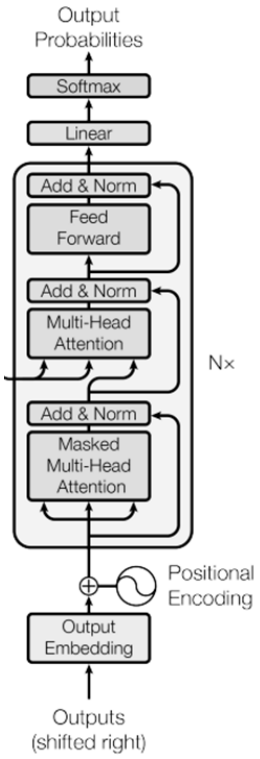

Figure 3.3: Decoder

The multi-head self-attention, functioning similar to the encoder's self-attention, makes up the first sub-layer of each decoder layer. It enables to focus on various portions of the output sequence concurrently and discover the mutual connections between various output tokens. The encoder's hidden representations are attended to by a multi-head attention arrangement in the second sub-layer, wherein the decoder uses the hidden states from the input sequence to generate corresponding output sequence. This attention technique aids the decoder in concentrating on portions of the input sequence that are each output token's most pertinent to. A feedforward neural network, which makes up the third sub-layer, uses a non-linear activation function on the output of the attention mechanism. This aids the decoder in understanding complicated connections between input and output sequences. Each layer in the decoder has residual connections and layer normalization, just like that of the encoder, to enhance the model's performance and stability. The resulting output sequence, which might be a translation, summary, or any other type of output depending on the task, is the output of the final layer of the decoder. One of the benefits of decoder in Transformers is its ability to generate output sequences in parallel. This is because each layer in the decoder can attend to all previous output tokens and the input sequence simultaneously. Overall, the decoder in the Transformer architecture is a powerful tool for generating output sequences based on input hidden states. Its use of multi-head attention

mechanisms, feedforward neural networks, residual connections, and layer normalization makes it capable of sequence-to-sequence tasks in natural language processing and beyond.

3.2 Attention

Attention mechanism used in Transformers allows to focus on parts of the input sequence. This enables the model to assign different weights to each input token based on its relevance to the current output token. In the context of the Transformer architecture, attention can be divided into self-attention and multi-head attention. Self-attention is used within the encoder and decoder to enable each token to attend to other tokens within the same sequence. Multi-head attention, on the other hand, is used only in the decoder to allow each output token to attend to the hidden states of all input tokens. Self-attention involves computing a weighted sum of the input sequence, where the weights are determined by similarity between each token and all other tokens in the same sequence. The similarity is computed through using a dot product between a query vector, a key vector, and a value vector for each token [26]. The query vector is used to determine the relevance of the token to other tokens in the sequence, while the key and value vectors are used to compute the weights. Multi-head attention involves computing multiple weighted sums of a sequence, each with a different set of query, key, and value vectors. These different vector sets enable the model to capture various aspects of the sequence in parallel and then combine to produce a single output vector.

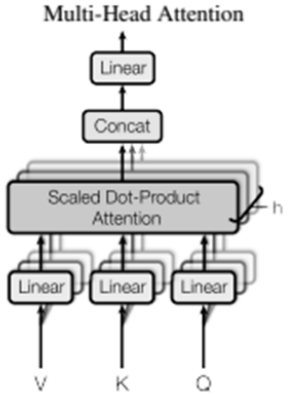

Figure 3.4: Attention

The weights in both types of attention mechanisms are computed using a softmax function, which ensures that the weights add up to one can be interpreted as a probability distribution over the whole chunk of tokens. This allows the model to assign higher weights to tokens that are more relevant to the current one. Application of attention in Transformers enables the model to handle sequences of variable length without the requirement for fixed-length representations and allows the model to capture complicated interactions between input and output sequences.

3.2.1 Scaled Dot-Product

Scaled Dot-Product Attention is a variant of the attention mechanism. The dot product operation can result in very large values for the weights, which can cause numerical instability and slow down the training process. To address this issue, the dot product is scaled by the square root of the dimension of the key vector [12]. This scaling ensures that the weights remain in a reasonable range and do not explode or vanish during training. The scaled dot product is then passed through a softmax function, which normalizes the weights to ensure that they add up to 1 and can be interpreted as a probability distribution. The resulting weighted sum of the values then compute output of the attention mechanism.

$$Attention\,(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3.2}$$

The Scaled Dot-Product mechanism is used multiple times in the Transformer architecture, allowing to attend to different parts of the sequence with different queries, keys and values. This enables the model to capture complex relationships between input and output sequences and process variable-length sequences. In overall, the Scaled Dot-Product Attention mechanism is a crucial element which has significantly improved the performance of Transformers in natural language processing tasks.

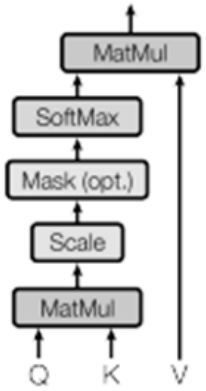

Figure 3.5: Scaled Dot-Product Attention

3.2.2 Numerical Calculation

$$\boxed{Haider} \quad \boxed{eating} \quad \boxed{cake} \tag{3.3}$$

$$\boxed{0.12|\,0.53} \quad \boxed{0.71|\,0.43} \quad \boxed{0.48|\,0.65} \tag{3.4}$$

$$P = \qquad 0 \qquad 1 \qquad 2 \tag{3.5}$$

$$\therefore \textit{ Applying Positional embedding} \tag{3.6}$$

$$\boxed{0.12|\ 0.53|\ 0.22} \quad \boxed{0.71|\ 0.43|\ 0.23} \quad \boxed{0.48|\ 0.65|\ 0.24} \tag{3.7}$$

We get Q,K and V,

$$Q = \begin{bmatrix} 0.12 & 0.53 & 0.22 \\ 0.71 & 0.43 & 0.23 \\ 0.48 & 0.65 & 0.24 \end{bmatrix} \tag{3.8}$$

$$K = \begin{bmatrix} 0.12 & 0.53 & 0.22 \\ 0.71 & 0.43 & 0.23 \\ 0.48 & 0.65 & 0.24 \end{bmatrix} \tag{3.9}$$

Taking transpose of K,

$$K^T = \begin{bmatrix} 0.12 & 0.71 & 0.48 \\ 0.53 & 0.43 & 0.65 \\ 0.22 & 0.23 & 0.24 \end{bmatrix} \tag{3.10}$$

$$V = \begin{bmatrix} 0.12 & 0.53 & 0.22 \\ 0.71 & 0.43 & 0.23 \\ 0.48 & 0.65 & 0.24 \end{bmatrix} \tag{3.11}$$

Now Calculating attention weights,

$$a = Q.K^T = \begin{bmatrix} 0.12 & 0.53 & 0.22 \\ 0.71 & 0.43 & 0.23 \\ 0.48 & 0.65 & 0.24 \end{bmatrix} * \begin{bmatrix} 0.12 & 0.71 & 0.48 \\ 0.53 & 0.43 & 0.65 \\ 0.22 & 0.23 & 0.24 \end{bmatrix} \tag{3.12}$$

$$a_{11} = (0.12 \times 0.12 + 0.53 \times 0.71 + 0.22 \times 0.48) = (0.4963) \tag{3.13}$$

$$a_{12} = (0.12 \times 0.71 + 0.53 \times 0.43 + 0.22 \times 0.23) = (0.3637) \tag{3.14}$$

$$a_{13} = (0.12 \times 0.48 + 0.53 \times 0.65 + 0.22 \times 0.24) = (0.4949) \tag{3.15}$$

$$a_{21} = (0.71 \times 0.12 + 0.43 \times 0.53 + 0.23 \times 0.22) = (0.3637) \tag{3.16}$$

$$a_{22} = (0.71 \times 0.71 + 0.43 \times 0.43 + 0.23 \times 0.23) = (0.7419) \tag{3.17}$$

$$a_{23} = (0.71 \times 0.48 + 0.43 \times 0.65 + 0.23 \times 0.24) = (0.4163) \tag{3.18}$$

$$a_{31} = (0.48 \times 0.12 + 0.65 \times 0.53 + 0.24 \times 0.22) = (0.4549) \tag{3.19}$$

$$a_{32} = (0.48 \times 0.71 + 0.65 \times 0.43 + 0.24 \times 0.23) = (0.4163) \tag{3.20}$$

$$a_{33} = (0.48 \times 0.48 + 0.65 \times 0.65 + 0.24 \times 0.24) = (0.7105) \tag{3.21}$$

$$a = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 0.4963 & 0.3637 & 0.4949 \\ 0.3637 & 0.7419 & 0.4163 \\ 0.4549 & 0.4163 & 0.7105 \end{bmatrix} \tag{3.22}$$

Now we get weighted word vector,

$$a' = a.V = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} * \begin{bmatrix} V_{11} & V_{12} & V_{13} \\ V_{21} & V_{22} & V_{23} \\ V_{31} & V_{32} & V_{33} \end{bmatrix} \tag{3.23}$$

$$a_{11} = (0.4963 \times 0.12 + 0.3637 \times 0.71 + 0.4549 \times 0.48) = (0.5361) \tag{3.24}$$

$$a_{12} = (0.4963 \times 0.53 + 0.3637 \times 0.43 + 0.4549 \times 0.65) = (0.7151) \tag{3.25}$$

$$a_{13} = (0.4963 \times 0.22 + 0.3637 \times 0.23 + 0.4549 \times 0.24) = (0.3020) \tag{3.26}$$

$$a_{21} = (0.3637 \times 0.12 + 0.7419 \times 0.71 + 0.4163 \times 0.48) = (0.7702) \tag{3.27}$$

$$a_{22} = (0.3637 \times 0.53 + 0.7419 \times 0.43 + 0.4163 \times 0.65) = (0.7823) \tag{3.28}$$

$$a_{23} = (0.3637 \times 0.22 + 0.7419 \times 0.23 + 0.4163 \times 0.24) = (0.3505) \tag{3.29}$$

$$a_{31} = (0.4549 \times 0.12 + 0.4163 \times 0.71 + 0.7105 \times 0.48) = (0.6912) \tag{3.30}$$

$$a_{32} = (0.4549 \times 0.53 + 0.4163 \times 0.43 + 0.7105 \times 0.65) = (0.8819) \tag{3.31}$$

$$a_{33} = (0.4549 \times 0.22 + 0.4163 \times 0.23 + 0.7105 \times 0.24) = (0.3663) \tag{3.32}$$

Now passing the weighted words through normalization layer,

$$a = \frac{a'}{\sqrt{d_k}} = \frac{a'}{\sqrt{9}} = \frac{a'}{3} \tag{3.33}$$

$$= \begin{bmatrix} \frac{a_{11}}{3} & \frac{a_{12}}{3} & \frac{a_{13}}{3} \\ \frac{a_{21}}{3} & \frac{a_{22}}{3} & \frac{a_{23}}{3} \\ \frac{a_{31}}{3} & \frac{a_{32}}{3} & \frac{a_{33}}{3} \end{bmatrix} = \begin{bmatrix} \frac{0.5361}{3} & \frac{0.7151}{3} & \frac{0.3020}{3} \\ \frac{0.7702}{3} & \frac{0.7823}{3} & \frac{0.3505}{3} \\ \frac{0.6912}{3} & \frac{0.8819}{3} & \frac{0.3663}{3} \end{bmatrix} \tag{3.34}$$

$$= \begin{bmatrix} 0.1787 & 0.2383 & 0.1006 \\ 0.2567 & 0.2607 & 0.1168 \\ 0.2304 & 0.2939 & 0.1221 \end{bmatrix} \tag{3.35}$$

Now passing through softmax activation function,

$$a = \frac{e^{ai}}{\sum_{j=1}^{n} e^{aj}} \tag{3.36}$$

$$\therefore e = exponantial\ which\ is\ equal\ to\ 2.718 \tag{3.37}$$

$$e^{0.1787} = 1.1956 \tag{3.38}$$

$$e^{0.2383} = 1.2690 \tag{3.39}$$

$$e^{0.1006} = 1.1058 \tag{3.40}$$

$$e^{0.2567} = 1.2926 \tag{3.41}$$

$$e^{0.2607} = 1.2978 \tag{3.42}$$

$$e^{0.1168} = 1.1238 \tag{3.43}$$

$$e^{0.2304} = 1.2591 \tag{3.44}$$

$$e^{0.2939} = 1.3416 \tag{3.45}$$

$$e^{0.1221} = 1.1298 \tag{3.46}$$

$$\sum_{j=1}^{n} e^{zi} = \quad 11.0151 \tag{3.47}$$

$$Softmax\left(Z_{11}\right) = \frac{1.1956}{11.0151} = 0.1085 \tag{3.48}$$

$$Softmax\left(Z_{11}\right) = \frac{1.2690}{11.0151} = 0.1152 \tag{3.49}$$

$$Softmax\left(Z_{11}\right) = \frac{1.1058}{11.0151} = 0.1003 \tag{3.50}$$

$$Softmax\left(Z_{11}\right) = \frac{1.2926}{11.0151} = 0.1173 \tag{3.51}$$

$$Softmax\left(Z_{11}\right) = \frac{1.2978}{11.0151} = 0.1178 \tag{3.52}$$

$$Softmax\left(Z_{11}\right) = \frac{1.1238}{11.0151} = 0.1020 \tag{3.53}$$

$$Softmax\left(Z_{11}\right) = \frac{1.2591}{11.0151} = 0.1143 \tag{3.54}$$

$$Softmax\left(Z_{11}\right) = \frac{1.3416}{11.0151} = 0.1217 \tag{3.55}$$

$$Softmax\left(Z_{11}\right) = \frac{1.1298}{11.0151} = 0.1025 \tag{3.56}$$

$$Z' = \begin{bmatrix} 0.1085 & 0.1152 & 0.1003 \\ 0.1173 & 0.1178 & 0.1020 \\ 0.1143 & 0.1217 & 0.1025 \end{bmatrix} \tag{3.57}$$

$$Z^1 + Z^2 + Z^3 \tag{3.58}$$

$$\boxed{Concatenate} \tag{3.59}$$

$$Z^T = head \tag{3.60}$$

The Z^H will be sent to Decoder block (which is responsible for translating the languages), to do whole process again but in reverse method and hence at the end we will get the output but in different language.

$$\boxed{Decoder} \tag{3.61}$$

$$\boxed{Z^H} \tag{3.62}$$

$$\boxed{Process\ again\ but\ in\ reverse\ method} \tag{3.63}$$

$$\boxed{Spanish\ Language} \quad Haider \quad comiendo \quad pastel \tag{3.64}$$

3.3 Transformer Based Models

Since its introduction, the Transformer model has served as the foundation for numerous other models in the field of NLP. Encoder and decoder are two major elements use in original architecture. BERT uses encoders and GPT uses decoders only. Some examples of Transformer-based models include:

1. **BERT:** It refers to Bidirectional Encoder Representation from Transformer. It was first introduced by Google [25]. It is a pre-trained language model that can be fine-tuned for a wide variety of NLP tasks such as text classification, named entity recognition, and automatic question answering.

2. **GPT:** It Stands for Generative Pre-trained Transformer. It was introduced by OpenAI and has several variations, with GPT-2 and GPT-3 being the two most well-known. These are pre-trained language models that can generate human-like text in response to a given prompt or question.

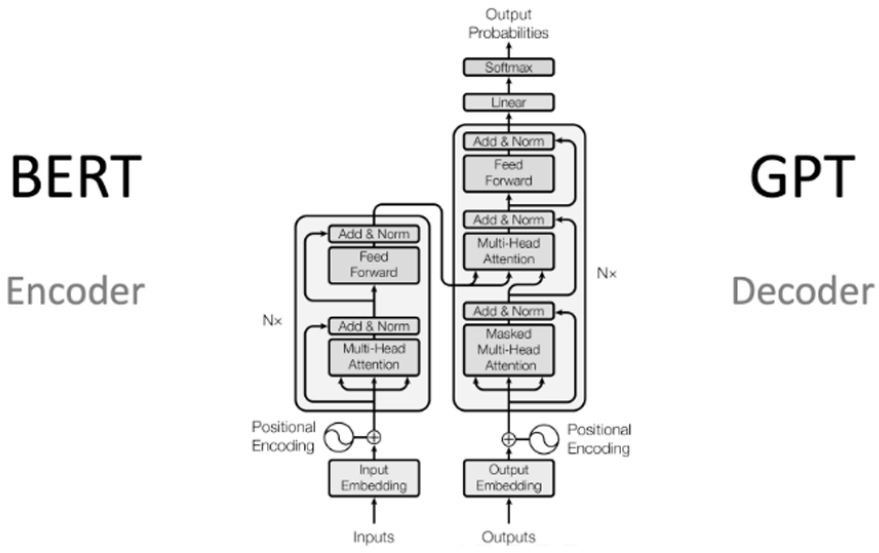

Figure 3.6: BERT and GPT

3.3.1 BERT Model

The BERT is a pre-trained deep learning model for NLP developed in 2018 by Devlin *et all* at Google [21]. It uses the transformer architecture. The key attribute of BERT is its ability to pre-train on huge amounts of text data using distributed computing system and memorize the underlying patterns and relationships in the language syntax and semantics. This pre-training is done through two techniques: masked language modeling (MLM) and next sentence prediction (NSP). A pre-trained BERT can be easily fine-tuned on specific NLP tasks. For

fine-tuning we just add a task-specific layer on top of the pre-trained BERT and then train the network on a smaller dataset related to the specific task. The effectiveness of BERT in NLP has been demonstrated through various benchmark datasets [8], and has revealed advanced performance. The availability of pre-trained BERT models has also facilitated the researchers and developers to apply it to various NLP tasks and achieve high levels of accuracy with less data and computing resources. The two techniques for pre-training of BERT are elaborated in the following sections.

3.3.2 Masked Language Modeling

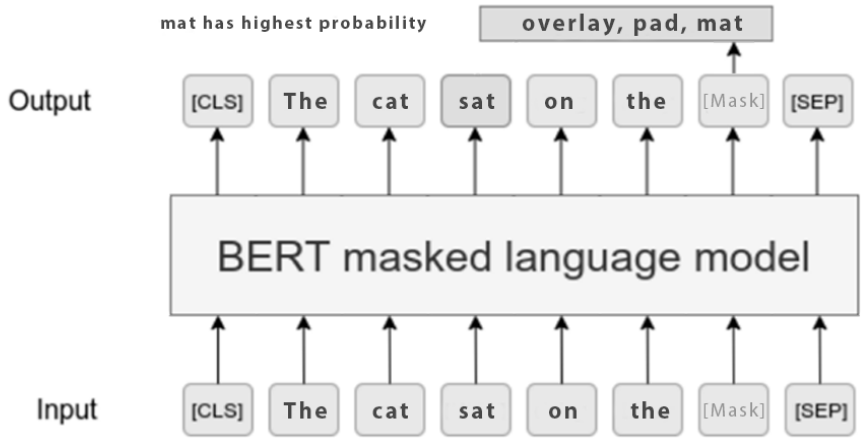

Figure 3.7: MLM structure

MLM is one of a pre-training technique used for BERT. In MLM, a certain percentage of input tokens in a sentence are randomly masked. The model is then trained to predict the masked ones based on the context of surrounding tokens. For example, for a sentence like "The cat sat on the ___", the word "mat" may be randomly selected and replaced with a special masked token, resulting in "The cat sat on the [MASK]". The BERT model is then trained to predict the masked token in reference to surrounding tokens, which in this case is "The cat sat on the", giving the model a clue that the missing word is likely to be "mat". The MLM task is important because it allows the BERT to learn bidirectional representations of the language. This means that the model can understand meaning of a word based on its context in both the forward and backward directions, which helps it to learn more accurate and robust language representations. The BERT model learns broad language representations during pre-training by being exposed to a huge quantum of text data, which enables it to be easily tailored for certain downstream NLP applications like text categorization, named entity recognition and question answering etc. Many benchmark datasets have shown that BERT is effective at completing these tasks, with better results. In the fig 3.7 [CLS] means Classification and [SEP] means Separator. These are special tokens that are added to sentences for Bert to easily understand it.

3.3.3 Next Sentence Prediction

NSP is another pre-training technique used for BERT. In NSP, the model is provided with two consecutive sentences and trained to predict whether the second sentence is a continuation of the first or not. For instance, for the two phrases: "Ali is a good guy. He will play a game.", the BERT model anticipates that if the second sentence would come after the first. It is important because it allows to understand the relationship as how the two are connected. This is particularly useful for tasks like natural language inference which tries to find whether one sentence entails, contradicts, or is neutral with respect to the other.

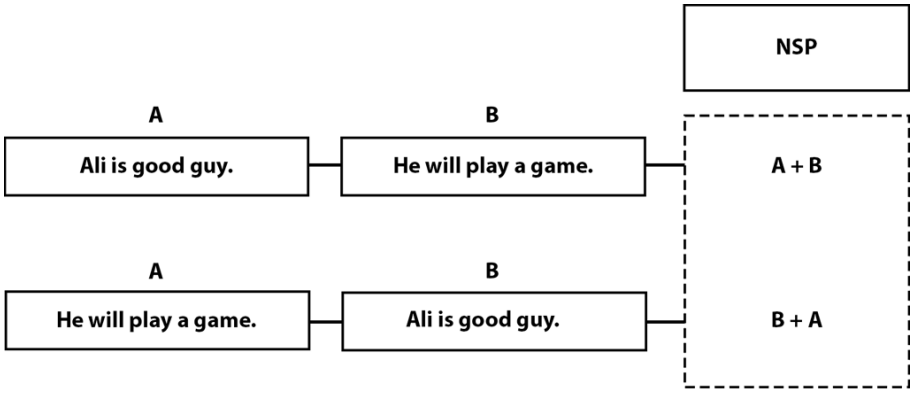

Figure 3.8: NSP Pipeline

3.4 WordPiece

BERT uses WordPiece tokenizer to represent words in a more flexible and efficient manner [7], to improve performance in various tasks [10]. WordPiece is a subword-based text segmentation algorithm which is used in NLP to tokenize words into smaller units [17]. The WordPiece was first introduced by Google in 2016 and has since been widely used in various NLP applications. The WordPiece algorithm works by iteratively merging the most frequent pair of consecutive character sequences in a corpus until a predefined vocabulary size is reached. This process produces subword units that represent any word in the corpus. The resulting vocabulary is typically much smaller than the original vocabulary of words, which can reduce the data sparsity and improve generalization of the model. For example, the word "university" can be segmented into "un", "##iver", and "##sity" using the WordPiece algorithm. The "##" prefix indicates that the subword unit is a continuation of the previous unit, which allows the algorithm to capture morphological information such as prefixes, suffixes, and inflections.

The use of subword units such as WordPiece allows to better handle out-of-vocabulary (OOV) words and improves the model ability to handle rare or misspelled words. It also helps to

Figure 3.9: Word Piece Tokenizer Example

capture the similarity between words that share common subword units, which help improve text classification, language modeling, and machine translation tasks.

3.5 Types of Bert Model

There are two types of BERT models which are given as below:

- **BERT-Base:** This base version has 12 layers and 110 million parameters, making it suited for smaller tasks or fine-tuning on larger datasets. Researchers use this model to evaluate performance compared to other models.
- **BERT-Large:** This larger version boasts 24 layers and 340 million parameters, designed for more complex NLP tasks demanding higher accuracy and performance. It has been used to achieve more accurate results on high-performing computational systems.

There are several other variants of BERT models modified by developers and practioners for various NLP tasks. Some of the most common types of BERT models include:

- **Multilingual BERT:** This is a version of BERT that has been trained on multiple languages. It can handle input text in various languages and can be used for cross-lingual NLP tasks.
- **BioBERT:** This is a specialized version that has been pre-trained on biomedical texts. It is fine-tuned for various biomedical tasks such as named entity recognition, relation extraction, and question-answering etc.
- **RoBERTa:** This BERT variation employs "dynamic masking," a distinct pre-training strategy. On a number of NLP benchmarks, it has been demonstrated to perform better than BERT.
- **DistilBERT:** This is a smaller and faster version of BERT that has been trained using a knowledge distillation technique. It has fewer parameters and can be fine-tuned more quickly than the larger BERT models.

3.6 Applications of Bert Model

- **Text Classification:** BERT has been used for text classification as in sentiment analysis, spam detection and topic segregation etc. The pre-trained BERT can be fine-tuned to improve performance in a particular classification task [21].
- **Question Answering:** BERT has been used for question-answering tasks like SQuAD (Stanford Question Answering Dataset) and TriviaQA. The pre-trained model is fine-tuned by feeding it with the question and the related paragraph or document, to find the answer to the question.
- **Named Entity Recognition:** BERT has been used to extract entities such as names, organizations and locations from text data. Here BERT is fine-tuned on a carefully compiled dataset to improve performance in recognizing entities.
- **Language Translation:** BERT has been used for machine translation purposes to translate text from one language into another. For this purpose, pre-trained BERT is fine-tuned on given languages pair to improve its translation worthiness.
- **Text Summarization:** To derive brief summary out of a lengthy text, BERT has shown promising results. To enhance BERT performance for summarization, it is fine-tuned on a particularly designed dataset.

3.7 Chapter Summary

This chapter introduced Transformer models and its basic algorithm. Transformer is a deep learning models which is made up of stack of BERT and GPT. Also, talked about BERT and its use strategy that includes applying pre-trained models to address new problems, allowing for more efficient and faster model training [15]. The chapter addresses the advantages of transfer learning models and looks at how it may be used in NLP and other domains [21].

Chapter 4

Proposed Methodology

The ability to accurately classify text into multiple categories has become increasingly important in various domains. With the advent of pre-training techniques in natural language processing models like BERT, remarkable performance has been witnessed in a variety of language understanding tasks. However, fine-tuning for multi-classes text categorization still remains an area of active research.

This thesis proposes a methodology for fine-tuning BERT for multi-class text classification using the latest versions and platforms. By leveraging the power of transfer learning and extensive language representation capabilities of BERT, the aim to come up with a robust and accurate BERT model that effectively categorizes text into multiple classes. By integrating the power of BERT with strategic cross-validation techniques, the desire to go beyond what is possible with traditional models. This amalgamation seeks to not only expedite the classification but also elevate accuracy of the results and accommodate resource constraints. The proposed methodology involves several key steps. First, a large-scale labeled dataset suitable for fine-tuning regarding multi-class text classification job is collected. Next, an appropriate BERT model, already pre-trained on huge quantum of language data to better capture language understanding, is selected and acquired. The labeled dataset is then prepared for fine-tuning by converting the text data into BERT-compatible input format. The BERT model is modified by adding a classification layer, and the fine-tuning process is carried out using a supervised learning technique. It may evaluate the model's performance before using it on the validation set since performance evaluation happens during the cross-validation procedure on the training data. Necessary optimizations and adjustments are made based on the evaluation results. The final model is tested on an unseen dataset to obtain unbiased performance metrics. The proposed methodology is compared with existing baselines and state-of-the-art models to assess its effectiveness and superiority. The successful implementation of this methodology has the potential to further enhance viability of multi-class text classification, enabling more accurate and efficient categorization across various applications. The findings of this research can contribute to advancements in NLP and have practical implications in areas such as automated content classification, sentiment analysis, and topic modeling, etc. Furthermore, this work open venues for future enhancements and explores research directions in the field of fine-tuning large-scale BERT models for multi-class text classification.

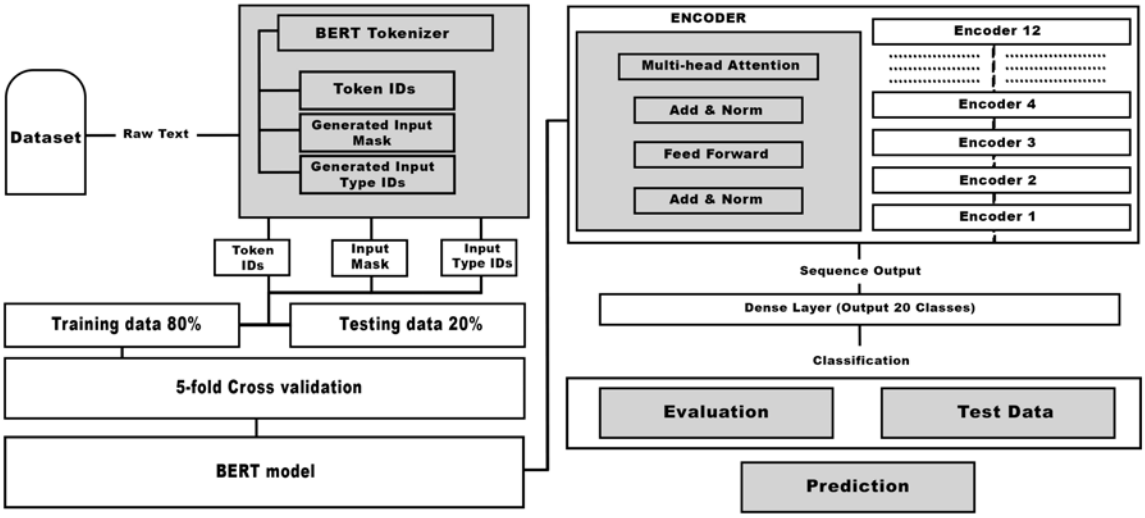

Figure 4.1: Proposed Methodology Structure

4.1 Proposed Algorithm

In this section, the algorithm for the fine-tuned large-scale BERT model for multi-class text classification is presented. The algorithm encompasses various components, including word embedding, hidden layers, activation functions, optimizers, and evaluation functions. Each component has bearing on and exhibit important role in enhancing overall performance and accuracy of the model.

4.1.1 Word Embedding

Word embedding is a critical step in NLP tasks [3]. In this proposed methodology, the Wordpiece tokenization technique is employed to break down words into sub-word units. Wordpiece tokenization enables the model to handle terms that are not commonly used and to collect more precise data. It splits words into smaller units, enabling the model to better understand complex morphological structures and handle rare or unknown words effectively. Pre-trained Wordpiece embeddings, such as those obtained from the BERT model, are utilized to represent the sub-word units. These embeddings provide a dense vector representation for each sub-word, capturing semantic and syntactic information that is essential for multi-class text classification.

4.1.2 Hidden Layers

The hidden layers in the proposed algorithm are composed of BERT's attention layers and additional dense layers. BERT is a powerful language model that employs attention mechanisms to capture the contextual relationships between words. By utilizing BERT's attention layers as

the initial hidden layers, the model can effectively encode the input text and capture important semantic information. Additionally, one or more dense layers are incorporated to further capture complex patterns and relationships in the data. Dense layers consist of fully connected neurons, allowing the model to learn non-linear representations of the input. These hidden layers play a crucial role in extracting meaningful features from the text and enables the model to make accurate predictions for multi-class text classification tasks.

4.1.3 Activation Functions

Activation functions induce non-linearity, enabling to learn complex connection and capture non-linear dependencies within the textual data. In the proposed methodology, the softmax activation function is utilized in the final layer of the model. The softmax function normalizes the output probabilities across multiple classes and provides a probability distribution across the classes. It ensures the predicted class probabilities add up to 1, allowing for a reliable and interpretable predictions for multi-class classification tasks. By employing the softmax activation function, the model can assign probabilities to each class and make confident predictions based on these probabilities.

$$S(x^i) = \frac{e^{xi}}{\sum_{j=1}^{n} e^{xj}} \tag{4.1}$$

4.1.4 Optimizers

Optimization algorithms play a crucial role in training the model and updating its network parameters. In the proposed methodology, the RAdam optimizer is implemented. Output from Adam optimizer passes through rectified linear unit to limit maxima and force improved convergence speed and performance. The Adam optimizer uses the predefined learning rate which allow for a faster convergence and better handling of sparse gradients. By incorporating rectified linear units, which introduces non-linearity into the optimization process, RAdam enhances its ability to learn more complex patterns and achieve better generalization. The choice of the RAdam optimizer in the proposed methodology is to optimize the training process and improve fine-tuning towards multi-class text classification.

4.1.5 Evaluation Function

To evaluate the fine-tuned BERT model for multi-class text classification, various evaluation metrics are employed. These metrics help assess the model's classification performance across multiple classes. A confusion matrix is generated, presenting a comprehensive overview of predictions and their alignment with respect to the true labels. The model's throughput for each class is then examined, including the quantities of true positives, true negatives, false positives, and false negatives. Metrics like F1 score, recall, accuracy, and precision are also computed. Accuracy is used to gauge the overall level of the model predictions. Precision indicates the model's ability to reduce false positives, calculating the percentage of accurately predicted positive cases out of all instances projected as positive. Recall calculates the proportion of correctly predicted positive instances out of all actual positive instances, representing the model's ability

to minimize false negatives. The F1 score provides a more balanced measure of precision and recall, considering both false positives and false negatives. These evaluation metrics help understand the model's performance, identify strengths and weaknesses, and make informed decisions for further optimization and fine-tuning. Macro avg is the average of the precision, recall, and f1-score for each class. Weighted avg is the average of the precision, recall, and f1-score for each class, weighted by the support for each class.

4.1.6 Cross-validation

Cross-validation is a technique widely employed for the assessment of machine learning models and their ability to generalize [9]. It operates by dividing the dataset into distinct subsets, often referred to as "folds." The model is trained on all the data except one subset, constituting the training set, and then evaluated on the only subset earlier separated from the training lot, and termed as the validation set. This process is repeated iteratively for all folds. ensuring that the model is trained and tested using all the provided data.

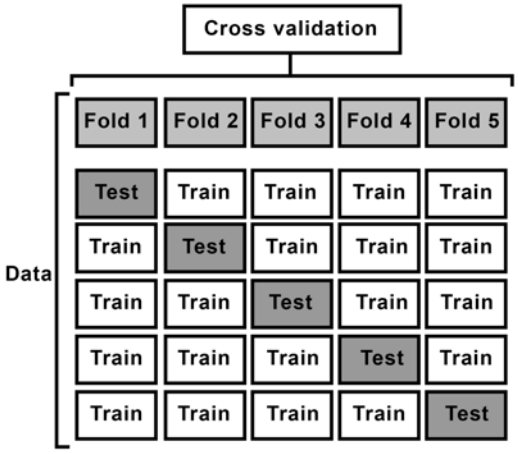

Figure 4.2: 5-fold Cross Validation

The utilization of cross-validation offers several valuable advantages. One primary benefit is its role in mitigating overfitting, as it subjects the model to diverse data partitions, thereby reducing the impact of data-specific idiosyncrasies. This approach yields a more balanced representation of the model's abilities and limitations. It optimally exploits available data by ensuring that each instance contributes to both training and validation across distinct iterations. In essence, cross-validation enhances the reliability, robustness, and generalizability of machine learning models. In this proposed methodology, 5-fold cross-validation is implemented.

Dataset name	Changes	Samples
20news-19997.tar.gz	Original 20 Newsgroups data set	20,000
20news-18828.tar.gz	20 Newsgroups; duplicates removed, only "From" and "Subject" headers	18,828

Table 4.1: 20newsgroup dataset types

By implementing all the steps above, the aim is to enhance the model's accuracy and ability to classify text into multiple classes. The utilization of BERT's attention and dense layers enables the model to capture contextual relationships and complex patterns effectively. The proposed methodology can be applied to various applications such as sentiment analysis, topic modeling, and content classification, among others.

4.1.7 Dataset

The 20 Newsgroups collection is a widely used benchmark dataset in NLP and text classification tasks. On a wide range of subjects, including sports, politics, technology, and more, it comprises posts from many newsgroups. The dataset offers brief textual information that can be used for multi-class text classification by concentrating on the "From" and "Subject" headings. The dataset serves as the basis for training, validation, and testing Pre-trained BERT model for multi-class text classification. The documents' "From" and "Subject" headers provide valuable information for predicting the appropriate category or topic associated with each document. By utilizing this dataset, the proposed methodology aims to develop a robust and accurate model capable of accurately classifying text into multiple classes. The performance of the model will be gauged on metrics like accuracy, precision, recall and F1 score using two of the 20newsgroup datasets independently for fine-tuning and evaluation.

The **20news-18828.tar.gz** dataset, which consists of documents from the 20 Newsgroups collection. The dataset has been pre-processed to remove duplicate documents, resulting in 18,828 unique documents. Only the "From" and "Subject" headers of each document are retained for text classification. By removing duplicates, we eliminate potential biases and redundancy in the data, enabling more accurate and reliable model training and evaluation. While 20news-19997 is a raw data of same dataset containing all samples in the original form. Table 4.1 shows type of datasets discussed here.

Utilizing the 20news-18828 and 20news-19997 datasets, the proposed methodology develops a robust and accurate classifier that yields state-of-the-art results.

4.2 Experimental Details

In this section, The development of practical implementation of the proposed methodology will be discussed, providing intricate details of its execution:

Libraries	Descriptions
import os	facilitates interaction with the operating system.
import codecs	supports diverse character encodings for efficient data handling.
import tensorflow as tf	forms the core of our neural network design and training.
import tensorflow_io as tfio	extends TensorFlow's capabilities for handling various data formats.
from tqdm import tqdm	enhances iterative processes with informative progress bars.
from chardet import detect	automatically detects the encoding of byte streams.
import tensorflow.keras	empowers us with the Keras API from TensorFlow for building models.
from tensorflow_addons.optimizers import RectifiedAdam	introduces the RAdam optimizer, vital for optimizing our model.
from keras_bert import load_trained_model_from_checkpoint	equips us with tools tailored for BERT model operations.
import numpy as np	plays a crucial role in numerical array operations.
import pandas as pd	offers versatile data manipulation capabilities.
from sklearn.model_selection import KFold	orchestrates k-fold cross-validation.
from sklearn.metrics import accuracy_score	calculates accuracy metrics.
from tensorflow.keras.initializers import GlorotNormal	enables weight initialization.
import tensorflow.compat.v1 as tf	ensures compatibility with TensorFlow version 1.x.

Table 4.2: Imported Libraries

4.2.1 Initialization and Configuration

The foundation of this experimentation is established through the inclusion of essential libraries and configurations. The table 4.2 contains all the necessary components required for the seamless implementation of this methodology:

4.2.2 Hyperparameters and Constants

The detail of hyperparameters used in the proposed research is given in table 4.3.

4.2.3 Data Reprocessing and Splitting

This phase delves into data reprocessing. Here, BERT's language capabilities intertwine with the dataset through tokenization. Tokenized data undergoes shuffling, priming it for division. Subsequently, the data is partitioned into 80% training and the remaining 20% is divided for validation and testing purpose, ensuring balanced representation.

LR	0.0001
SEQ_LEN	128
BATCH_SIZE	50
EPOCHS	7
NUM_FOLDS	5
RANDOM_STATE	42

Table 4.3: Hyperparameter and parameter

4.2.4 Model Compiling

- The climax of model preparation arrives with compilation. Central to this step is the RAdam optimizer, adept at steering training with adaptive learning rates. Accompanying this is the softmax activation function, an enabler of probabilistic clarity. Nestled within the final model layer, it normalizes output probabilities into interpretable predictions.

```
tf.enable_eager_execution()


optimizer = RectifiedAdam(learning_rate=LR)

inputs = model.inputs[:2]
dense = model.get_layer('NSP-Dense').output
outputs = tensorflow.keras.layers.Dense(units=20, activation='softmax',  kernel_initializer=GlorotNormal(seed=42))(dense)

model = tensorflow.keras.models.Model(inputs, outputs)
model.compile(
  optimizer=optimizer,
  loss='sparse_categorical_crossentropy',
  metrics=['sparse_categorical_accuracy'],
)
```

Figure 4.3: Model Compiling

  - In the first line of code, **Eager execution** is enabled, allowing TensorFlow operations to be evaluated immediately. This feature is particularly useful for interactive debugging and development, as it facilitates tracking and understanding of operations in a more intuitive manner.
  - In the second line, **RAdam optimizer** is defined, and the learning rate is specified. The optimizer helps in adjusting the weights to minimize loss function during training.
  - In the third line, the input tensors required for the modified model are extracted from the original BERT model. These input tensors typically include tokenized text and segment information.
  - In the fourth line, the output tensor of the **NSP-Dense** layer in the BERT model is retrieved. This output will be used as input for the final classification layer, allowing to capture essential features for an accurate classification.

- In the fifth line, a new classification layer is created with 20 units and a **softmax activation** function. This layer's purpose is to produce probability distributions over the possible classes, and the **GlorotNormal initializer** with a specified seed is used to initialize the layer's weights.
- In the sixth line, the model is constructed by combining the modified input tensors and the newly created classification layer. This results in a new model that includes the changes made for multi-class text classification.
- In the seventh line, the model is compiled with the defined optimizer, **sparse_categorical_crossentropy** as the loss function suitable for multi-class classification with integer labels, and **sparse_categorical_accuracy** as the evaluation metric. The compilation step prepares the model for training by specifying how it should update its weights and measure its performance during training and validation.

4.2.5 Cross-Validation Strategy

This section introduces an essential cross-validation strategy, foundational for comprehensive evaluation. By segmenting the dataset into distinct folds, each containing unique training and validation data, a meticulous assessment unfolds. Iterative training, evaluation, and refinement culminate in a holistic understanding of model performance,robustness, and adaptability.

- In the first line of code, a K-Fold cross-validator is initialized with the specified number of folds **NUM_FOLDS**, enabling the dataset to be split into multiple subsets for cross-validation. Shuffle is set to True to randomize the order of instances before splitting, and a fixed **random_state (42)** is used for reproducibility.
- In the second to fifth lines, lists are created to store various values and metrics for each fold. **all_predictions** will store the predictions from each fold. **train_loss_list** and **train_accuracy_list** will store the training loss and accuracy for each fold, while **val_loss_list** and **val_accuracy_list** will store the validation loss and accuracy.
- In the sixth line, a counter variable **fold** is initialized to keep track of the current fold being processed. The loop iterates through each fold created by K-Fold.
- In the next few lines, inside the loop, the indices for training and testing instances in the current fold are extracted. **train_indices** and **test_indices** represent the indices of instances used for training and validation, respectively.
- In the next few lines, the data is split based on the extracted indices for both training and testing sets. **train_x_fold** and **train_y_fold** represent the input features and labels for the training set, while **test_x_fold** and **test_y_fold** represent those for the validation set.
- In the following lines, the model is trained using the training data of the current fold. The history object captures various training metrics, such as loss and accuracy,

```
kf = KFold(n_splits=NUM_FOLDS, shuffle=True, random_state=42)

all_predictions = []

train_loss_list = []
train_accuracy_list = []
val_loss_list = []
val_accuracy_list = []

fold = 1
for train_indices, test_indices in kf.split(train_x[0]):
    print(f"Fold {fold}/{NUM_FOLDS}")
    fold += 1

    train_x_fold = [train_x[0][train_indices], train_x[1][train_indices]]
    train_y_fold = train_y[train_indices]
    test_x_fold = [train_x[0][test_indices], train_x[1][test_indices]]
    test_y_fold = train_y[test_indices]

    history = model.fit(
        train_x_fold,
        train_y_fold,
        epochs=EPOCHS,
        batch_size=BATCH_SIZE,
        verbose=1,
        validation_data=(test_x_fold, test_y_fold)
    )

    predicts_fold = model.predict(test_x_fold, verbose=1).argmax(axis=-1)
    accuracy = accuracy_score(test_y_fold, predicts_fold)
    print(f"Fold {fold - 1}/{NUM_FOLDS} - Accuracy: {accuracy:.4f}")
    all_predictions.append(predicts_fold)

    train_loss = history.history['loss']
    train_accuracy = history.history['sparse_categorical_accuracy']
    val_loss = history.history['val_loss']
    val_accuracy = history.history['val_sparse_categorical_accuracy']

    # Append the values to the lists
    train_loss_list.append(train_loss)
    train_accuracy_list.append(train_accuracy)
    val_loss_list.append(val_loss)
    val_accuracy_list.append(val_accuracy)
```

Figure 4.4: Cross validation implementation

over the epochs specified in **EPOCHS**, with a batch size of **BATCH_SIZE**. The validation data, composed of **test_x_fold** and **test_y_fold**, is used to evaluate the model's performance after the training.

- In subsequent lines, the model's predictions are obtained using the validation data of the given fold. The predictions are converted to class labels using the argmax function. The accuracy of the predictions is computed using the actual validation labels, and it is printed along with the current fold number.
- In the last few lines, Metrics from the history object such as training loss, training accuracy, validation loss, and validation accuracy, are extracted and stored in their respective lists. These metrics provide insights into the model's performance and learning progress over each fold.

Collectively, this section navigates the dynamic landscape of model development, optimization,

and validation. Through preprocessing, compilation, and cross-validation orchestration, a profound understanding is cultivated. This paves the way for informed decision-making and an ongoing journey of model refinement.

4.3 Chapter Summary

In this chapter, the intricate details of the techniques incorporated in the project are delved into, elucidating the rationale behind their selection and utilization within the Proposed Algorithm. The focus here is not on performance evaluation, as this aspect is reserved for Chapter 5. Instead, each technique, such as word embedding, hidden layers, activation functions, optimizers, and evaluation functions, is meticulously dissected, explaining their roles in enhancing the BERT model's capabilities for multi-class text classification. By thoroughly elucidating the inner workings of these techniques, a robust foundation is aimed to be established for the subsequent discussions in upcoming chapters. The pivotal inclusion of the Cross-validation technique, a potent tool for model assessment, is also highlighted. Through its iterative nature, Cross-validation safeguards against overfitting and furnishes a reliable framework to evaluate the model's performance across diverse scenarios, thereby fortifying the credibility of the research outcomes. In addition to technique details, the chapter delves into the selection and preprocessing of two datasets, namely **20news-18828.tar.gz** and **20news-19997.tar.gz**.

Chapter 5

Result and Discussion

In this chapter, the results from experiments conducted to indicate the performance of a fully trained (pre-trained + fine-tuned) BERT model for multi-class text classification by implementing the cross-validation technique are presented. This chapter provides a detailed analysis of the results obtained and discusses their implications.

Here, the details of the experimental results are directly presented, highlighting key findings and the insights gained. The impact of different components of the proposed methodology, such as the use of Wordpiece word embeddings, BERT's attention layers, the dense layer, the softmax activation function, and the RAdam optimizer, is analyzed. The effect of hyperparameters, parameter tuning, and architecture itself on the performance of the model is investigated.

Starting with the pre-processed dataset (Dataset 1), the presentation of evaluation metrics, including accuracy, precision, recall, and F1 score, provide insights into the model's capability to correctly classify text into multiple classes. These metrics are calculated using the test dataset for the 20news-18828.tar.gz dataset. The performance of a fully trained BERT model is then examined. The confusion matrix in the following figure lists the proportions of true positives, true negatives, false positives, and false negatives for each class. This analysis helps spot any correct or incorrect behaviors that may exist during the classification process, as well as the model's advantages and disadvantages for various classes. The data in the figure below shows that the numbers that move along the diagonal are those that were correctly derived from the whole data. The chart also shows no issue of class imbalance, indicating that the model has been trained well and is in a stable mode.

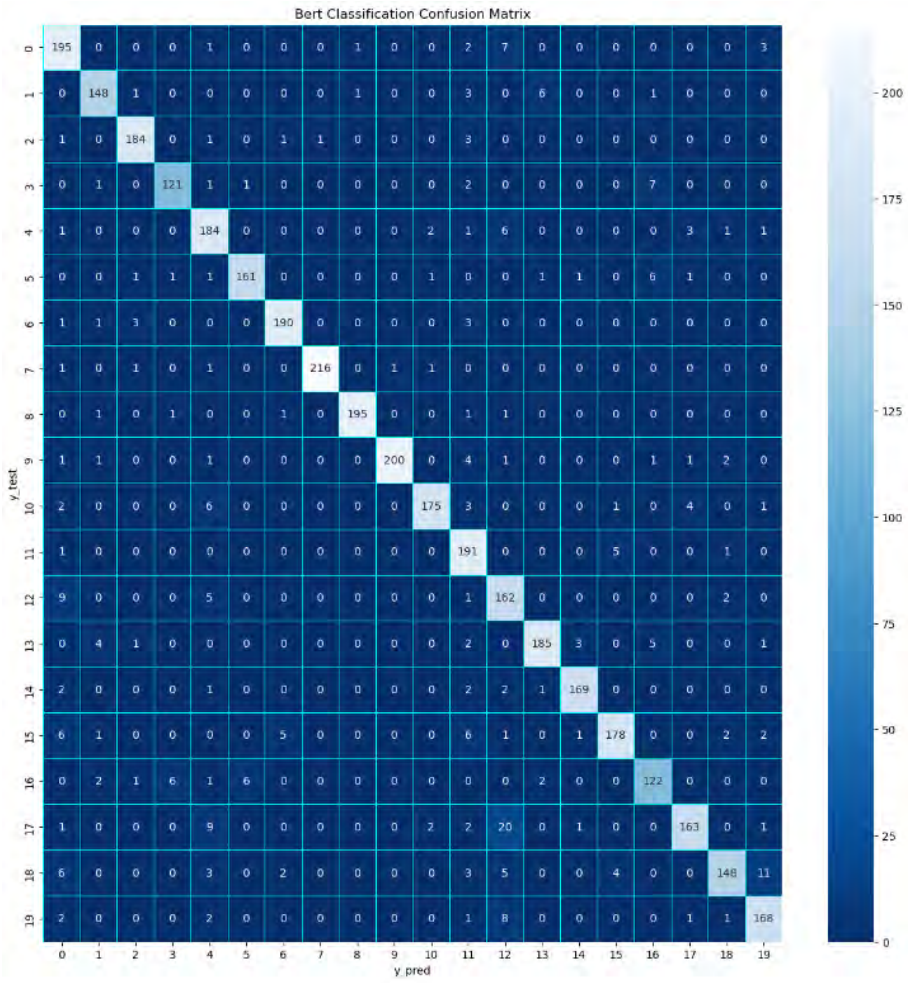

Figure 5.1: Confusion Matrix Table

Figure 5.2 provides a visual representation of the model weaknesses and strengths. The x-axis of the graph shows the epoch number, and the y-axis gives the accuracy and loss. A bar plot is generated using the Matplotlib library with the help of validation data. This plot displays the number of folds, allowing visualization of any overfitting exhibited by the model, which can impact its performance and provide insights for further analysis. The plot showed that the BERT pretrained model has trained well on the 20newsgroup dataset. The training and validation accuracy curves are fairly close to each other, and there is no indication that the model is starting to plateau or even decreasing the accuracy. This suggests that the model is able to generalize well to new unseen data as intended, and that it is not simply memorizing the training data. The image shows the training and validation accuracy of a model that was trained with cross-validation.

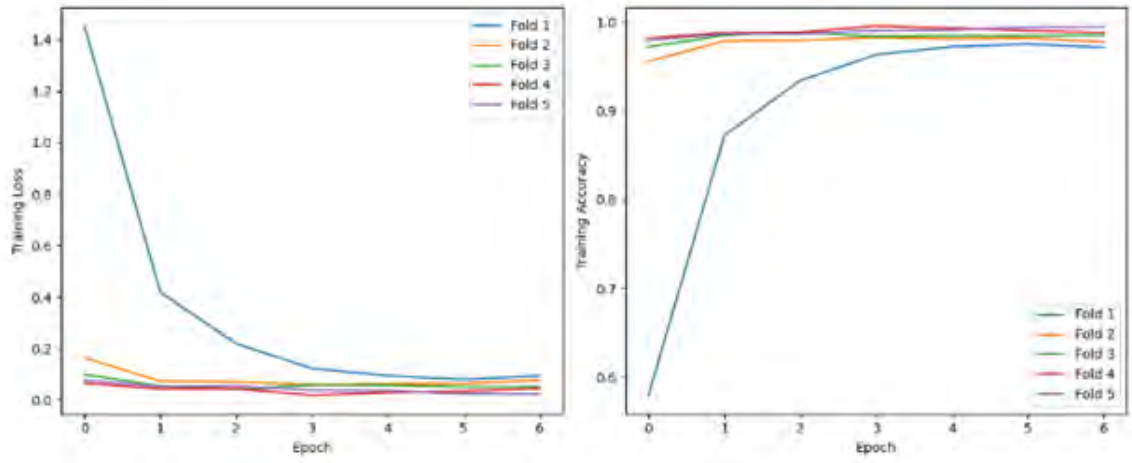

Figure 5.2: Model evaluation using 5-fold cross-validation.
This plot shows no sign of overfitting for dataset 1.

Figure 5.3 and Figure 5.4 provide a visual representation of the class distribution in the test and train data subsets. A bar plot is generated using the Pandas library. The x-axis represents the numbers of samples in each class, and the y-axis shows the numbers of classes. These plots display the number of instances in each class, allowing us to see any class imbalances or biases in the dataset, which can impact the performance of the model and provide insight for further analysis.

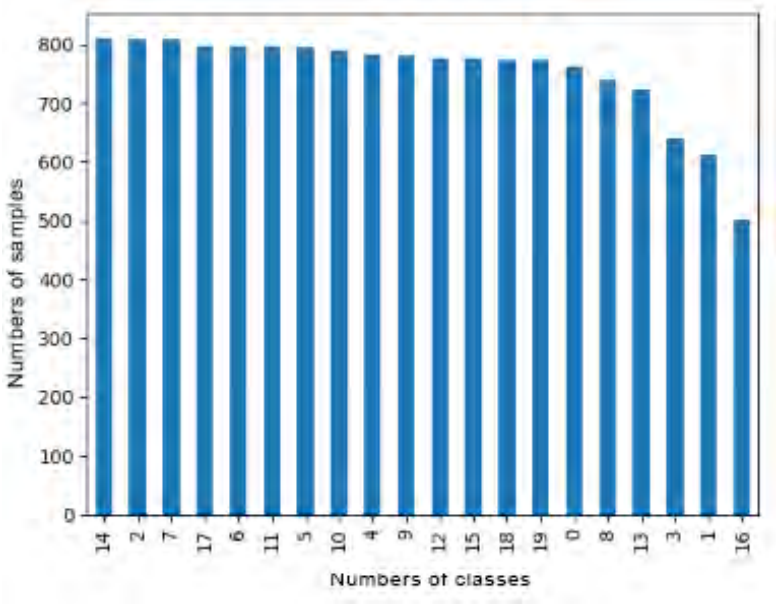

Figure 5.3: Train-y plot

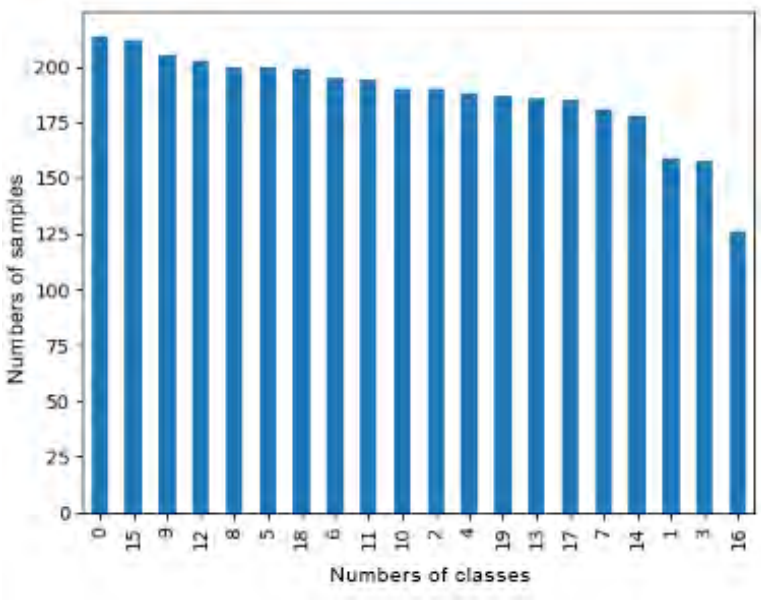

Figure 5.4: Test-y plot

The Figure 5.5 gives a table of metrics that show the performance statistics. The metrics in the table include precision, recall, f1-score, support, accuracy, macro avg, and weighted avg. Figure 5.5 also shows the metrics for each class in the dataset, as well as the overall metrics for the model. The overall metrics are calculated by averaging the metrics for each class. It also shows the number of instances in each class. This information can be used to calculate the class imbalance, which is the difference in the number of instances in different classes. Overall, the image shows the performance of the BERT model on a multi-class classification task. The metrics in the table can be used to evaluate performance and to identify any potential problems.

5.1 Dataset 2

Dataset 2 represent **20news-19997.tar.gz**, which was also used on the model for performance evaluation. Starting with the presentation of the confusion matrix, followed by plotting to identify class imbalance and over-fitting issues, and finally, evaluation metrics including accuracy, precision, recall, and F1 score.

	precision	recall	f1-score	support
0	0.85	0.93	0.89	209
1	0.93	0.93	0.93	160
2	0.96	0.96	0.96	191
3	0.94	0.91	0.92	133
4	0.85	0.92	0.88	199
5	0.96	0.93	0.94	174
6	0.95	0.96	0.96	198
7	1.00	0.98	0.99	221
8	0.99	0.97	0.98	200
9	1.00	0.94	0.97	212
10	0.97	0.91	0.94	192
11	0.83	0.96	0.89	198
12	0.76	0.91	0.83	179
13	0.95	0.92	0.93	201
14	0.97	0.95	0.96	177
15	0.95	0.88	0.91	202
16	0.86	0.87	0.87	140
17	0.94	0.82	0.88	199
18	0.94	0.81	0.87	182
19	0.89	0.92	0.91	183
accuracy			0.92	3750
macro avg	0.92	0.92	0.92	3750
weighted avg	0.93	0.92	0.92	3750

Figure 5.5: Classification Report

Fig 5.7 provides a visual representation of the model weaknesses (overfitting) and strengths. The x-axis of the graph shows the epoch number, and the y-axis shows the accuracy and loss. A bar plot is generated using the Matplotlib library with the help of validation data. The plot shows that the BERT pre-trained model was trained well on the 20news-19997.tar.gz dataset. The training and validation accuracy curves are fairly close to each other, and there is no indication that the model is starting to plateau or even decreasing its accuracy. This suggests that it is not simply memorizing the training data.

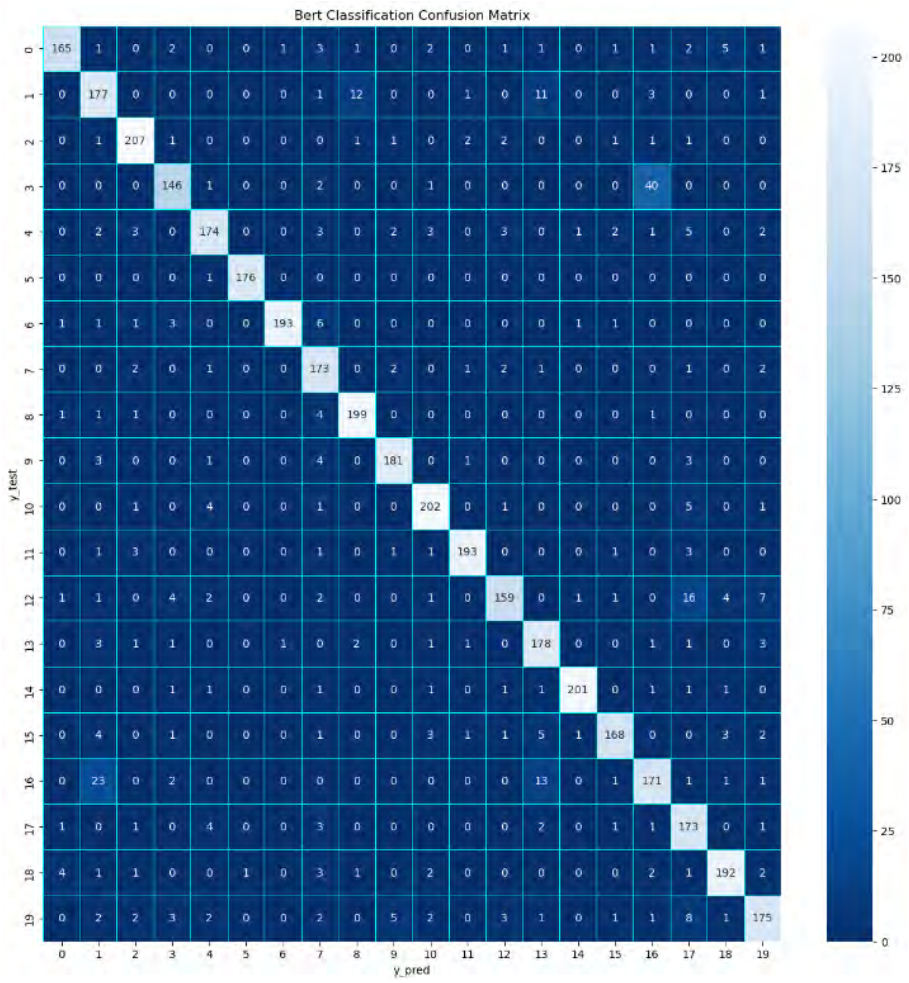

Figure 5.6: Confusion Matrix Table

Figure 5.8 also present the overall metrics for the model on dataset 2. The overall metrics are calculated by averaging the metrics for each class. It also shows the number of instances in each class. This information can be used to see class imbalance, which is the difference in the number of instances in different classes. Overall, the image shows the performance of BERT model on the multi-class classification task. The metrics in the table can be used to evaluate performance and to identify any potential problems with the model.

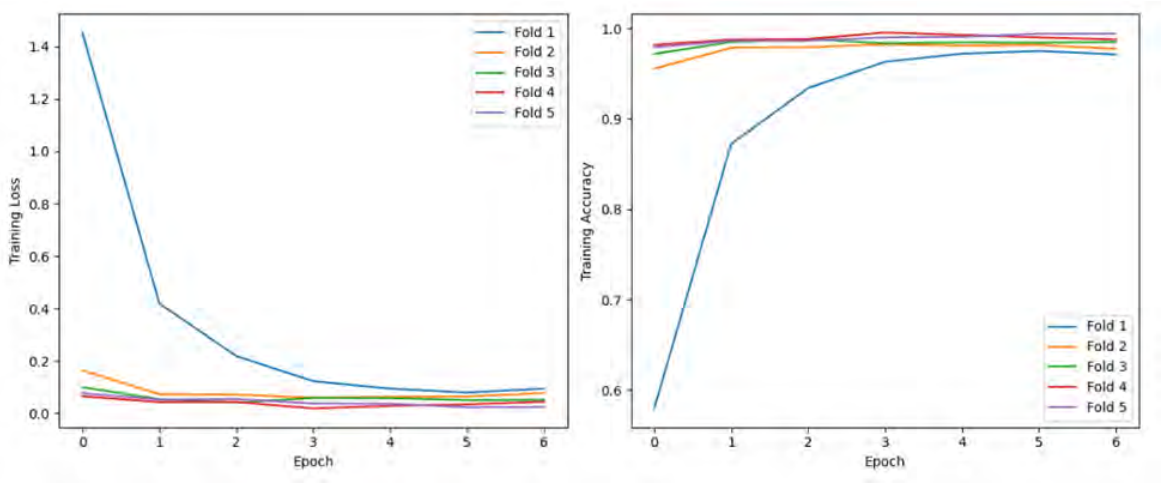

Figure 5.7: Model evaluation using 5 fold cross-validation.
This plot shows no sign of overfitting for dataset 2.

	precision	recall	f1-score	support
0	0.95	0.88	0.92	187
1	0.80	0.86	0.83	206
2	0.93	0.95	0.94	218
3	0.89	0.77	0.82	190
4	0.91	0.87	0.89	201
5	0.99	0.99	0.99	177
6	0.99	0.93	0.96	207
7	0.82	0.94	0.88	185
8	0.92	0.96	0.94	207
9	0.94	0.94	0.94	193
10	0.92	0.94	0.93	215
11	0.96	0.95	0.96	204
12	0.92	0.80	0.85	199
13	0.84	0.92	0.88	193
14	0.98	0.96	0.97	210
15	0.94	0.88	0.91	190
16	0.76	0.80	0.78	213
17	0.78	0.93	0.85	187
18	0.93	0.91	0.92	210
19	0.88	0.84	0.86	208
accuracy			0.90	4000
macro avg	0.90	0.90	0.90	4000
weighted avg	0.90	0.90	0.90	4000

Figure 5.8: Classification Report

Figure 5.9 shows different text classification models compared each other. Each model is described by a number of features, including the size of the dataset it was trained on, the type of models used, the architecture of the models, whether the models were pre-trained, the accuracy of the models, the disadvantages of the models, the performance of the models, and whether the models are good for text classification tasks and lastly Classification tab. The classification tab include Binary classification which means models are only able to predict two classes, while multi-class classification models can predict more than two classes.

5.2 Comparison

In this chart, the performance of different text classification models is investigated. The accuracy, disadvantages, performance, and suitability for text classification tasks are also compared using the Pre-trained BERT model with different other models [8] [1].

Model	Size of dataset	Models	Architecture	Pre-Trained	Accuracy	Disadvantage	Performance	TC task	Classification
NDM, SVM, NB	20	3	Simple	No	71.10%	Not as accurate as other models	Poor	No	binary
MMR, SVM, NB	20	3	Simple	No	84.00%	Not as accurate as other models with fine-tuning	Fair	No	binary
Modified CNN	4 classes only	2	Complex	No	96.49%	Requires large dataset to achieve high accuracy	Good	No	multi
Auto encoder	20	2	Complex	No	73.78%	Not as accurate as other models	Poor	No	binary
DBN+Softmax	20	2	Complex	No	85.57%	Not as accurate as other models with fine-tuning	Fair	No	binary
Deep learning and meta-thesaurus	20	2	Complex	No	69.82%	Not as accurate as other models	Poor	No	binary
LSTM	20	2	Complex	No	86.00%	Requires large dataset to achieve high accuracy	Good	Yes	binary
SVM+NB	9 classes only	2	Complex	No	82.20%	Not as accurate as other models with fine-tuning	Fair	No	multi
CNN+LSTM	topic categorize	2	Complex	No	90.00%	Requires large dataset to achieve high accuracy	Good	Yes	multi
ML classifier comparison	20	4	Complex	No	86.00%	Not as accurate as other models with fine-tuning	Fair	No	multi
Feature representation, ML classifiers	20	2	Complex	No	68.00%	Not as accurate as other models	Poor	No	binary
Cross domain Transfer learning	6 classes only	3	Complex	Yes	95.62%	Requires large dataset to train the initial model	Good	Yes	multi
Multi Channel CNN	20	2	Complex	Yes	82.76%	Requires large dataset to get better accuracy	Fair	No	multi
Feature Engineering, Multi Channel CNN	20	2	Complex	Yes	91.72%	Requires large dataset to get better accuracy	Good	No	multi
Pre-trained BERT	**20**	**1**	**Deep bidirectional transformer**	**Yes**	**92.13%**	**Already trained with large dataset, but can give better accuracy with fine-tunning specifics dataset**	**Excellent**	**Yes**	**multi**

Figure 5.9: Comparison of Pre-Trained BERT model with state-of-the-art machine and deep learning methodologies on 20 newsgroup dataset in terms of Advantages and Disadvantages

This comparison helps us assess the effectiveness and superiority of the proposed methodology in achieving higher accuracy and better performance in multi-class text classification tasks.

Figure 5.10: Visualization of research results comparing the performance of BERT and traditional machine learning models

Dataset	Accuracy
20news-18828	0.92293
20news-19997	0.90075

Table 5.1: Results of Model

The model showed some limitations or constraints during the experiments, such as computational resources, data availability, or potential biases in the dataset. The potential impact of these limitations on the model's performance is addressed, and suggestions for future research to overcome these constraints are provided. Overall, this chapter provides a comprehensive and detailed assessment of the results obtained from the fully trained BERT model for multi-class text classification. Through this analysis and discussion, the aim is to gain a deeper understanding of the model's performance, uncover insights, and draw meaningful conclusions for further improvements and advancements in the field.

5.3 Chapter Summary

This final chapter summarizes everything covered so far. Findings and comments, as well as the outcomes of testing the classifier using cross-validation techniques, are discussed. Specifics of how the model performed well on two separate datasets are examined. An evaluation of how the model handled each dataset's unique challenges and strengths is provided. This helps determine where the model thrives and where it can struggle. One interesting finding is that the first dataset, which doesn't have any distracting noise, led to better accuracy compared to the second dataset that had some noise in it.

This highlights the significance of clean data for our model's success. Another important aspect is how the model reacts to noise in the data. It is observed that it performs better when the data is devoid of noise, implying that our model is sensitive to the quality of the information provided. In summary, this chapter wraps up all the insights gained, reminding us of the model's strengths and limitations and how different types of data can impact its performance. It sets the stage for future exploration in the area of text classification techniques.

Chapter 6

Conclusion and Future work

6.1 Conclusion

The research's main contribution is to enhance the performance of BERT model for text classification task using transfer learning technique. In this regard, BERT has been fine-tunned on 20newsgroup dataset by incorporating various components to get optimum results. The performance measures achieved by BERT (on both datasets) showed remarkable accuracy of 92% for preprocesed and 90% for raw dataset of 20newsgroup dataset. Comparison analysis of the proposed model with all baseline models reveals that the former has outperformed among all. The reason behind this performance is the selection of the cross validation technique for the training of the model. Cross-validation makes the performance of the model robust by avoiding overfitting. The bi-directional transformer architecture of pre-trained BERT has made it suitable for multi-class text classification task.

6.2 Future Work

Better accuracies in text classification can be pursued through various approaches. During fine-tunning, adjustment of parameters of the model can optimize the results. For enhancing overall performance of predictions, some powerful tools like ensemble learning and stacking can be employed. Harnessing the collective intelligence of multiples models can capitalize on their diverse capabilities, leading to more accurate and reliable classifications. Synthetic data can be generated from existing samples by using advance data augmentation strategies. The expansion of dataset, makes the model more effecient and fit for real-world scenerios.

Bibliography

[1] Muhammad Nabeel Asim, Muhammad Usman Ghani Khan, Muhammad Imran Malik, Andreas Dengel, and Sheraz Ahmed. A robust hybrid approach for textual document classification. In *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pages 1390–1396, 2019.

[2] Babajide O. Ayinde, Tamer Inanc, and Jacek M. Zurada. On correlation of features extracted by deep neural networks. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, July 2019.

[3] Alexei Baevski and Michael Auli. Adaptive input representations for neural language modeling, 2019.

[4] Qifang Bi, Katherine E Goodman, Joshua Kaminsky, and Justin Lessler. What is machine learning? a primer for the epidemiologist. *American journal of epidemiology*, 188(12):2222–2239, 2019.

[5] Jose Camacho-Collados and Mohammad Taher Pilehvar. On the role of text preprocessing in neural network architectures: An evaluation study on text categorization and sentiment analysis. *arXiv preprint arXiv:1707.01780*, 2017.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[7] Fatma Elghannam. Text representation and classification based on bi-gram alphabet. *Journal of King Saud University-Computer and Information Sciences*, 33(2):235–242, 2021.

[8] Santiago González-Carvajal and Eduardo C Garrido-Merchán. Comparing bert against traditional machine learning text classification. *arXiv preprint arXiv:2005.13012*, 2020.

[9] Thenmolli Gunasegaran and Yu-N Cheah. Evolutionary cross validation. 2017.

[10] Zied Haj-Yahia, Adrien Sieg, and Léa A Deleris. Towards unsupervised text classification leveraging experts and word embeddings. In *Proceedings of the 57th annual meeting of the Association for Computational Linguistics*, pages 371–379, 2019.

[11] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. Deep neural networks classification over encrypted data. In *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy*, pages 97–108, 2019.

[12] Lijie Huang. Evaluation of query strategy of active learning for text classification. In *2nd International Conference on Artificial Intelligence, Automation, and High-Performance Computing (AIAHPC 2022)*, volume 12348, pages 532–540. SPIE, 2022.

[13] Ammar Ismael Kadhim. Survey on supervised machine learning techniques for automatic text classification. *Artificial Intelligence Review*, 52(1):273–292, 2019.

[14] Kamran Kowsari, Kiana Jafari Meimandi, Mojtaba Heidarysafa, Sanjana Mendu, Laura Barnes, and Donald Brown. Text classification algorithms: A survey. *Information*, 10(4):150, 2019.

[15] Songsong Liu, Haijun Tao, and Shiling Feng. Text classification research based on bert model and bayesian network. In *2019 Chinese Automation Congress (CAC)*, pages 5842–5846. IEEE, 2019.

[16] Dipti Mahajan and Dev Kumar Chaudhary. Sentiment analysis using rnn and google translator. In *2018 8th international conference on cloud computing, data science & engineering (Confluence)*, pages 798–802. IEEE, 2018.

[17] Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. Advances in pre-training distributed word representations. *arXiv preprint arXiv:1712.09405*, 2017.

[18] Zun Hlaing Moe, Thida San, Mie Mie Khin, and Hlaing May Tin. Comparison of naive bayes and support vector machine classifiers on document classification. In *2018 IEEE 7th global conference on consumer electronics (GCCE)*, pages 466–467. IEEE, 2018.

[19] Daniel W Otter, Julian R Medina, and Jugal K Kalita. A survey of the usages of deep learning for natural language processing. *IEEE transactions on neural networks and learning systems*, 32(2):604–624, 2020.

[20] Zhenhui Peng and Xiaojuan . A survey on construction and enhancement methods in service chatbots design. *CCF Transactions on Pervasive Computing and Interaction*, 1(3), 2019.

[21] Rukhma Qasim, Waqas Haider Bangyal, Mohammed A Alqarni, Abdulwahab Ali Almazroi, et al. A fine-tuned bert-based transfer learning approach for text classification. *Journal of healthcare engineering*, 2022, 2022.

[22] Afia Sajeeda and BM Mainul Hossain. Exploring generative adversarial networks and adversarial training. *International Journal of Cognitive Computing in Engineering*, 3:78–89, 2022.

[23] Sasan Sharifipour, Hossein Fayyazi, Mohammad Sabokrou, and Ehsan Adeli. Unsupervised feature ranking and selection based on autoencoders. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3172–3176. IEEE, 2019.

[24] C M Suneera and Jay Prakash. Performance analysis of machine learning and deep learning models for text classification. In *2020 IEEE 17th India Council International Conference (INDICON)*, pages 1–6, 2020.

[25] Ian Tenney, Dipanjan Das, and Ellie Pavlick. Bert rediscovers the classical nlp pipeline. *arXiv preprint arXiv:1905.05950*, 2019.

[26] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[27] Jesse Vig and Yonatan Belinkov. Analyzing the structure of attention in a transformer language model, 2019.

[28] Hongping Wu, Yuling Liu, and Jingwen Wang. Review of text classification methods on deep learning. *Computers, Materials & Continua*, 63(3), 2020.

[29] Weidi Xu, Haoze Sun, Chao Deng, and Ying Tan. Variational autoencoder for semi-supervised text classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[30] Shiwei Zhang, Xiuzhen Zhang, and Jeffrey Chan. A word-character convolutional neural network for language-agnostic twitter sentiment analysis. In *Proceedings of the 22nd Australasian Document Computing Symposium*, pages 1–4, 2017.

[31] Jiakun Zhao, Ju Jin, Si Chen, Ruifeng Zhang, Bilin Yu, and Qingfang Liu. A weighted hybrid ensemble method for classifying imbalanced data. *Knowledge-Based Systems*, 203:106087, 2020.