POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Detection and extraction of adverse drug reaction in social media using deep learning

ZAHRA SEDAGHAT
Département de génie électrique

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie électrique

Décembre 2021

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

Detection and extraction of adverse drug reaction in social media using deep learning

présenté par **Zahra SEDAGHAT**
en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

DEDICATION

To Mom and Dad

ACKNOWLEDGEMENTS

I would like to thank the people involved in this project for their continuous support and guidance. First and foremost, I would like to express my gratitude to my supervisor Prof. Yvon Savaria and my co-supervisor, Prof. Chahé Nerguizian, for the opportunity they gave me and for their continuous support, guidance, useful feedback as well as for being so patient with me with multiple reviews of the thesis and the learning process. Thanks to my committee members Prof. Michel Desmarais and Prof. François Leduc-Primeau for their help reviewing my thesis and providing important feedback. I would also like to thank the Medvalgo company for providing me with the information and resources to complete my thesis. Last but not least, thanks to my family, who have endured this journey with me with their constant support and love.

RÉSUMÉ

Un effet indésirable médicamenteux (EIM) peut être défini comme une réaction sensiblement nocive ou désagréable résultant d'une intervention liée à l'utilisation d'un médicament. D'après les services de santé, les EIM sont à l'origine de nombreuses hospitalisations et, dans certains cas graves, du décès des patients chaque année. En plus d'avoir un impact significatif sur la santé, les EIM ont également des effets économiques néfastes pour les personnes affectées. En raison de tous ces problèmes, la recherche en pharmacovigilance s'est popularisée pour détecter et extraire avec diligence et précision les effets secondaires des médicaments. La pharmacovigilance concerne les activités de surveillance de l'innocuité du médicament dans la phase post-commercialisation du médicament. Les sites de réseautage social comme Twitter ont fourni aux gens une plate-forme pour se connecter les uns aux autres, pour discuter et partager des informations et des nouvelles. L'identification et la collecte de tweets contenant des mentions d'expérience personnelle en matière de santé seraient l'une des sources de données les plus rapides et les plus importantes pour les chercheurs. La détection et l'extraction des mentions de médicaments indésirables dans les tweets peuvent compléter la liste des effets indésirables des médicaments résultant des essais de médicaments et peuvent aider à l'amélioration des médicaments. La détection et l'extraction d'EIM à partir de texte peuvent être considérées comme un problème d'extraction d'informations qui a connu une croissance significative ces dernières années en raison des progrès de l'apprentissage en profondeur et de l'apprentissage par transfert. Ce mémoire vise à étudier les performances de trois modèles différents de BiLSTM (Bidirectionnel Long-Short Term Memory), BERT (Bidirectionnel Encoder Representations from Transformers) et BERT+BiLSTM dans les tâches de détection et d'extraction EIM. Pour former et évaluer ces modèles, un ensemble de données de 2735 tweets qui sont étiquetés manuellement par certains étudiants en pharmacologie est utilisé. Dans l'étape suivante, pour vérifier la possibilité d'obtenir de meilleurs résultats dans la tâche de détection, la tentative consiste à agrandir l'ensemble de données en utilisant certaines techniques d'augmentation proposées avec Easy Data Augmentation (EDA) et back-translation et à former le meilleur modèle par l'ensemble de données augmenté.

ABSTRACT

Adverse drug reaction (ADR) is an unwanted, uncomfortable, or dangerous effect that occurs after prescribing the drug and directly correlates with the drug. ADR occurs during 10 to 20% of hospitalizations, some of which are severe and lead to death [1]. In addition, it is also a substantial financial drain on the bottom line for taxpayers. However, early and accurate detection of ADRs will mitigate and prevent the number of ADR related potential issues. In addition, for detrimental reactions in patients caused by taking drugs, social media conversations may report ADRs more quickly and efficiently than other means. By knowing that, there is a clear need for automated ADR surveillance at the post-marketing phase of drugs. This study aims to use deep learning methods to accurately identify unreported drug side effects in reviews of twitter users. Utilizing a dataset of 2735 tweets manually labeled by some pharmacology students, this research conducts deep learning based and pre-trained transfer learning based approaches as a solution for detecting and extracting ADRs automatically. As a first deep learning approach, we apply Bidirectional Long-Short Term Memory Networks (BiLSTM), units that recurrently compute nonlinear transformations on the dataset. In the next step, we utilize the Bidirectional Encoder Representations from Transformers (BERT) model and different combinations of its encoder layers for ADR extraction. The study's final model is a combination of the best model from the previous BERT experiments, a BiLSTM, and a classifier layer. The experiments have shown that the hybrid BERT+BiLSTM model outperforms all others. In the final step, the efficiency of some proposed data augmentation techniques along with Easy Data Augmentation (EDA) and back-translation for ADR detection tasks is explored. Ultimately, the experimental results show that using a different combination of these simple augmentation strategies to enlarge the training dataset boosts the model performance.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

ADR	Adverse Drug Reaction
AI	Artificial Intelligence
API	Application Programming Interface
BERT	Bidirectional Encoder Representations from Transformers
BiLSTM	Bidirectional Long Short-Term Memory
CHV	Consumer Health Vocabulary
CNN	Convolutional Neural Network
CNNA	Convolutional Recurrent Neural Network
CRF	Conditional Random Fiels
EHR	Electronic Health Records
FAERS	FDA Adverse Event Reporting System
FDA	Food and Drug Administration
KNN	K-nearest Neighbor
LR	Logistic Regression
LSTM	Long Short-Term Memory
MedDRA	Medical Dictionary for Regulatory Activities
MUC	essage Understanding Conference
NB	Naïve Bayes
NER	Named Entity Recognition
NLP	Natural Language Processing
PIDM	Program for International Drug Monitoring
PV	Pharmacovigilance
RF	Random forest
RNN	Recurrent Neural Network
SIDER	Side Effect Resource
SRS	Spontaneous Reporting System
SVM	Support Vector Machine
UMLS	Unified Medical Language System
WHO	World Organization

LIST OF APPENDICES

CHAPTER 1 INTRODUCTION

1.1 Motivation

After administering a drug at standard doses, any unintended side effects are categorized as adverse drug reactions (ADRs) by the World Health Organization (WHO) definition. Ideally, ADRs are detected during the randomized control trial phases. However, due to limitations such as the number of participants and the short duration of the study, they are not always detected before a drug is put on the market. These reactions or side effects can vary from minor problems like a simple rash, allergies, dry mouth, and headache to some severe life-threatening issues, such as the risk of a heart attack, internal bleeding, and cancer. In Canada, an estimated 200,000 severe adverse drug reactions kill up to 22,000 Canadians per year. Moreover, it is estimated that 95% of ADRs are not reported, costing the Canadian healthcare system between 13.7 and 17.7 billion dollars [9]. In the United States, statistics show a higher number. ADRs are estimated to represent over 3.5 million physician visits, over 1 million trips to the emergency department, and more than 2 million injuries, hospitalizations, and deaths [10]. Apart from a significant impact on health, ADRs incur significant economic losses, totaling over $75 billion each year [11].

In order to detect ADRs early, the World Health Organization set up the Program for International Drug Monitoring (PIDM) which is referred to as pharmacovigilance. As defined by the WHO, Pharmacovigilance (PV), also known as drug safety, is a scientific approach to collecting, detecting, assessing, monitoring, and preventing adverse effects or any other drug product problem. Two different approaches will create this information. The first approach is a spontaneous reporting system (SRS), consisting of the submitted report from manufacturers and healthcare professionals. The second approach gathers data during routine clinical visits such as Electronic Health Records (EHR), hospital discharge summaries, and medical prescriptions.

Accurate and early detection and extraction of ADRs and appropriate regulatory action can prevent many deaths and hospitalizations and reduce related financial costs. It can also help the pharmaceutical industries improve drug safety and prevent huge financial losses since post-market ADRs may reduce their drug production. All these reasons motivat studying ADR detection and extraction.

1.2 Problem definition

Despite the importance of detecting ADRs, they are unfortunately vastly under-reported [12]. This is because only health professionals could report ADRs at first. In the following years, however, many studies demonstrated the importance of patients as reporters, and drug user reporting became a complementary source of knowledge [13].

One of the problems in this field is that most of the healthcare providers report severe ADRs only, and also the voluntary submission by the consumers limits the reporting quantity [14], giving rise to a median under-reporting of ADRs for about 94% [15] which is a worrying amount. To compensate for the lack of clinical reports of ADRs, some researchers decided to use social media networks to extract a rich source of information [16]. The total number of people who use social media grows worldwide continually, resulting in vast amounts of data. Twitter is one of the most popular platforms for academic research. According to estimates, 500 million tweets are posted every day, and of the total users, 26% have discussed health information in their tweets [17]. Statistics show that there is a considerable correlation between ADRs detected in Twitter and those reported in the FDA Adverse Event Reporting System (FAERS), which offers the reliability of Twitter as a rich source of ADRs detection in addition to the clinical reports [18]. Therefore, Twitter is chosen as the source used in this study to compensate for the limitation.

Detection and extraction of drug side effects are expertise-dependent tasks, and if they are done only in a manual way, it will be expensive and time-consuming, which is another issue of this field. However, thanks to the significant improvement in AI and machine learning, especially deep learning methods [19], working with text to extract information that is one of the subtasks in natural language processing (NLP), becomes more convenient and practical compared to the approaches employed in the last few years such as Rule-based extraction methods. Recently, the NLP community has been witnessing the effectiveness of pre-trained models and paying more attention to this sort of deep language representation models such as ELMo [20], GPT [21], and especially Bidirectional Encoder Representations from Transformers (BERT), which achieves state of the art in most of the NLP tasks [22] in contextual representation. Therefore, deep learning and pre-trained approaches will be used to detect and extract ADR automatically in this study.

1.3 Objective

This thesis aims to address two challenges: automatic detection of ADR and automatic extraction of ADR in social media, specifically on Twitter. The focus of the ADR detection task is on identifying adverse drug reactions as a medication-related outcome. The detection model as a sentence-level task is expected to differentiate tweets as reporting adverse drug reactions or not. As a word-level task, the ADR extraction model focuses on identifying the location of ADR mentions in each tweet. The core approach for this task is, generally, a Named Entity Recognition (NER) system. The NER task identifies objects (one or many words) like adverse drug reactions that belong to some predefined categories.

For implementing the research tasks, different models based on recurrent neural network Bidirectional Long Short-Term Memory (BiLSTM) [23], BERT, and some combination of BERT and BiLSTM is explored to find the model with the best performance in the dataset of the study. Moreover, to gain high accuracy in the ADR detection task, some simple data augmentation techniques are proposed and examined to check their effectiveness for pre-trained models and small size domain-specific datasets.

1.4 Outline

The rest of the thesis is organized as follows: in Chapter 2, the related works that have been done in this field and the theoretical background behind the concepts and models used in this study are discussed. Chapter 3 describes the dataset preparation steps, different methods used, experiments set-ups, all the results, and discussions of implementing the ADR extraction tasks. In Chapter 4 some new data augmentation techniques are proposed and examined to improve the ADR detection performance. Finally, Chapter 5 presents the conclusions of this work and propose further options of the study.

CHAPTER 2 PHARMACOVIGILANCE AND RELATED EXISTING NATURAL LANGUAGE PROCESSING METHOD

2.1 Pharmacovigilance methods

The detection of adverse drug reactions (ADRs) has received plenty of attention, and many studies have been done in this area. Published work in this field can be divided into three primary classes: lexicon-based approaches, traditional machine learning approaches, and deep learning approaches.

2.1.1 Lexicon-based approach

The lexicon-based approaches rely on analyzing language based on the idea that it comprises lexical units rather than grammatical structures. For the detection of ADRs, a lexicon-based approach primarily uses large data sets of medical and drug-related terms called lexicons to identify if the tokenized words in the input text are matched with the ADR entries in the lexicon.

There exist some standard data sets or, in other words, dictionaries among all lexicon-based approaches that include sources such as the Side Effect Resource [24] (SIDER, that connects 888 drugs to 1450 side effect terms), Consumer Health Vocabulary [25] (CHV, the open-access terminology system contains consumer alternatives for medical terms), Medical Dictionary for Regulatory Activities (MedDRA) [26], Unified Medical Language System (UMLS, Unified Medical Language System, which is commonly used in a medical field to manage a variety of standards and resources for biomedical information (e.g., health records) to facilitate collaboration [27], the FDA Adverse Event Reports System (FAERS, which is used to track information pertaining to the quality of pharmaceutical products, including reports of negative side effects, errors, etc.).

Leaman and Wojtulewicz [28] implemented one of the first studies to extract ADRs from social networks. They created a dataset by gathering the user comments from DailyStrength, a social network for sharing health-related experiments. Their technique to extract the ADRs was to tokenize the comments and check their matches by terms in lexicons created from UMLS Methathesaurus2, SIDER, and Canada Drug Adverse Reaction Database [29]. Their results showed that comments from health-related social networks could reflect known ADRs.

Benton et al. [30] collected a corpus of medical message board posts related to breast cancer and attempted to extract ADRs by finding the co-occurring terms in lexicons and corpus.

Yates and Goharian [31] collected their data based on the names of five commonly used breast cancer drugs from three drug review social media sites, namely, askapatient.com, drugs.com, and drugratingz.com. Using various lexicons such as Side Effect Resource and the Unified Medical Language System, which includes a collection of glossaries of medical terms and presents them in a graph-like data structure indicating how the terms are related. Their method process creates equivalent terms for ADRs and extracts all the expected and unexpected ADRs from the reviews.

Liu and Chen [32] developed an automated crawler to download web pages from patient forums to collect their data. Then, they applied multiple types of lexicon sources to extract drug names and adverse events from the text, including UMLS, FAERS, and CHV. They initialized the medical entity extraction with MetaMap, a highly configurable Java API from the National Library of Medicine. It was used to map patient social media text to the UMLS to recognize terms matching standard medical lexicons in patient forums.

Lexicon-based approaches tend to be fast from a processing time standpoint since they do not require training on their data; however, some limitations exist. For example, users frequently rely on everyday informal language to describe the symptoms they are experiencing instead of recognized medical terms. For example, the phrase “messed up my sleeping patterns” was used to report “sleep disturbances” or use phrases with misspellings and abbreviations. Furthermore, approaches that rely on a fixed lexicon are challenging to use because they require that the lexicon be periodically updated. Thus, the limitations of a lexicon approach call for improvements to solve these problems.

2.1.2 Traditional machine learning approach

The availability of annotated data for ADR identification tasks in recent times makes it possible to use machine learning techniques to detect ADRs automatically. Advanced statistics and machine learning methods make the users’ analysis more effective and accurate for descriptive and predictive purposes. In statistical and machine learning approaches, features are carefully designed using word-level features, documents, and corpus features for representing the training samples. These algorithms are then utilized to learn a model to find out similar patterns from unseen data.

Nikfarjam and Gonzalez [16] introduced an automatic machine learning-based concept extraction system called ADRMine that uses conditional random fields (CRFs), a statistical-based method, applied to pattern recognition by taking the context into account. They used the ADRMine to identify specific social media posts mentioning adverse reactions to drugs. In addition, they used a new feature that models the semantic of word similarities by applying a clustering method based on pre-trained word representation vectors to enhance their results.

Gurulingappa et al. [33] worked on adapting a machine learning-based system to identify and extract potential adverse drug event relations. They begin with a statistical approach by adopting an ontological framework and coupling the results with a machine learning model. Next, they employed the Java Simple Relation Extraction (JSRE) system to create a supervised classification platform that used Support Vector Machines (SVMs) with different kernels. The data set used for training and validation of the model was made from MEDLINE case reports, a high-quality corpus that includes bibliographic information for academic journals. Finally, they manually annotated this created dataset using an ontology-driven methodology.

I. S. Alimova and E. V. Tutubalina [34] developed an adverse drug reaction classifier based on user reviews, comparing the efficiency of a message-level approach to an entity-level one. For message-level classification, they applied their model to a dataset of tweets named the Twitter corpus. They used a dataset of user reviews called the CSIRO Adverse Drug Event Corpus (CADEC) for entity-level. Logistic Regression and Linear Support Vector Machine (SVM) classifiers are two machine learning algorithms they utilize in their proposed model.

For the ADR prediction task, Liu M, Wu Y, Chen Y, et al. [35] proposed a new drug surveillance framework based on a machine learning approach. First, they created different feature combinations by integrating chemical (i.e., compound signatures), biological (i.e., protein targets, transporters, enzymes, and pathways), and phenotypic (i.e., indications and other known side effects) properties. Then they developed five different machine learning algorithms, logistic regression (LR), naïve Bayes (NB), K-nearest neighbor (KNN), random forest (RF), and SVM for ADR prediction task, and SVM was found to outperform the others.

Using traditional machine learning methods leads to a substantial improvement, but there exist many limitations. For example, some of the approaches used were lexicon-based, and they had significant limitations; in particular, while they can be used to identify named entities or relationships between different parts of speech, they ignore semantics and deeper language structure. Furthermore, in traditional machine learning problems, features are created manually, which is time-consuming. Moreover, a domain expert must identify most of the underlying features applied to the data to work effectively for the machine learning algorithms.

2.1.3 Deep Learning approach

Deep Learning techniques are more effective due to the low concern about feature engineering since they attempt to learn high-level features from data rather than engineering them from scratch. This is one feature of Deep Learning that makes it more effective and accurate than traditional Machine Learning.

One of the first studies on deep learning for ADR detection is Cocos et al. research [36]. A bidirectional Long Short-Term Memory (BiLSTM) was used to train their model on a sample Twitter dataset and showed promising results compared to previous methods. Lee et al. [37] developed various semi-supervised Convolutional Neural Network (CNN) models to classify ADRs in tweets; they also used unlabeled data to develop their models. To solve the ADR detection problem, Huynh et al. [38] used CNN as a baseline. They proposed two new neural network models, a Convolutional Recurrent Neural Network (CRNN) that starts with a convolutional layer like the CNN but is followed by a recurrent layer and Convolutional Neural Network with Attention (CNNA) that is created by adding attention weights to convolutional neural networks. As a result, CNN performs better than other more complex CNN variants on the ADE dataset. In another research for ADR detection from electronic health records (EHRs) of patients in hospitals, Wunnava et al. [39] presented a three-layer deep learning architecture consisting of a recurrent neural network BiLSTM to represent character-level terms in medical terminology, another BiLSTM to capture the context of each word and conditional random fields (CRF) as the final label prediction method. Using BiLSTM, Li et al. [40] generated a joint model for extracting drug, disease, and adverse event mentions simultaneously.

From the different architectures used to implement neural networks, we can distinguish mainly RNNs, more specific LSTM, or Bi-LSTM. Considering the wide use of the LSTM model and its promising results, we select it as our baseline model.

Recently, with the advent of Transformer based approaches [5] such as BERT [7], and GPT [41] language models, which achieved performance improvement over the state-of-the-art RNN based architectures on several NLP tasks, researchers began to use these more advanced models for ADR detection tasks. For example, Chen et al. [42], for automatic classification and extraction of adverse effect mentions in tweets, using a BERT-based model, and their system ranks first among all systems on SMM4H Shared Task 2019. Social Media Mining for Health Applications (SMM4H) provided some NLP challenges on social media mining for health monitoring and surveillance. Another research for a pharmacovigilance task [43] used BERT and two other BERT domain-specific implementations: BioBERT, which is a BERT model that pre-trained on biomedical corpora, including PubMed abstracts and articles (PubMed is a free search engine accessing the MEDLINE database) and clinicalBERT which is also a BERT model pre-trained on 2M clinical notes. They train and test the models on Twitter, and as a result, their BERT model got better results than others. Then as their final model, they create an ensemble model of these three models, which has better results than all previously reported models. However, they mentioned that such a large ensemble model might not be realistic to put into production, and using simple alternative models such as LSTM may be more suited.

With deep learning, features can be learned from large amounts of data and require no manual work to extract those features. This is a significant advantage over traditional machine learning, where each feature has to be handcrafted. However, to understand and learn patterns from data, deep learning methods require large amounts of data. Finding sufficiently large and quality annotated datasets in many domains is hard. A transfer learning strategy can be used as a solution to the insufficient training data problem. The transfer learning process involves training a model on a large-scale dataset then utilize the pre-trained model in a posterior task by training it with that task-specific dataset. Patterns that are learned from the large-scale dataset are used as a starting point for the training. We attempted to use the Bidirectional Encoder Representations from Transformers (BERT) based model by considering all these cases. We combined it with a simple BiLSTM layer as our leading solution for our research objective. Table 2.1 shows a brief overview of the literature reviews and this project contribution.

Table 2.1 Literature review and our project contribution in pharmacovigilance

Study	Year	Task	Method	Dataset
Leaman et al.	2010	ADR extraction	Lexicon based approach (Lexical pattern-matching)	Social network
Benton et al.	2011	ADR extraction	Lexicon based approach	MEDLINE articles
Yates et al.	2013	ADR extraction	Lexicon based approach	Social media sites
Liu et al.	2012	ADR detection/ extraction	Lexicon based approach	online patient forums
Gurulingappa et al.	2015	ADR extraction	Traditional machine learning approach (Support Vector Machines (SVM))	MEDLINE articles
Liu M et al.	2012	ADR detection	Traditional machine learning approach (logistic regression (LR), naïve Bayes (NB), K-nearest neighbor (KNN), random forest (RF), and SVM)	SIDER, PubChem, DrugBank, and KEGG
Nikfarjam et al.	2015	ADR detection/ extraction	Traditional machine learning approach (Conditional Random Fields (CRFs))	Social media
I. S. Alimova et al.	2017	ADR detection	Traditional machine learning approach (SVM)	Twitter corpus, CSIRO, CADEC
Cocos et al.	2017	ADR detection/ extraction	Deep learning approach (bidirectional Long Short-Term Memory (BiLSTM))	Twitter
Lee et al.	2017	ADR detection	Deep learning approach (CNN)	Twitter
Li et al.	2017	ADR extraction	Deep learning approach (BiLSTM)	PubMed
Wunnava et al.	2019	ADR extraction	Deep learning approaches (BiLSTM and CRF)	health records (EHRs) of patients in hospitals
Chen,Y et al.	2019	ADR extraction	Deep learning approach (BERT)	Twitter (SMM4H Shared Task 2019)
Breden et al.	2020	ADR detection/ extraction	Deep learning approach (BERT, BioBERT, Bio + clinicalBERT and an ensemble model of them)	Twitter
Our study	2021	ADR detection/ extraction	Deep learning approach (BiLSTM, BERT, BERT (concatenation of 4 last hidden layers)+BiLSTM	Twitter (3 specific drug)

2.1.4 Extraction (Word-Level classification) task

In this study, extraction performed as a word-level classification refers to the task where each word in the document is identified and classified as belonging to a specific pre-defined class, such as "ADR", "Drug". This task can be solved automatically by a technique in natural language processing (NLP) called named entity recognition (NER). Originally, Named Entities were proposed at the Message Understanding Conference (MUC-6) [44] to describe names of organizations, people, and locations in the text, currency, time, and percentage expressions. Throughout the text document, particular terms represent specific entities that are more informative and have a unique context. These terms are named entities, and they are terms like people (such as name, family name), places (such as cities, countries, rivers), organizations (such as companies, government organizations, committees), often indicated by proper names. Named entity recognition (NER), also called entity extraction, is a popular technique for selecting and classifying named entities using various pre-defined classification categories, as an example shown in Figure 2.1.

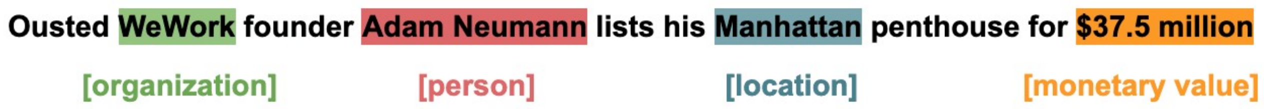

Ousted WeWork founder Adam Neumann lists his Manhattan penthouse for $37.5 million
[organization]
[person]
[location]
[monetary value]

Figure 2.1 An example for Named Entity Recognition (NER) technique

To recognize an entity, a NER model needs to detect words or strings of words (e.g., United States of America) that form an entity and recognize the entity category it relates to. So, first, we need to create entity categories, such as Name, Location, Event, Organization, and feed these categories as an annotated dataset to a NER model. Then, by tagging some word and phrase samples with their corresponding entities, the NER model will eventually be taught to detect entities themselves.

2.1.5 Detection (Sentence-Level classification) task

In this study, detection performed as a sentence-level classification refers to the task where each sentence (each user review) in the document is identified and classified as belonging to a specific pre-defined class. Detecting adverse drug reactions is considered as a binary classification in which there are only two classes. For example, a review containing adverse drug reactions is classified as "adverse drug detected", and on the other hand, if there is no adverse drug reaction in the review, it is classified as "adverse drug not detected".

2.2 Representation of words in natural language processing (NLP)

In general, the text data format cannot be directly processed into the language deep learning models, and it needs to be converted into a vector or matrix of numbers.

2.2.1 One-hot encoding, word embedding, embedding layer

The most straightforward technique to represent a word is the one-hot representation. To present the words by the one-hot representation method, all the unique words are gathered in a vocabulary and assigned a unique index to form a bag of words. The one-hot vector size will be the same as the created vocabulary. In one-hot representation, for the i-th word of the vocabulary, the vector's value at the index i is 1 and all others are 0. For example, for the sentence " I got my Avastin injection", the vocabulary and one-hot vectors are as follow:

$$Vocabulary = \{I, got, my, Avastin, injection\}$$

$$\begin{aligned} I =& [1, 0, 0, 0, 0] \\ Avastin =& [0, 0, 0, 1, 0] \end{aligned} \tag{2.1}$$

There are two significant issues with this representation approach. The first issue is the high dimension of a one-hot vector when presenting a large vocabulary. Most of the vector is taken up by zeros, so valuable data are sparse. The second issue is that it does not consider any relation between the words. With one hot representation, it can be able to recognize the exact words but not similar words. The word embedding approach was introduced to solve these issues. Word embeddings is a method to transform a word into a real-valued vector in a low-dimensional space instead of a large binary vector with many zeros (one-hot vector). Moreover, the objective is to make the distance between vectors of two semantically close words (for example, "feline" and "cat") small while making it large for any random pair of words ("car" and "cat") to solve the lack of meaningful relationship issue in the one-hot method. As shown in Figure 2.2 in the representation space, similar words have similar distances.

When deep learning models are applied to language processing tasks, the first layer is always an embedding layer. The embedding layer will convert text inputs to word embedding vectors. These vector representations of each word (word embeddings) are trainable parameters. Their

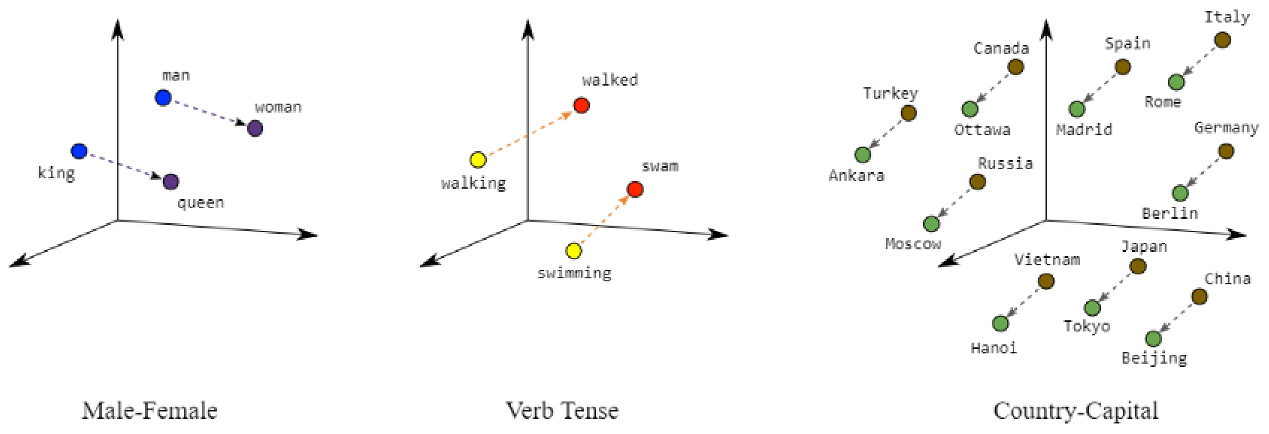


Figure 2.2 Semantic relationship between words in word embedding representation [2]

values are first initialized randomly and then learned by the model during training. In the same way, a model learns the weights of layers as the parameters. Two values must be determined for defining the embedding layer: the number of embeddings (the size of the dictionary of embeddings) and the dimension of the embedding (the size of each embedding vector). The size of the embedding dictionary is equal to the number of unique words in the dataset, and the size of the embedding vector is the number of dimensions in which each word is represented. For calculating embedding dimension Equation 2.2 is suggested by Google Developers [45]. They mentioned that this formula is just a general guideline, and researchers can set the number of embedding dimensions as they please.

$$embedding\ dimensions = (number\ of\ unique\ words)^{0.25} \tag{2.2}$$

2.3 Overview of deep learning models in this study

Neural networks are a set of algorithms inspired by the functioning of the human brain. Feed-forward neural networks are the most common type of neural network that consists of different layers, one input layer, one or more hidden layer(s), and one output layer Figure 2.3.

The layers consist of basic units called neurons or nodes. All these nodes are connected and have associated weights which are model parameters. Each node receives data from several nodes connected to it from the previous layer and sends its output to the nodes in the next layer.

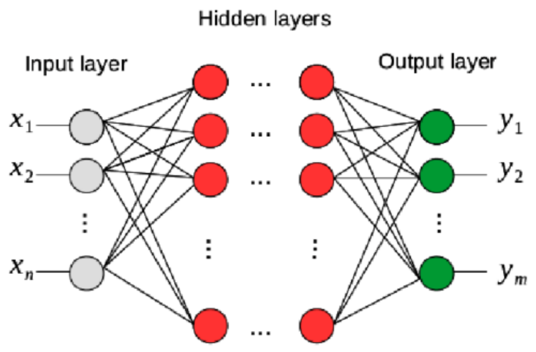


Figure 2.3 The architecture of a deep feed-forward neural network [3]

As shown in Figure 2.4, inside each neuron, the weighted average of received inputs are computed, and this sum value is fed to a nonlinear function called the activation function.

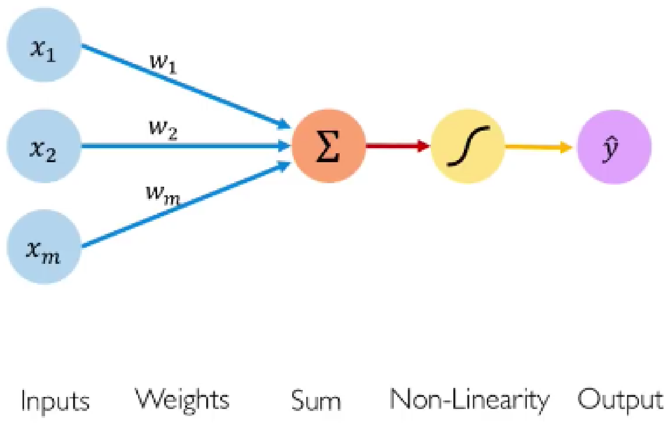


Figure 2.4 The inner structure of a neuron

2.3.1 Recurrent Neural Network and LSTM model

Recurrent Neural Network (RNN)

In the case of sequential models such as language and text processing, the sequence, the order in which one word follows another, plays an important role. This is because a single input item from the series is related to others in the sequence and likely influences its neighbors. For example, in a sentence, the meaning of a word mostly depends on the previous words in the sentence. So, for implementing language-related tasks, a link between sequences are needed.

To obtain such a link, a loop between higher- and lower-layers is needed in the feed-forward neural network for feeding signals of earlier timesteps back into the network (Figure 2.5). These networks with recurrent connections are called Recurrent Neural Networks (RNN) [46]. A recurrent neural network is a feed-forward neural network rolled out over time. With the help of these loops within the network, RNN can remember what they have learned from prior inputs. As a result, RNNs build a memory of time-series events. In addition, they may handle different lengths of inputs (which was another limitation for feed-forward neural networks in sequential data) because each new data point can be fed into the model for the duration of the sequence, and there is no pre-set limit to the size of the vector.

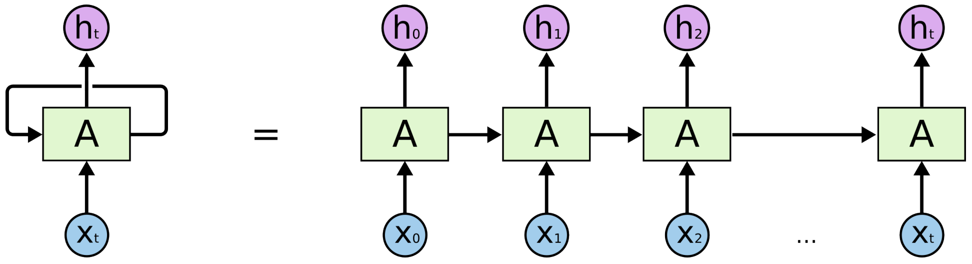


Figure 2.5 Architecture of an unrolled recurrent neural network [4]

For some tasks, we may only need to consider the most recent information available. An example could be predicting the next word based on the previous words. An RNN can be successfully used when there is a small gap between the relevant information and its needed location. However, RNNs are unfortunately unable to learn links between distant data elements as that gap between relevant information and the point where it is required grows. Furthermore, this model fails to take into account long-term dependencies that may lead to leaving out valuable information. Practical RNNs are limited to look back in time to approximately ten timesteps [47]. The fed-back signal vanishing causes the problem; the continuous multiplication of partial derivatives leads to vanishing gradient values (these values are used to adjust the network's internal weights).

Long Short-Term Memory(LSTM)

A more complex network based on a recurrent architecture with long short-term memory (LSTM) was developed to solve short-term memory issues by introducing a memory unit into the network (Figure 2.6). An LSTM recurrent neural network can learn more than 1,000 timesteps, depending on its complexity [48].

An LSTM memory unit contains three gate mechanisms: forget gate, input gate, output gate, and the cell state. The cell state can store temporal information that has been written into it,

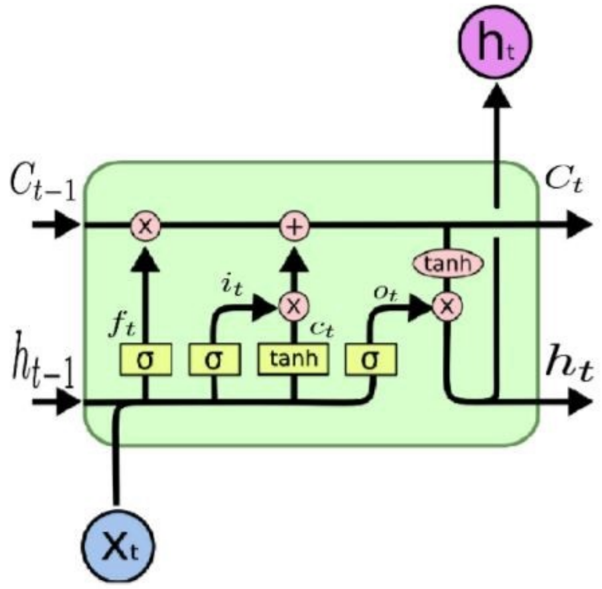


Figure 2.6 The architecture of LSTM memory unit [4]

and the network can read the stored information. The three gates control how information enters and leaves the cell. The forget gate decides how much of the old memory is to be preserved, the input gate determines how much of this unit is added to the current state, the output gate determines what the next hidden state should be.

LSTM works according to a three-step process [4]. In the first step, the forget gate decides what information will be thrown away from the cell state. The sigmoid function $\sigma()$ (which is a function that maps the whole real range of z into $[0, 1]$, decides it. It takes the previous state $h_{\text{t-1}}$ and the current input(x_{t}) and outputs a number between 0 and 1 for each number in the cell state $C_{\text{t-1}}$ (Equation 2.3).the f_t is the forget gate output, W_f represents the forget gate weight matrix, and b_f represents the forget gate bias vector. An output 1 value means keep this information, and 0 means omit it.

$$f_t = \sigma(W_f.[h_{\text{t-1}}, x_t] + b_f) \tag{2.3}$$

In the next step, the input gate decides which new information will be stored in the cell state; a sigmoid function determines which values to let according to its 0,1 output. The i_t returns the output of the input gate, W_i represents the input gate weight matrix, and b_i represents the input gate bias vector. At the same time, a *tanh* function gives weight to the passed values deciding their level of importance ranging from -1 to 1, creating a vector of new candidate values, $\widetilde{C}_t$ (Equation 2.4).

$$\begin{aligned} i_t &= \sigma(W_i.[h_{\text{t-1}}, x_t] + b_i) \\ \widetilde{C}_t &= \tanh\left(W_c.[h_{\text{t-1}}, x_t] + b_c\right) \end{aligned} \tag{2.4}$$

To create the new cell state, we multiply (element-wise product ($*$)) the old state $C_{\text{t-1}}$ by f_t which was calculated earlier, and add $i_t * \widetilde{C}_t$ (Equation 2.5). This is the new candidate values, scaled by how much we decided to update each state value.

$$C_t = f_t * C_{\text{t-1}} + i_t * \widetilde{C}_t \tag{2.5}$$

The final output is based on the cell state with some modifications. First, a sigmoid function decides what part of the cell state is to be sent to the output, then to regulate the cell state value between -1 and 1, a *tanh* function is used. The final output is obtained by multiplying these two functions (Equation 2.6). The o_t returns the output gate's output, W_o represents the output gate weight matrix, and b_o represents the output gate bias vector. Finally, the h_t indicates the final value of the LSTM memory cell.

$$\begin{aligned} o_t &= \sigma(W_o.[h_{\text{t-1}}, x_t] + b_o) \\ h_t &= o_t * \tanh(C_t) \end{aligned} \tag{2.6}$$

Each LSTM cell only considers the previous context but does not utilize the future context. In text processing, capturing only the previous words is not sufficient to fully understand the context of a word in a text. Indeed, it is more efficient to consider words occurring after the word under consideration. Schuster and Paliwal [23] invented a bidirectional recurrent neural network (BRNN) to overcome this limitation.

A BiLSTM combines two independent LSTM, one that looks left to right (with the output of $\overrightarrow{h}_t$) and the other look right to left (with the output of $\overleftarrow{h}_t$) and then combines their output at inference, usually by simple concatenation (Equation 2.7).

$$\begin{aligned} \overrightarrow{h}_t &= \sigma(W_{\overrightarrow{h}x} x_t + W_{\overrightarrow{h}\overrightarrow{h}} \overrightarrow{h}_{t-1} + b_{\overrightarrow{h}}) \\ \overleftarrow{h}_t &= \sigma(W_{\overrightarrow{h}x} x_t + W_{\overrightarrow{h}\overrightarrow{h}} \overleftarrow{h}_{t+1} + b_{\overrightarrow{h}}) \\ y_t &= (W_{y\overrightarrow{h}} \overrightarrow{h}_t + W_{y\overrightarrow{h}} \overleftarrow{h}_t + b_y) \end{aligned} \tag{2.7}$$

2.3.2 Transformers and BERT model

The transformer concept was proposed by Vaswani et al. [5] in the famous paper "Attention Is All You Need" to solve the problem of language translation and was very well received. However, LSTMs and other RNN methods use sequential word input and output, and for a neural network to learn, there can be many timesteps required, making them slow. Further, the BiLSTM method is not the most accurate to capture the meaning of words for the simple reason that it is learning left to right and right to left separately and then concatenating them. As a result, the actual context is somewhat lost. The transformer architecture, however, addresses some of these concerns.

First, they are faster since they can process words simultaneously, which means they apply the process in parallel. Second, they are bidirectional and learn context simultaneously from both directions, so they can better understand what the words mean.

Like LSTM, a transformer is an architecture which takes a sequence as an input and returns another one. However, unlike recurrent models, a transformer relies on an attention mechanism to represent word sequences rather than recurrence. Transformers have an increased ability to retain long-term information due to the attention mechanism. It can "attend" or "focus" on all previous tokens that have been generated. The concept of attention was introduced for the first time in [49], it was used to solve the forgetting problem of RNNs and to improve their contextual awareness. Attention is nothing more than saying which part of the sentence should get more importance. In other words, the attention mechanism focuses on the certain parts of the input, while the rest of the input is in low focuses. Rather than relying solely on the prediction based on the RNN's final hidden state representation, the network can look back over previously hidden states to detect context. In addition, the model may be able to detect more complex context information.

As shown in Figure 2.7, a transformer consists of two key components, an encoder, and a decoder. On the left is the encoder, and on the right is the decoder. In the first step, input is fed to the embedding layer. Essentially, a word embedding layer is a lookup table for mapping words to a vector of real numbers for representation. In the next step, there is a positional encoding layer. As mentioned before, the transformer removes the recurrent structure. Instead, it uses an attention mechanism, so there is no recurrent network used to encode a model's sequence pattern. Keeping the positional information of inputs is an important issue. This is done using positional encoding with sine and cosine functions. So by using positional encoding, the positions are added to the embedded representation of each word.

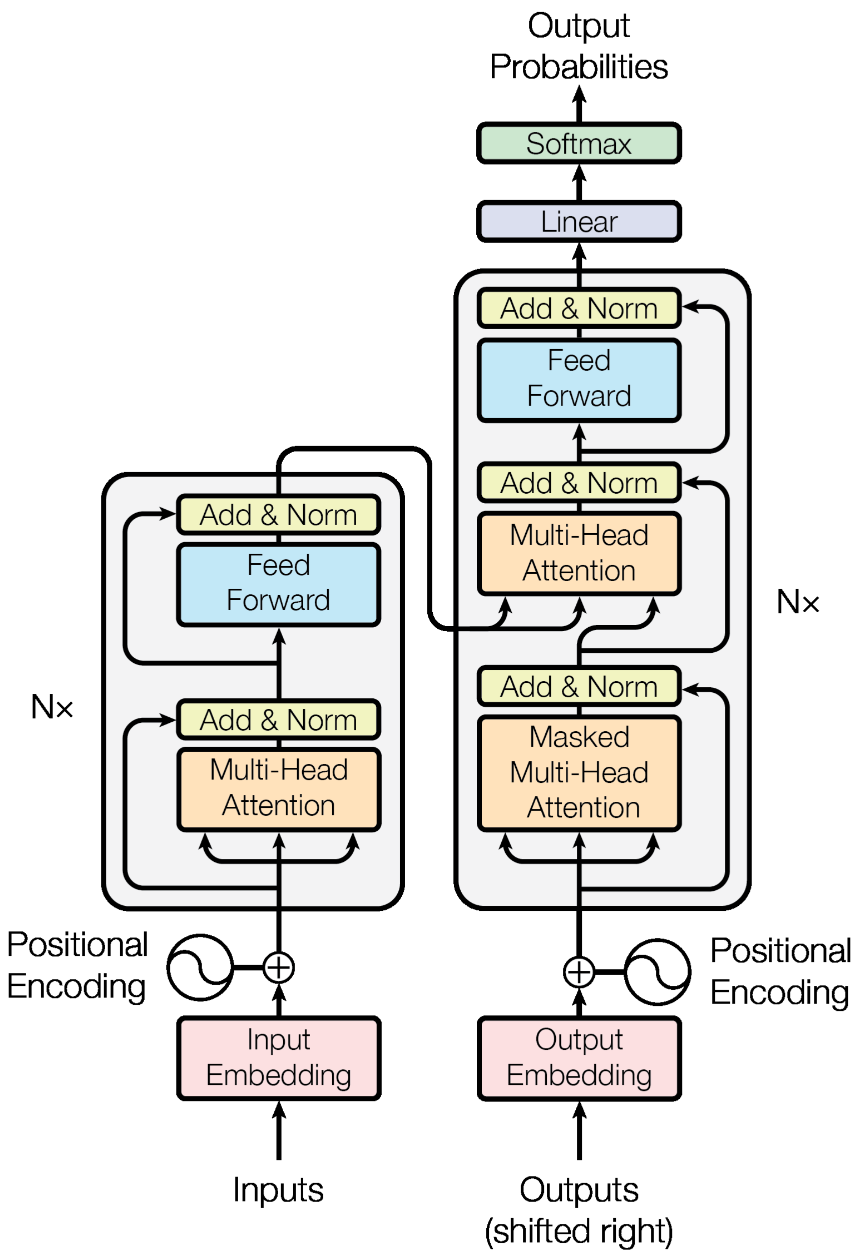


Figure 2.7 The transformer model architecture extracted from [5]

The encoder layer consists mainly of Multi-Head Attention and Feed Forward layers. The core technology in Transformers is the "Self-Attention" mechanism. Self-Attention is a specific attention mechanism that is used in Multi-Head Attention. Self-Attention uses a token pair-wise scoring system instead of a recurrent network to associate each word in the input to other words. To be more specific, given a word in a sequence, the algorithm can learn which other words to prioritize in determining the meaning of a word. As every token in the input is processed, self-attention looks at all other tokens to detect possible dependencies.

To achieve self-attention, we need the Queries (Q), Keys (K), and Values (V) of the embedding matrix (Equation 2.8). Q, K, and V are query, key, and value vectors stacked together as matrices and K^T is the transpose of matrix K. These matrices do not change across input tokens.

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.8}$$

The Queries (Q), Keys (K), and Values (V) matrices are obtained by multiplying the input embedding vectors (X) by three different learnable weight matrices W_Q, W_K, and W_V Figure 2.8.

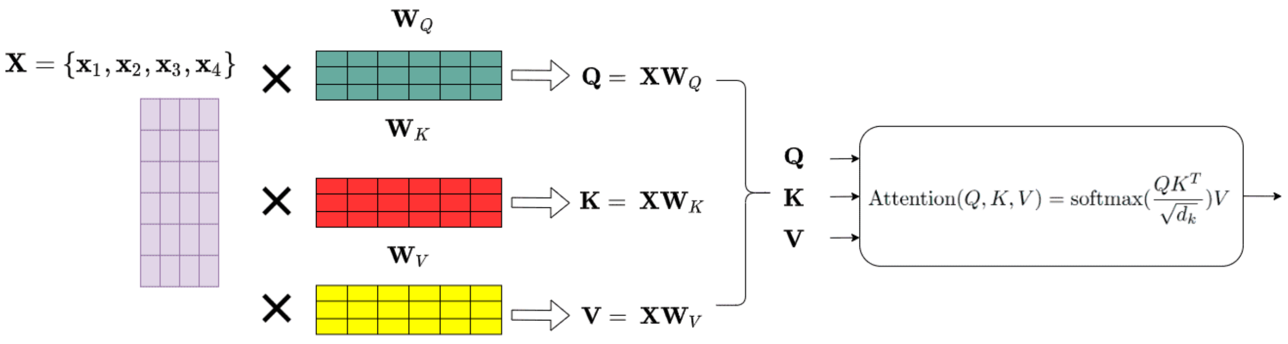

$\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\}$
$\mathbf{W}_Q$
$\mathbf{W}_K$
$\mathbf{W}_V$
$\mathbf{Q} = \mathbf{XW}_Q$
$\mathbf{K} = \mathbf{XW}_K$
$\mathbf{V} = \mathbf{XW}_V$
Q
K
V
$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$

Figure 2.8 Queries (Q), Keys (K), and Values (V) of the embedding matrix [6]

This is followed by a dot product matrix multiplication of the queries and keys to produce a scoring matrix. The score matrix can determine how much attention each word should get; the higher the score, the more focus one has. Afterward, the product will be divided by the square root of the dimension of the key vector ($\sqrt{d_k}$), as multiplying values can generate vary large values; this step allows for more stable gradients. Finally, a softmax function (which is a function that calculates the relative probabilities) is applied to the scaled score in the next step to get the attention weights. The higher scores are boosted with a softmax, and the lower scores are lowered, and we get the final attention weights as a probability distribution. Finally, these attention weights are multiplied by the value vector to get an output vector (Equation 2.8). The goal is to keep only the values v of the input word(s) we want to focus on by multiplying them with high probability scores and remove the rest by driving them towards 0 by multiplying them with the low probability scores. So now, the output is ready to be fed into a linear layer to process.

In order to make this self-attention mechanism into a multi-headed computation (in other words, run attention mechanism several times in parallel), it is necessary to split the key, the value, and the query into N vectors before applying self-attention. Then, each split vector is subjected to the self-attention algorithm. In the transformer structure, each self-attention process is called the head. The output from each head is then concatenated before it is passed through the final linear layer (Equation 2.9). The "concat" in the equation refers to the concatenation function. Thus, multi-head attention allows us to pay attention to different parts of the sequence differently each time.

$$\begin{aligned} Multihead(Q,K,V) &= concat(h_1,...,h_h)W^o \\ h_i &= Attention(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{2.9}$$

Other features of the transformer structure include residual connections and a normalization layer. A residual connection reduces the problem of vanishing or exploding, and the normalization layer ensures that all layers' features have the same scale.

The next transformer key component, the decoder, has a self-attention, encoder-decoder attention, and feed-forward layer (Figure 2.7). Most concepts of the decoder are already familiar; only the encoder-decoder is a new concept. The self-attention looks at the input from the previous layer in the decoder stack. In contrast, the encoder-decoder attention looks at combining the self-attention layer and input from the encoder stack. In the self-attention of the decoder, future tokens are "masked" when computing attention for that token. Finally, the encoder pushes the last encoded vector to all layers in the decoder stack.

BERT (Bidirectional Encoder Representations from Transformers):

What makes the transformer architectures more practical than the LSTM cell is that both the encoder and the decoder, even separately, have some basic understanding of the language, so each part (encoder, decoder) can be used separately to build systems that understand language. At the end of 2018, Google published their NLP model called Bidirectional Encoder Representations from Transformers (BERT) by using and stacking the encoder part of the transformer. The original transformer was meant to be used for language translation tasks, but BERT can be used for language translation, Question Answering, Sentiment Analysis, Text summarization, and many other tasks.

Devlin et al. [7] provided researchers with several pre-trained models along with the paper. The experiments reported in [7] used two of the released pre-trained models:

- BERT Base, Uncased: 12 layers, 768 hidden layers, 12 heads, 110M parameters
- BERT Large, Uncased: 24 layers, 1024 hidden layers, 16 heads, 340M parameters.

The BERT can be used to understand language and then fine-tune it depending on the problem to solve. Training a BERT is done in two phases. The first phase is pre-training, where the model is trained to understand the language and context, and the second phase is fine-tuning, where the model learns how to solve a specific problem. There are three essential steps before feeding the input to the BERT model. The first step, called tokenization, converts the sequence of elements to a set from which each member is referred to as a token. After tokenization, words are projected in a geometrical space or called word embeddings. The order is lost by converting a sequence into a set (tokenization), and positional encoding is suggested as a solution.

BertTokenizer converts the cleaned data into BERT-compatible input in two steps:

- Basic tokenization, normalizes the text and separates the punctuation. The first step is to lowercase the input, convert all whitespace characters to spaces, and remove accent marks. The next step adds space around punctuation characters (not letters, numbers, or spaces).
- WordPiece tokenization, Out-of-vocabulary (OOV) tokens are replaced with a special token [UNK], which stands for unknown token. Conversion of all unseen tokens to [UNK] will remove a great deal of information from the input data. Hence, BERT uses a WordPiece algorithm to segment words into subword levels, such that the model can also represent commonly seen subwords and get richer word information. For example, the different tenses "training," "trains," and "trained" will all be segmented into "train #ing," "train #s," and "train #ed." BERT's English vocabulary consists of 30,522 segmented words that are learned beforehand.

To complete the tokenization, BERT also adds the following special tokens. For the CLS Token, every sequence begins with a special classification token ([CLS]). CLS stands for classification, and it is there to represent sentence-level classification. For the SEP Token, the token is inserted at the end of each sentence to separate the sequences specifically for sequence-pair tasks. It is appended at the end of the sequence when a single sequence is used in other tasks.

Pre-training process:

The purpose of pre-training for BERT is to improve understanding of the context. BERT learns it by training two unsupervised tasks simultaneously: Masked Language Modeling (MLM), which is a multi-class classification problem, and Next Sentence Prediction (NSP), which is a binary classification problem.

The training works through the following training procedure. First, two sentences are sampled from the corpus, either following each other or from different parts of the corpus. Then, in MLM, 15% of the words in both sentences are masked with a special [MASK] token, and the model is taught to predict these words. By providing these mask tokens, BERT can know the context bidirectionally. At the same time, in the following sentence prediction, BERT determines if the second sentence follows the first. As a result, BERT can figure out context across multiple sentences and, using these two techniques together, BERT becomes knowledgeable about various languages.

On the input side, BERT takes a sequence of words as input, just like any encoder of the transformer. The initial embedding is constructed from three vectors (Figure 2.9).

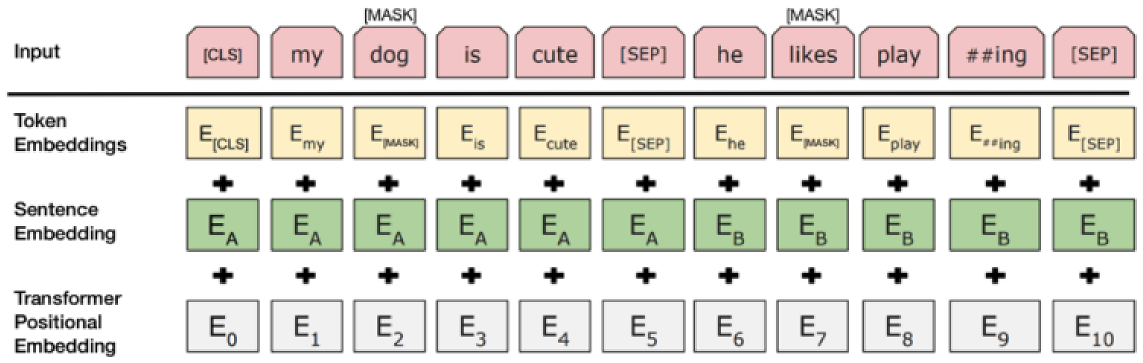

Input
[MASK]
[MASK]
[CLS] my dog is cute [SEP] he likes play ##ing [SEP]
Token Embeddings
Sentence Embedding
Transformer Positional Embedding

Figure 2.9 BERT different Tokenization layers [7]

- The token embeddings layer transforms each WordPiece token into a vector representation of a fixed length. This fixed length is 768 and 1024 for BERT Base and BERT Large, respectively.

- Segment embeddings are primarily relevant for sentence pair classification, where BERT needs to distinguish the tokens in each input pair. This layer consists only of two vector representations, namely zeros and ones. The vector that consists of zeros is assigned to the first input sequence, i.e., every token to the left of the [SEP] token. The other vector with ones is assigned to the second input sequence, every token to the right of the [SEP] token. In cases with only one input sentence, the whole vector is filled with only zeros. The length of these token vectors is the same as for token embeddings.

- The final layer consists of positional embeddings. BERT's design allows for input sequences up to length 512. This layer is the same as positional embedding described in the transformer section.

These three vector representations are summed elementwise to construct a single vector representation. The final vector is used as an input representation for BERT's encoder stack.

As shown in (Figure 2.10), on the output side, there are C and T_is as model outputs. The C is the binary output for the sequence-pair task. It means that if sentence 2 follows sentence 1 in the context, the C value will be 1; otherwise, it will return 0. The Ts are numerical vectors of representation of each word in the input sequence. The number of word vectors (Ts) is equal to the number of input tokens from 1 to N in sentence 1 and 1 to M in sentence 2. The word vectors are then converted into distributions to train the BERT model based on cross-entropy loss. After completing these two unsupervised training tasks, the pre-trained BERT model will understand word contexts.

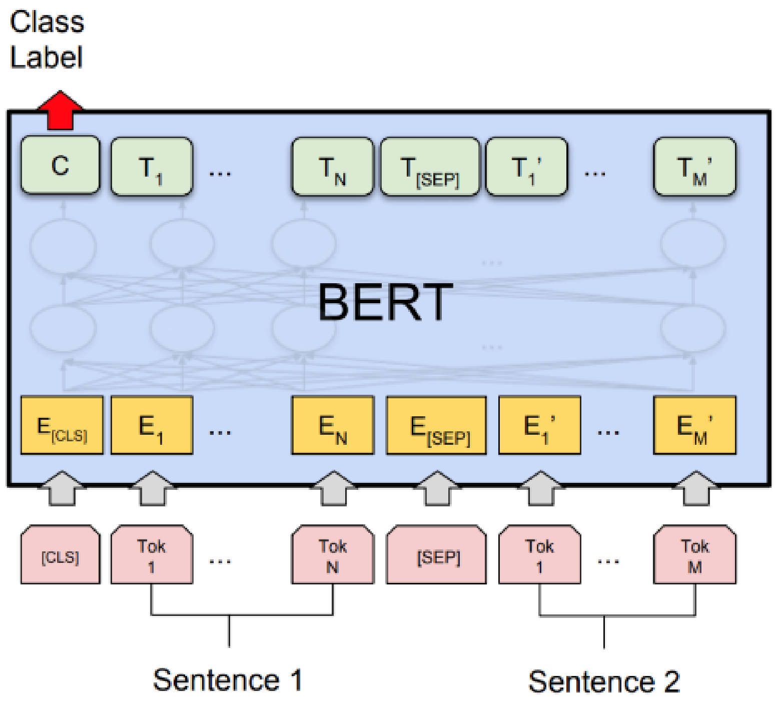


Figure 2.10 BERT classification output (C) [7]

Fine-tuning process:

Fine-tuning the BERT means using the pre-trained BERT model on specific NLP tasks by replacing the fully connected output layers with the layers corresponding to our task and then performing supervised training with the specific dataset. Since only the output parameters are learned from scratch, and the rest of the model parameters are just slightly fine-tuned, the training time is fast, and we can do it for any NLP problem.

2.4 Training process description

The goal of a deep learning neural network model is to learn how to map inputs to outputs automatically. To achieve this, the model passes a training process in which the network's weights (parameters) are updated based on the errors made by the model in the prediction of the training dataset in the different iterations of the training process. It is continually updated to reduce this error until either a good enough model is found or the learning process stops. The training process consists of two iterative parts, forward propagation, and backpropagation through the model's layers. The first phase, forward propagation, is the process in which input is fed into the network in the forward direction. The hidden layers accept input data, process it according to the activation function, and pass it to the next layer. The final layer will be reached with a result of label prediction for the input. Following that, a loss function will be used to estimate the loss (or error) and compare and measure how well or poorly the prediction did compare to the actual result. After calculating the loss, this information is propagated backward, which is called the backpropagation step. This is the step where the error is sent backward to update the weights. Now that the error information is spread back, the network parameters' weights can be adjusted. This process will be repeated until the error is as close as possible to zero. For this, a technique called gradient descent is used. Gradient descent is an optimization algorithm used to find the values of parameters of a function that minimizes a loss function. It changes the weights by computing the derivative (or gradient) of the loss function, which shows the direction towards the global minimum; this is generally done on batches of data in successive iterations (epochs) of all the passed datasets to the network in each iteration.

2.4.1 Loss function

The loss function is based on the difference between a predicted value and the actual label. Cross-Entropy is a common loss function in multiclass and binary classification. The cross-entropy measures the difference between two probability distributions. It means that the distributions are very different if the cross-entropy value is high. On the other hand, if the cross-entropy value is low, the distributions are similar.

2.4.2 Hyperparameter

Hyperparameter variables are entirely different from the model parameters (weights); they control the training process and determine the model architecture (for example, the number of hidden layers). Hyperparameters are specified manually before the training process starts, while the training process automatically determines the model parameters. Among these values, one can find:

- Learning rate: Learning rate is a hyperparameter that governs the step size that the model should take toward achieving the minimum loss function value in each iteration. Learning rate is a crucial hyperparameter since it may prevent the model from converging if it is chosen incorrectly.
- Batch size: Batch size refers to the number of samples passed to the network at once. Batches are also known as mini-batch. To improve training speed, the user reviews in the input are batched into mini-batches.
- Epoch: Each complete forward and backward pass of the dataset into the model is called 1 epoch. In an epoch, each sample in the training dataset updates the internal model parameters.

2.4.3 Overfitting

Overfitting happens when a model starts to memorize the data; in other words, it fits precisely against its training data by learning the details and noises. The overfitted model achieves high accuracy in predicting the training dataset but low accuracy in predicting the unseen data of validation and test set. Following the training and validation loss graph helps to detect the overfitting. The point (epoch) where the training loss keeps decreasing while the validation loss starts to increase, implies that the model is likely to face overfitting. This is visualized in Figure 2.11. The training of the model should be stopped when the validation loss starts to increase.

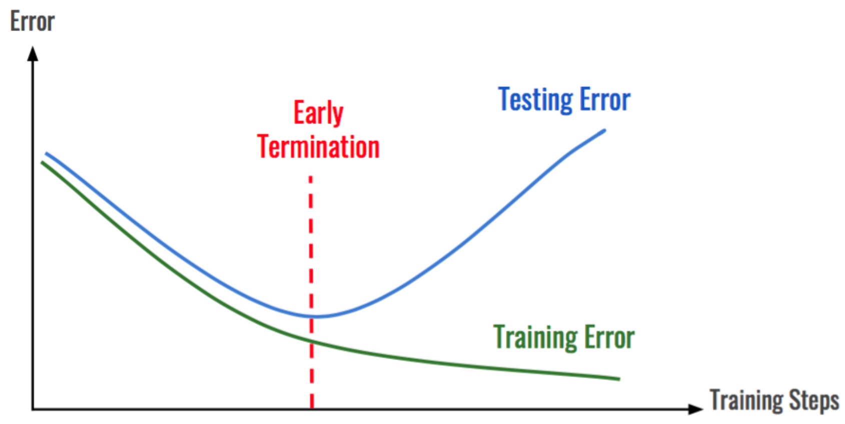


Figure 2.11 The model should stop at early termination point [8]

Another method to combat overfitting is to use dropout layers. The Dropout layer randomly sets input units to 0 with a rated frequency at each step during training time.

2.5 Evaluation metrics for extraction and detection tasks

Accuracy is measured by dividing the number of correct predictions from the total number of predictions and multiplying by 100 to turn it into a percentage. It is possible to achieve seemingly good accuracy with a model that guesses the largest class every time if the class sizes are uneven. When the classes in the dataset are imbalanced, the three performance measures, precision, recall, and F1-score, are used to have more accurate and reliable metrics for evaluating the performance of the tasks. To describe a classification model's performance, the notations true positive (TP), true negative (TN), false positive (FP), and false negative (FN) are used. Classifiers use positive and negative terms to indicate their prediction on an instance, while true and false indicate whether or not the prediction corresponds to the actual class.

- True positive (TP) = Number of correctly labeled positive samples
- False positive (FP) = Number of negative samples incorrectly labeled as positive
- True negative (TN) = Number of correctly labeled negative samples
- False negative (FN) = Number of positive samples incorrectly labeled as negative

Each time a model predicts that a token in the sentence of input data belongs to a particular class of label, it is by default predicting that this token is not a member of the other k−1 label classes (K is the total number of classes). According to this description and previous definitions, the number of true and false positive (and negative) model predictions are calculated and reported in a table called the confusion matrix. Moreover, the Precision, Recall, and F1-score can be computed using these true and false positive (and negative) values based on the following definitions.

Precision is defined as the ratio of the correct number of predictions to the total number of predictions. If a model achieves high precision, it means that most of the predicted positive instances were correctly classified. Thus, precision is a good evaluation metric when we want to validate our prediction (Equation 2.10).

$$Precision = \frac{TP}{(TP + FP)} \tag{2.10}$$

For a certain class, the recall is the fraction of successfully classified texts relevant to that class. With high recall, it means that a model has identified most of the actual positive instances correctly. Thus, recall is a valid choice of evaluation metric when we want to capture as many positives as possible (Equation 2.11).

$$Recall = \frac{TP}{(TP + FN)} \tag{2.11}$$

The F1-score can be seen as a harmonic mean of the precision and recall values. It is a number between 0 and 1, and values close to 1 mean the model performs better. When both good precision and recall are critical, the F1-score can be used as an evaluation metric. (Equation 2.12).

$$F1\_score = 2 \times \frac{Precision \times Recall}{(Precision + Recall)} \tag{2.12}$$

These individual scores can then be averaged in three ways: micro, macro, and weighted averaging. The global average of the total true positives, false negatives, and false positives of all different classes is called the Micro average defined by Equation 2.13, where K is the number of label classes, in all equations.

$$\begin{aligned} Micro\ Precision &= \frac{\sum_{i=1}^{K} TP_i}{\sum_{i=1}^{K}(TP_i + FP_i)} \\ Micro\ Recall &= \frac{\sum_{i=1}^{K} TP_i}{\sum_{i=1}^{K}(TP_i + FN_i)} \\ Micro\ F1 &= 2 \times \frac{Micro\ Precision \times Micro\ Recall}{(Micro\ Precision + Micro\ Recall)} \end{aligned} \tag{2.13}$$

Macro averaging calculates metrics for each label and finds their unweighted mean as defined by Equation 2.14. Macro average takes the average of the precision and recall of the system on different sets.

$$\begin{aligned} Macro\ Precision &= \frac{1}{K}\sum_{i=1}^{K} Precision_i \\ Macro\ Recall &= \frac{1}{K}\sum_{i=1}^{K} Recall_i \\ Macro\ F1 &= \frac{1}{K}\sum_{i=1}^{K} F1_i \end{aligned} \tag{2.14}$$

Weighted averaging calculates metrics for each class and takes their average weighted by the number of true instances for each label as defined Equation 2.15, where M is the number of instances, with M_i being the number of instances of class i.

$$\begin{aligned} Weighted\ Precision &= \frac{1}{M}\sum_{i=1}^{K} M_i Precision_i \\ Weighted\ Recall &= \frac{1}{M}\sum_{i=1}^{K} M_i Recall_i \\ Weighted\ Recall &= \frac{1}{M}\sum_{i=1}^{K} M_i F1_i \\ M &= \sum_{i=1}^{K} M_i \end{aligned} \tag{2.15}$$

2.6 NER labeling method for evaluation

Since the named entity recognition task involves tagging token and multi-token entities with predefined classes, it sets a need for an evaluation method different from other classification tasks. There are two kinds of methods to evaluate the predicted results in NER tasks: the token-level and entity-level methods. For example, consider the sentence "Avastin gives me a minor blood nose." after prediction using BOI labeling model (will be described in Section 3.1.3), the result is like Table 2.2:

Table 2.2 Samples for named entity recognition (NER) prediction

Tokens	Ground true Label	Predicted Label
Avastin	B-Drug	B-Drug
gives	O	O
me	O	O
a	O	O
minor	B-Adverse Event	B-Adverse Event
blood	I-Adverse Event	I-Adverse Event
nose	I-Adverse Event	O

Avastin with B-Drug is a single token entity, and (minor, blood, nose) with (B-Adverse Event. I-Adverse Event, I-Adverse Event) are multi-token entities. In the evaluation process, the token-level method is used if each of these predicted tokens is compared separately with related true labels. For example, the token-level method will consider the predicted label of minor and blood true and consider the predicted label of the nose as false. However, for the same example, the entity-level method offers a different approach. Two possible sources of errors occur in model prediction in multi-token entities: entity boundary-related error or entity type-related error. The B- and I- prefixes determine the boundary of an entity, and the type of entities are predefined classes such as Drug or Adverse Event.

There are different evaluation schemes for entity-level evaluation in the literature, such as CoNLL (Computational Natural Language Learning), Automatic Content Extraction (ACE), Message Understanding Conference (MUC). CoNLL is used for this study since it is most common, simple, and strict when compared with the other methods, and it is closer to a measure of performance on the actual task; a user of the NER system cares about entities, not individual tokens. In CoNLL, three prediction scenarios will be checked as follows: both entity boundary (start and end point of entity) and type are correct, a predicted entity does not exist in the ground truth, the entity exists in the ground truth, but it is not predicted by the NER system. For example, the CoNLL considers the Avastin predicted label as the only true prediction in the previous example's sentence seen in Table 2.2.

2.7 Data augmentation techniques

Although good performance is achieved due to improved language models, the size of training data still plays a key role in this field [50]. Data augmentation is a strategy to increase the amount of training data by adding modified existing data without collecting new data. Thus, it helps to solve the data scarcity and overfitting problem and reduces the cost of annotating more data. This method was first used for image data such as LeNet-5, which is one of the first applications of CNNs on handwritten digit classification using data warping to create an image augmentation and an increased model performance [51].

Compared to computer vision, data augmentation techniques in NLP have not been thoroughly explored since it is difficult to find appropriate transformation techniques that also preserve the contextual and grammatical structure of natural language texts. However, in recent years, we have witnessed several successful methods in the context of text data augmentation. Synonym replacement is one of the common strategies used by many works such as [52], [53]. The basic mechanism is to replace words in training dataset sentences with words from a created thesaurus of synonyms by randomly choosing the closest synonym for the selected word. Another NLP data augmentation strategy called back-translation translates all texts from one language to another and back again. Yu et al. [54] generated new data by translating sentences into French and back into English. Ma & Li [55], used Chinese as their intermediate language and translated it back to English for data augmentation. Fadaee et al. [56] propose a method based on the generation of additional training data by using low-frequency words. Easy Data Augmentation (EDA) [57] was proposed as a means to introduce some random modification on the main training dataset such as random insertion, random swap, and random deletion to create new data for adding to the training dataset. Among text augmentation methods, EDA and back-translation are two simple and effective methods to enlarge the dataset and improve the model's performance that we used for this study. Moreover, inspired by EDA, 4 data augmentation operations are proposed, and their efficiency to improve the model results are explored. Some descriptions of EDA and back-translation methods are given below.

Easy data augmentation (EDA) consist of the following operations :

- Synonym Replacement (SR): randomly selecting n words (should not be stop word) and replacing them with their synonyms.
- Random Insertion (RI): Insertion of n random synonyms of a word in n sentence at random positions.
- Random Swap (RS): selection of a pair of words in a sentence and randomly swap their positions n times.
- Random Deletion (RD): Based on a probability value of p, randomly delete some words of a sentence.

According to the sentence length l, a value n is computed with the formula $n = \alpha l$, where α indicates what fraction of the words in a sentence are changed and for RD, $p = \alpha$.

Back-Translation: Back-Translation (BT) is a technique that generates synthetic data for NLP. With this technique, sentences are taken from the main training set; these sentences are translated to a target language and then retranslated to the source language. This process generates more training data for the model.

CHAPTER 3 EXTRACTION OF ADVERSE DRUG REACTION

3.1 Extraction of ADR introduction

This study aims to extract some unreported adverse drug reactions from social media such as Twitter. Figure 3.1 shows the several steps which are taken to implement this task. The first attempt is to create a proper dataset and then choose a model with the best performance for our task. To find the model with the best performance, a different model is applied to the created dataset and tested. Details in this chapter explore each step.

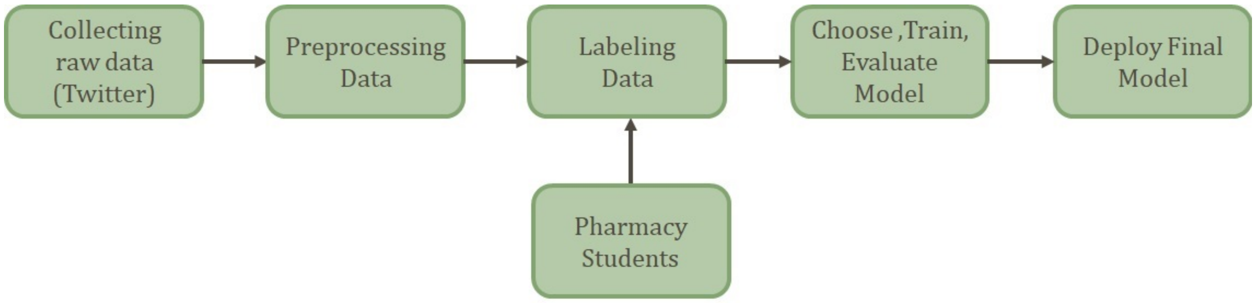


Figure 3.1 ADR extraction task pipeline

3.2 Dataset preparation steps

A model requires training data to learn what is and is not a relevant entity and how to categorize them in the extraction task. The more relevant the training data is to the task, the more accurate the model will be in requested tasks. So, the first step is to define specific categories (classes or labels) and prepare an accurate dataset.

3.2.1 Collecting data from Twitter

Users can share their drug experiences on social media websites, such as Twitter, Facebook, and Google Circle. A great deal of that information is not shared with healthcare providers or the FDA in the USA, or similar organizations in other countries. It is estimated that more than 50 million posts are made every day on Twitter [58]. Thus, Twitter offers rich, large-scale multimedia data relevant to many different research projects involving automatic ADR extraction. Moreover, this platform makes it easy and safe to access the data via several

Application Programming Interfaces (API). As a result, Twitter is considered the primary resource for collecting data in this study.

Researchers can download and access the content of tweets posted to the public using the Twitter API. However, there is a limitation; Twitter does not allow users to freely access data older than two weeks. Therefore, to stream a large amount of data, we spent several months collecting data from Twitter. In addition, the Twitter API requires some keys and tokens, which can be provided by creating a Twitter developer account and initiating a Twitter platform application. In this project, Tweepy [59] is used to collect data from Twitter. Tweepy is a Python package for accessing the Twitter API. The search keywords to gather Twitter data in this study are based on the brand name and generic name of 3 drugs as follows: Avastin (Bevacizumab), Eylea (Aflibercept), Lucentis (Ranibizumab), which are usually given as part of a combination of cancer medicine. Tweepy returns a JSON (JavaScript Object Notation) array with tweets expressed as JSON objects. These JSON objects contain the tweets and their attached metadata. As a result, 5,000 tweets were collected.

3.2.2 Pre-processing data

Cleansing data is an essential part of the data management process. By removing too much, we may lose valuable aspects, and by retaining too much, we may create a complex vocabulary. Several data cleaning steps have been performed to boost clarification. First, the original tweets are accompanied by some retweets (RT). When someone retweets, it shares another person's tweet. So, it is better to remove these retweets, since they are just repetitions of identical posts.

Second, there are many URLs in the context tweets that are uninformative for this study's topic, so they are omitted. For example, a review like *"eye infections appear linked to avastin injections medpage today. https://t.co/rn9OncU44V"* is changed to *"eye infections appear linked to avastin injections medpage today"*. Third, the extra whitespaces (more than one space between words) are removed, and one space is inserted between punctuation marks and words. Finally, only the tweets in English are retained, and the tweets in other languages are removed.

3.2.3 Data labelling for ADR extraction

The essential part of a supervised learning task like the task of this study is to have an adequately annotated dataset for training the model to implement. Once the data is cleaned up, three pharmacy students from Université de Montréal collaborating on this project have manually labeled a total of 5,000 user reviews from Twitter. To do the annotations, pharmacy students classified each word into four different categories: not important (O), Drug, Adverse Event, and Indication. Drug indications are reasons for using the drug, most commonly to treat illness or disease. For instance, insulin is indicated for treating diabetes. The indication class is added to reduce the possibility of false positives, for example, taking the drug for a headache rather than developing a headache due to the drug. The more precisely the dataset is labeled, the more accurate the result of the deep learning model for extraction of the different predefined classes will be, so labeling is one of the critical tasks in this study.

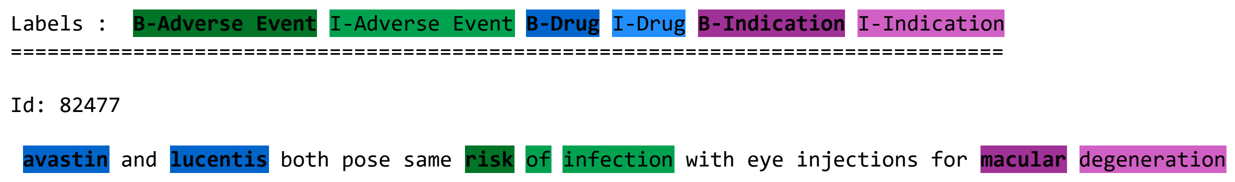
In the literature, there has been a variety of annotation schemes used such as IO, IOB or (BIO), IOE or (IEO), IOBES or (BIEO, OBIE), and some others.

- The IO model is not a common name for this model. It is just an optional name used in some research papers to explain the model. Each entity is represented only by one tag in this annotation system, which does not need any prefix. Each word in a sentence that belongs to pre-defined annotation categories is assigned with a proper tag, and O is for other ordinary words not contained in this category.
- The BOI model (or BIO) has been adopted by the Conference on Computational Natural Language Learning (CoNLL) [60]. It can denote the inside, outside, and the beginning of a chunk (chunks are made up of words) using B- and I- prefixes to represent each entity. The B- tag is used at the beginning of every chunk (i.e., all chunks start with the B- tag). The I- prefix before a tag indicates that the tag is inside a chunk. An O tag indicates that a token belongs to no chunk.
- The IOE model (or IEO) It is quite similar to BOI, with this difference that instead of indicating the end of an entity (E- tag), it detects its beginning.
- The IOBES model (or BIEO, OBIE) detects and tags the entity words as beginning (B), inside (I), end (E), and outside (O) of a named entity as well as single entities that consist of a single token and tags it by (S).

Between all mentioned schemes, although the IO scheme is the simplest and was used in studies that achieved the highest scores [61], it has a limitation, as it cannot correctly encode consecutive entities of the same type, and subsequent chunks of the same type cannot be distinguished from each other. So, the BOI model is selected. It is the “industry standard” encoding model, and when named entities of the same kind immediately follow each other, it enables us to locate the boundaries to have more accurate annotation. Therefore, it learns to distinguish seven different labels:

{ *B-Drug, I-Drug, B-Indication, I-Indication, B-Adverse Event, I- Adverse Event, and O*}

A typical labeled review is provided in Figure 3.2, each color indicates a specific label: dark blue: B-Drug, light blue: I-Drug, dark green: B-Adverse Event, light green: I-Adverse Event, dark purple: B-Indication, light purple: I-Indication, All other words: O.



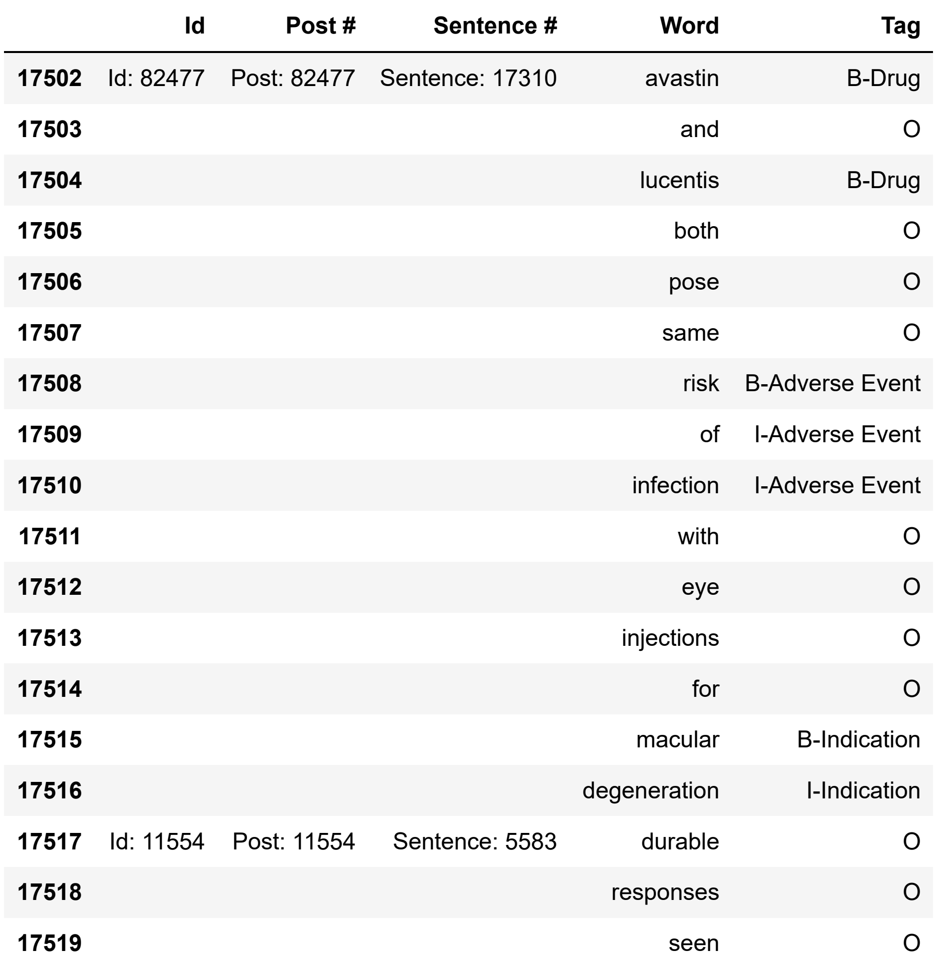
```
Labels :  B-Adverse Event I-Adverse Event B-Drug I-Drug B-Indication I-Indication
===================================================================================

Id: 82477

 avastin and lucentis both pose same risk of infection with eye injections for macular degeneration
```

Figure 3.2 An example of Named Entity Recognition (NER) labeling

For the implementation of manual labeling, a text annotation tool called Doccano is used. Doccano [62] is an open-source annotation program used by machine learning practitioners. The system offers features for text classification, sequence labeling, and sequence-to-sequence matching. The annotated result of this tool is a JSON file consisting of lines of information for each review. A typical output of Doccano is as follow:

{"id":188385,
"text": "avastin and lucentis both pose same risk of infection with eye injection for macular degenerat.",
"annotations":{"label":"Drug", "startoffset":0, "endoffset":6, "user":6},{"label":"Drug",
"startoffset":12, "endoffset":19, "user":6},{"label":"Adverse Event",
"startoffset":36, "endoffset":53, "user":6} ,{"label":"Indication",
"startoffset":78, "endoffset":97, "user":6}}

In the next step, the resulting JSON file is converted to the proper format for the named entity recognition model. The words of each review are tokenized based on the whitespaces and linked to their proper label. Finally, a CSV (comma-separated values) file is returned as a result. A typical result is reported in Figure 3.3.



	Id	Post #	Sentence #	Word	Tag
17502	Id: 82477	Post: 82477	Sentence: 17310	avastin	B-Drug
17503				and	O
17504				lucentis	B-Drug
17505				both	O
17506				pose	O
17507				same	O
17508				risk	B-Adverse Event
17509				of	I-Adverse Event
17510				infection	I-Adverse Event
17511				with	O
17512				eye	O
17513				injections	O
17514				for	O
17515				macular	B-Indication
17516				degeneration	I-Indication
17517	Id: 11554	Post: 11554	Sentence: 5583	durable	O
17518				responses	O
17519				seen	O

Figure 3.3 Samples of our annotated dataset for word-level adverse drug reaction extraction

Among all 5,000 samples available for this work, some of the user reviews consisted of advertisements or discussed the price of the drugs. Since the purpose of the study is extracting ADRs, only the samples that contain Adverse Event or Indication labels are kept, and the others are removed.

The result is a dataset that can be used to train and evaluate a model for automatically performing ADR extraction tasks. In total, the final dataset consists of 2735 sentences. In order to compare the different models to find the best one, the dataset is split into training, validation, and test sets. The training set consists of 55% of all the samples, validation 15%, and the test set consists of 30% of all the samples. Figure 3.4 and Table 3.1 show the distribution of the dataset in details.

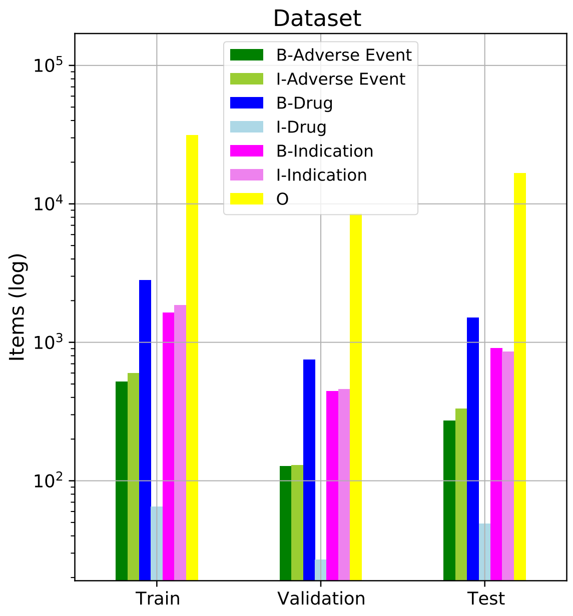


Figure 3.4 Dataset distribution for word-level extract adverse drug reaction task

Table 3.1 The number of instances in each class in the dataset

	B-Adverse E.	I-Adverse E.	B-Drug	I-Drug	B-Indic.	I-Indic.	O	Reviews
Train	518	596	2814	65	1631	1853	31388	1503
Valid.	127	130	751	27	445	457	8492	411
Test	272	331	1502	49	906	855	16709	821
All	917	1057	5067	141	2982	3165	56589	2735

3.3 Description of models used for ADR extraction

After preparing the dataset, a description of different approaches used for ADR extraction is given in this section. The experiments explored three approaches: a baseline deep learning model (BiLSTM-based), a transfer learning method where BERT is used, and a combination of these two approaches is used to explore whether it can achieve better results.

3.3.1 BiLSTM model, the study baseline model

A baseline model is important because it gives us something to compare to in evaluating the final model. Based on the methods found in the literature, the BiLSTM model appeared to be appropriate for that purpose. The BiLSTM model architecture consists of three main layers: the embedding layer, the BiLSTM layer, and the fully connected layer, as shown in Figure 3.5.

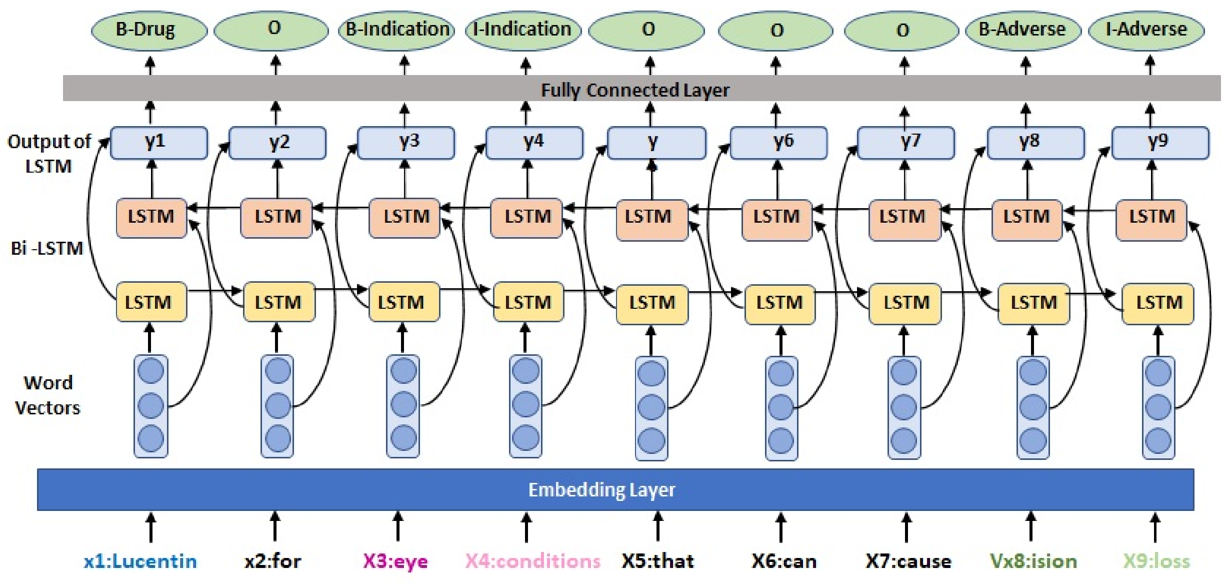


Figure 3.5 The architecture of the BiLSTM-based (Baseline) model

As described in Section 2.2.1 for the embedding layer, the number of embedding values must be determined in the first step. For that purpose, a vocabulary is created from all the unique words in the dataset and assigned unique indices (the size of the created vocabulary is 7907 in our dataset). Also, based on the Equation 2.2 explained in Section 2.2.1, the calculated embedding vector dimension for the embedding layer is 10 ($(7907)^{0.25} = 10$), but as suggested in [40], larger values such as 100, 200, and 512 will be tested as one of the hyperparameters of the model that can be changed to try obtaining better results.

3.3.2 Pre-trained BERT model

A pre-trained BERT model can be fine-tuned with additional output layers to create models for many natural language processing tasks, such as the named entity recognition. The BERT model that is adopted in this study comprises of 12 layers and 110 million parameters. The adopted BERT model architecture consists of three main parts: a BERT embedding layer, a BERT encoder layers, and a fully connected layer, as shown in Figure 3.6.

Similar to the previous model, two values must be determined to define the embedding layer: the number of embeddings and their respective dimension of embeddings. BERT has its predefined dictionary of size 30522, and the word embedding vector dimension is fixed to 768. The main difference between the BERT embedding layer and the embedding layer used in the BiLSTM model is that in the BiLSTM model, word embedding vectors are first initialized randomly and then trained during the training process. However, since the BERT

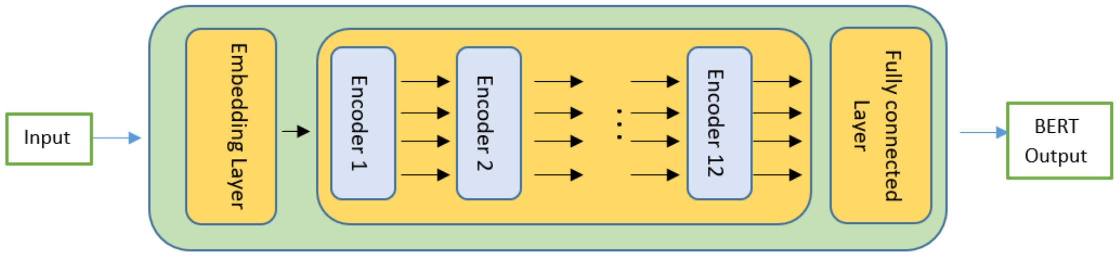


Figure 3.6 Architecture of the BERT model

model is pre-trained, these vectors contain proper values from the outset, and by fine-tuning, these values will be improved.

It is possible to access and use the output of each of these 12 BERT encoder layers separately. However, inspired by [22] to explore the idea of achieving a better score for the ADR extraction task, instead of using the last encoder layer output, some experiments are applied by creating different combinations of BERT hidden layers as follows: concatenation of last 3 hidden layers, concatenation of last 4 hidden layers, concatenation of last 5 hidden layers, summation of the values of last 3 hidden layers, summation of the values of last 4 hidden layers and summation of the values of last 5 hidden layers. These concatenations will change the dimension of BERT word embedding. For example, for 4 concatenated layers, the 768 layer size will be changed to $4 \times 768 = 3072$. After trying these models, the one that produces the best result is chosen as our BERT-based model for the following experiments.

3.3.3 Combining the BERT and BiLSTM models

In the final step, a combined model from BERT and BiLSTM is created to check the possibility of achieving a better score than the previously selected best BERT model. The final architecture is as follows: the selected BERT model is combined with a BiLSTM layer and a fully connected layer as shown in (Figure 3.7). The idea behind this combination is that both models have bidirectional word awareness. In addition, the BiLSTM layer continues to extract the features received from BERT and improves the word representation vectors.

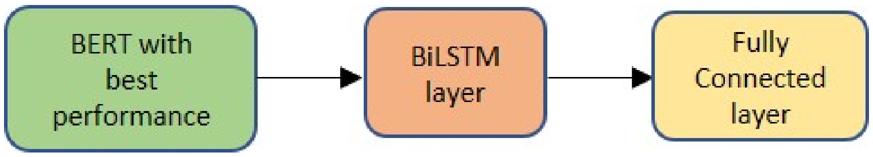
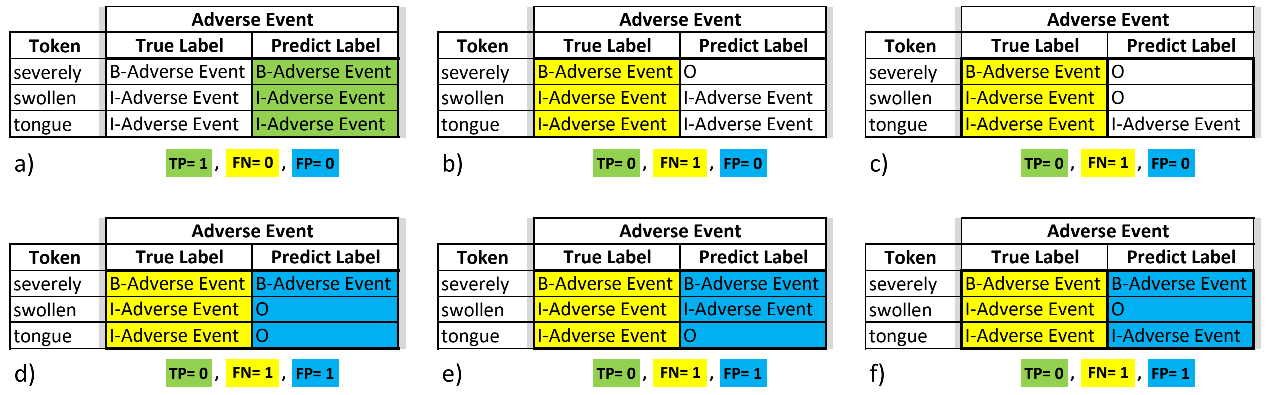


Figure 3.7 The final model which is created by combining the best BERT and BiLSTM models from previous experiments

3.4 Evaluation metrics for word-level task of ADR extraction

Since the classes in the dataset are imbalanced, this study will evaluate the performance of the models with three performance measures, precision, recall, and F1-score. The O label dominates the dataset; O is omitted from the evaluation to improve the dataset analysis and the models comparison. The scores for each class are calculated and shown separately to provide a more accurate view of performance. For the average illustration, the micro average (explained in Chapter 2.5) is used. As mentioned in Section 3.1.3, the BOI labeling scheme is used to annotate the dataset since considering tokens separately will not return valuable information, so the CoNLL evaluation script (called "exact-match evaluation") described in Section 2.5 is selected for evaluating the performance of the study. Evaluation of the named entity recognition task is complex, and the related resources are limited. Therefore, some simple examples will show the concept of TP, FN, and FP (explained in Section 2.5) for named entity recognition, BOI is adopted for labeling and exact-match evaluation. An example is developed with the entity term of " *severely swollen tongue* ", which is an Adverse Event. This example lead to several possible predictions, that illustrate the concept of TP, FN, and FP (see Figure 3.8, green highlights flag TP, yellow highlights flag FN, and blue highlights flag FP).



a)

	Adverse Event	
Token	True Label	Predict Label
severely	B-Adverse Event	B-Adverse Event
swollen	I-Adverse Event	I-Adverse Event
tongue	I-Adverse Event	I-Adverse Event

TP= 1 , FN= 0 , FP= 0

b)

	Adverse Event	
Token	True Label	Predict Label
severely	B-Adverse Event	O
swollen	I-Adverse Event	I-Adverse Event
tongue	I-Adverse Event	I-Adverse Event

TP= 0 , FN= 1 , FP= 0

c)

	Adverse Event	
Token	True Label	Predict Label
severely	B-Adverse Event	O
swollen	I-Adverse Event	O
tongue	I-Adverse Event	I-Adverse Event

TP= 0 , FN= 1 , FP= 0

d)

	Adverse Event	
Token	True Label	Predict Label
severely	B-Adverse Event	B-Adverse Event
swollen	I-Adverse Event	O
tongue	I-Adverse Event	O

TP= 0 , FN= 1 , FP= 1

e)

	Adverse Event	
Token	True Label	Predict Label
severely	B-Adverse Event	B-Adverse Event
swollen	I-Adverse Event	I-Adverse Event
tongue	I-Adverse Event	O

TP= 0 , FN= 1 , FP= 1

f)

	Adverse Event	
Token	True Label	Predict Label
severely	B-Adverse Event	B-Adverse Event
swollen	I-Adverse Event	O
tongue	I-Adverse Event	I-Adverse Event

TP= 0 , FN= 1 , FP= 1

Figure 3.8 Illustrative example of TP, FN and FP predictions

The first columns show the tokenized format of the entity term, and the second and third columns consist of the true labels (target labels or ground truth) and model-predicted labels, respectively. In this example, the values of TP, FN, and FP are computed for the Adverse Event label so that the Adverse Event will be considered as a positive label, and any other predicted label would be considered as a negative label (like "O" in this example). In Figure 3.8 (a), the true labels and predicted labels are exactly the same (highlighted by green), to be more specific, the type of the prediction (Adverse Event) and the boundary of prediction (Start by B-Adverse Event and end by I-Adverse Event) are exactly the same as the true label, so the result is $TP = 1$. As shown in Figures 3.8 (b) and (c), the true labels and predicted labels do not exactly match. The true labels are B-Adverse Event and I-Adverse Event, but the model predicted them as O, which is considered FN, highlighted by yellow. In Figure 3.8 (d), (e) and (f), the FN is equal to 1, for the same reason as (b) and (c). Moreover, since there is a prediction as B-Adverse Event, but the other parts of the entity are not predicted correctly, the FP is equal to 1.

In a second example, in Figures 3.9, the described concepts are shown in a case producing different labels. As the previous figure, the first column shows some test dataset reviews' tokenized format. The second and third columns consist of true labels (target labels) and model-predicted labels. Green highlights show the TP, yellow highlights show FN, and blue highlights show FP predicted for each related label in its respective column. As shown in the figure, in this part (second and third columns), the values of TP, FN, and FP for the Adverse Event label are computed so that the Adverse Event will be considered as a positive label, and all other labels (Drug, Indication, and O) will be considered as negative labels. In the case of two terms *" severely swollen tongue "* and *" minor blood nose "* in the first column, the true labels and predict labels are exactly the same, so the number of TP for the Adverse Event is 2. In the term *" Wound healing "*, the true label for *healing* is I-Adverse Event, but the predicted model predicts it as O, so the number of FN for the Adverse Event label is 1. For *chemo* word and the term *" Wound healing "*, the model predicts it differs from the true labels, hence the FP is 2.

In the next part (fourth and fifth columns), the computation of TP, FN, and FP for the Drug label is explained. In this example, the Drug label will be considered as a positive label, and all other labels (Adverse Event, Indication, and O) will be considered as negative labels. For words *avastin* (2 times), *biosimillar avastin* , and *biosimillar lucentin* the model predicts labels completely correctly, so the TP is 4. In the case of word *lucentis* , the true label is B-Drug, but the model predicts it as O, so the FN is 1. In the case of two words *treatment* and *works*, the true labels are O, but the model predicts them as B-Drug wrongly, so the FP is 2. The process is the same for the Indication label in the sixth and seventh columns.

Token	True Label	Predict Label	True Label	Predict Label	True Label	Predict Label
I	O	O	O	O	O	O
get	O	O	O	O	O	O
a	O	O	O	O	O	O
severely	B-Adverse Event	B-Adverse Event	B-Adverse Event	B-Adverse Event	B-Adverse Event	B-Adverse Event
swollen	I-Adverse Event	I-Adverse Event	I-Adverse Event	I-Adverse Event	I-Adverse Event	I-Adverse Event
tongue	I-Adverse Event	I-Adverse Event	I-Adverse Event	I-Adverse Event	I-Adverse Event	I-Adverse Event
about	O	O	O	O	O	O
1	O	O	O	O	O	O
week	O	O	O	O	O	O
affter	O	O	O	O	O	O
treatment	O	B-Drug	O	B-Drug	O	B-Drug
with	O	O	O	O	O	O
avastin	B-Drug	B-Drug	B-Drug	B-Drug	B-Drug	B-Drug
.	O	O	O	O	O	O
biosimilar	B-Drug	B-Drug	B-Drug	B-Drug	B-Drug	B-Drug
avastin	I-Drug	I-Drug	I-Drug	I-Drug	I-Drug	I-Drug
gives	O	O	O	O	O	O
me	O	O	O	O	O	O
a	O	O	O	O	O	O
minor	B-Adverse Event	B-Adverse Event	B-Adverse Event	B-Adverse Event	B-Adverse Event	B-Adverse Event
blood	I-Adverse Event	I-Adverse Event	I-Adverse Event	I-Adverse Event	I-Adverse Event	I-Adverse Event
nose	I-Adverse Event	I-Adverse Event	I-Adverse Event	I-Adverse Event	I-Adverse Event	I-Adverse Event
.	O	O	O	O	O	O
no	O	O	O	O	O	O
avastin	B-Drug	B-Drug	B-Drug	B-Drug	B-Drug	B-Drug
for	O	O	O	O	O	O
chemo	O	B-Adverse Event	O	B-Adverse Event	O	B-Adverse Event
round	O	O	O	O	O	O
as	O	O	O	O	O	O
it	O	O	O	O	O	O
interferes	O	B-Indication	O	B-Indication	O	B-Indication
with	O	O	O	O	O	O
wound	B-Adverse Event	B-Adverse Event	B-Adverse Event	B-Adverse Event	B-Adverse Event	B-Adverse Event
healing	I-Adverse Event	O	I-Adverse Event	O	I-Adverse Event	O
.	O	O	O	O	O	O
lucentis	B-Drug	O	B-Drug	O	B-Drug	O
works	O	B-Drug	O	B-Drug	O	B-Drug
for	O	O	O	O	O	O
me	O	O	O	O	O	O
on	O	O	O	O	O	O
diabetic	B-Indication	B-Indication	B-Indication	B-Indication	B-Indication	B-Indication
eye	I-Indication	I-Indication	I-Indication	I-Indication	I-Indication	I-Indication
disease	I-Indication	O	I-Indication	O	I-Indication	O
.	O	O	O	O	O	O
my	O	O	O	O	O	O
father	O	O	O	O	O	O
in	O	B-Indication	O	B-Indication	O	B-Indication
under	O	O	O	O	O	O
biosimilar	B-Drug	B-Drug	B-Drug	B-Drug	B-Drug	B-Drug
lucentis	I-Drug	I-Drug	I-Drug	I-Drug	I-Drug	I-Drug
treatment	O	O	O	O	O	O
for	O	O	O	O	O	O
wet	B-Indication	B-Indication	B-Indication	B-Indication	B-Indication	B-Indication
amd	I-Indication	I-Indication	I-Indication	I-Indication	I-Indication	I-Indication
.	O	O	O	O	O	O
	Adverse Event		Drug		Indication	
	Items :	3	Items :	5	Items :	2
	TP :	2	TP :	4	TP :	1
	FN :	1	FN :	1	FN :	1
	FP :	2	FP :	2	FP :	3

Figure 3.9 A practical example illustrating the concepts of TP, FN, and FP in NER task evaluation for complete sentences of reviews

3.5 Implementation and hyperparameter optimization

This section will present data, hyperparameters, and other configurations necessary to run the experiments. The implementation of the model is done in Python. The experimental codes are implemented in Python (v.3.7.6) by using the HuggingFace Transformers package [63] with Pytorch(1.2.0) and some other libraries such as Keras, Sklearn, Seqval. Computations were performed locally on a GeForce GTX 1650 GPU.

To perform the experiments, the dataset is split into train, validation, and test set. The training+validation set consists of 70% of all samples, and the test set consists of 30% of all samples (Figure 3.4). The most frequently suggested data partitioning in the literature is 80% for training+validation data and 20% for testing data. However, since the dataset of this study is small (2735 instances), 30% of all data is considered for the test dataset to have more reliable evaluation scores. Considering higher than 30% of the entire dataset for the test dataset will lead to having less training data for training and lower performance.

On the other hand, with less testing data that use less than 30% of the dataset, the performance score (F1-score) will have greater variance. For example, one wrong or correct prediction will change the F1-score by 1% or more. Some statistical examples describe this concept in appendix A. The validation set is used to find the optimal hyperparameters, check the model during training epochs to prevent overfitting, and select the best model parameters, i.e., those that maximize the validation F1-score. Finally, the test dataset, which is entirely unseen data of the dataset, is used to evaluate and assess the performance of the models in their ability to generalize .

Before carrying out the experiments and receiving final results to compare the models, it is essential to set the hyperparameters of the training process. In the hyperparameters optimization process, each possible combination of all hyperparameters with different values is provided to find the optimal set of hyperparameters. For example, if there are two hyperparameters such as learning rate and batch size, and if for the first hyperparameter, a set of 2 values like {0.005,0.001} each chosen and the second one, a set of 3 values like {16,32,64} is considered, it will lead to 6 different combination sets of hyperparameters such as {0.005,0.001,16,32,64}. The different values of the sets are selected based on literature and initial testing in this project. Finally, all these combinations are applied to the models, and the combination that returns the best F1-score value with the validation dataset is selected as the optimal hyperparameters set. Since the study dataset is limited, to ensure that the experiment results for each set of hyperparameters are reliable, the K-fold cross-validation method is used for evaluation, which means that the implementation of each hyperparameter set is tested K times instead of one. To apply the K-fold cross-validation in this study, the training dataset is divided into K=5 equal datasets. In each experiment, one of these K datasets is considered as a validation dataset, and all the other K-1 datasets is considered as a training set. The hyperparameter sets for the three different models of the study mentioned in Section 3.2 (BiLSTM, BERT, BERT+BiLSTM) are explained as bellow.

For the BiLSTM model, the list of hyperparameters is as follows: learning rate, batches size, dropout, number of hidden units, and embedding layer dimension. The learning rate is one of the essential training hyperparameters, and the used set of values is considered {0.005, 0.001, 0.0005}. The batch size is also critical as it determines how many samples the model sees each time that its weights are updated. Batch size values of 16, 32, and 64 are tested. The large batch sizes need more memory space. Since the average value of the loss in each epoch is used to optimize the model's weights, it takes longer time to get a good training with the chosen value. In contrast, the model is not guaranteed to converge to the global optimal when using the smallest batch size. Dropout is another hyperparameter that is used to combat overfitting, and the set values tested contained is {0.2, 0.3 }. In the LSTM models, there is usually no rule for the number of hidden units to use. To guide us in the selection of suitable values, some numbers found in literature such as {150, 200, 256} were tested. In the case of embedding layer dimension, as mentioned in Section 3.2.1, the set of tested value is {10, 100, 256, 512}. Finally, an important parameter in the NLP tasks is the sequence length for the number of words per sentence. Due to varying lengths of user reviews, a sequence length is set for the model. The sentences shorter than the sequence value are padded to the right with a unique padding token, and each review with the extra length is truncated. The sequence length is set based on the maximum sequence length of the sentences in the dataset or a sequence length value that does not result in many sentences getting truncated. Based on the length of sentences in the available dataset, a value of 100 is selected.

For different BERT-based models, the list of hyperparameters and their set of values are as follows: learning rate={0.005, 0.0005, 0.0001}, batches size={16,32,64}, dropout={0.2,0.3,0.4}, sequence length=120, because of the word-pieces method of tokenization in the BERT model, 120 is the value of its sequence length. Finally, for combined model of BERT and BiLSTM, the list of hyperparameters and their set of values are as follows: learning rate={0.005, 0.0005, 0.0001}, batches size={16,32}, dropout={0.2,0.3}, embedding layer dimension={200,256}, sequence length=120.

The early-stopping rule is used to avoid overfitting by calculating a loss function value for every epoch in the validation dataset. The training is stopped when the validation loss reaches a minimum; however, a patience parameter of 5, the number of epochs to wait before the early stop, is imposed in our early stopping to allow for a local minimum. Thus, if the loss function does not improve for five epochs, then the model stops training. Moreover, to identify the best model for our experiments, all the model parameters are saved at the epoch where the model reaches a maximum F1-score in the validation set.

3.6 ADR extraction experiment results and discussions

Several experiments we conducted to find the optimal hyperparameters based on the different combinations of hyperparameters mentioned in Section 3.4. First, for the BiLSTM model, the model got the best F1-score in the validation set for a learning rate of 0.005, a batch size of 16, a dropout of 0.2, hidden units of 256, an embedding layer dimension of 512, and a maximum sequence length of 100. Then, for different BERT-based models, the optimal hyperparameters are achieved. For example, the BERT model with 4 last hidden layers concatenation got a learning rate of 0.0005, batch size of 16, dropout of 0.2. Finally, in the case of the combined model of BERT (with 4 last hidden layers concatenation) and BiLSTM, the optimal hyperparameters are found to be, learning rate=0.0005, batch size=16, hidden units=200, and dropout=0.3. The results of all experiments mentioned above can be found in appendix B.

After finding the optimal hyperparameters, the final training and testing experiments are performed with the BiLSTM model, different BERT-based models, and the final combined ($BERT + BiLSTM$) model to find the best model by comparing their results. As mentioned before, the performance of each model is evaluated and presented according to three performance measures, precision, recall, F1-score, and since the "O" label dominates the dataset, "O" is omitted from the evaluation to improve the dataset analysis. The Algorithm 1 shows briefly the experimental procedure to extract adverse drug reactions. In the next section, the results of the experiments are shown and followed by a related discussions.

Algorithm 1: ADR Extraction Procedure

```
input  : Prepared dataset based on Twitter user reviews
output: Model predicted label for each review at word-level
1. read the prepared dataset
2. shuffle the dataset
3. split the dataset to train,validation and test dataset
4. build the model ;        ▷ BiLSTM model or BERT model or BERT+BiLSTM model
5. set the hyperparameter values (batch size, learning rate, maximum
   length, ...)
6. make batches of train, validation and test dataset
7. while termination condition is not satisfied do
    7.1 model.train() ;        ▷ model is in training mode
    for each batch in (train batches) do
        - outputs=model(train batch) ;   ▷ Feed forward:pass each train batch into the
          model and calculate the outputs
        - loss=loss_function(outputs,true label) ;   ▷ compute training loss by using
          cross_entropy function
        -loss.backward ;   ▷ backpropagation:apply backpropagation by calculating the
          gradient of the loss function with respect to parameters (minimizing the
          error)
        - optimizer.step() ;   ▷ update parameters (weights) of the model by using Adam as
          optimizer function
    end
    7.2 training loss=mean(training batches loss) ;   ▷ average the training losses for
      all batches
    7.3 model.eval() ;        ▷ model is in validation mode
    for each batch in (validation batches) do
        - outputs=model(validation batch) ;   ▷ Feed forward:pass each validation batch
          into the model
        - loss=loss_function(outputs,true label) ;   ▷ compute validation loss using
          cross_entropy function
    end
    7.4 validation loss=mean(validation batches loss) ;   ▷ average the validation losses
      for all batches
    if validation F1_score greater than previous maximum validation F1_score then
        - save model parameters(weights)
    end
end
8. outputs=trained model(test dataset) ;   ▷ Calculate the output value for the
   test dataset with the trained model
9. predict=label-lists[argmax(outputs)] ;   ▷ argmax returns the index of the
   predicted label based on the maximum value among the predicted outputs
10. evaluate the model predictions by F1_score,precision and recall
```

Baseline (BiLSTM) model experiment results:

The architecture of the BiLSTM model is expressed in Algorithm 2. This algorithm is a detailed explanation of step 4 in Algorithm 1 for the BiLSTM_based method.

```
Algorithm 2: Building BiLSTM model for ADR extraction
input         : pre-processed data
                                    ▷ input_dim:[batch size x seq_length]
output        : model output for each review at word-level
InitioalValues: batch size=16, seq_length=100, embedding_dim=512, dropout=0.2,
                hidden_layer_number=256, label_number = 7
1. emb-out=Embedding (input) ;     ▷ (convert each token to a word embedding
 vector) emb-out_dim:[batch size x seq_length x embedding_dim]
2. drop-out=Dropout (emb-out,dropout) ;     ▷ drop-out_dim:[batch size x
 seq_length x embedding_dim]
3. lstm-out=LSTM (drop-out,bidirectional=True); ▷ lstm-out_dim:[batch size x
 seq_length x 2*hidden_layer_number]
4. outputs=Linear (lstm-out,label_number) ;     ▷ outputs_dim:[batch size x
 seq_length x label_number]
```

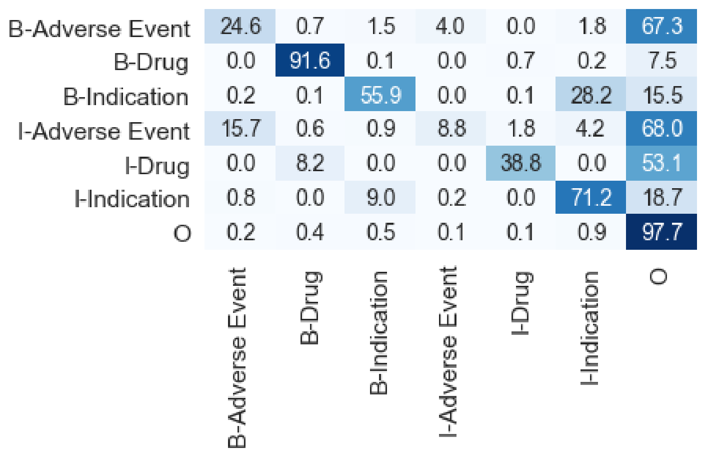
The BiLSTM model with the optimal hyperparameters obtained from previous section's experiments is trained and applied to the test dataset. The results are reported in Table 3.2.

Table 3.2 Test result of BiLSTM model

	Precision(%)	Recall(%)	F1-score(%)	Support
Adverse Event	31.35	19.12	23.80	272
Drug	93.54	90.73	92.11	1502
Indication	63.03	47.35	54.07	906
Micro avg	80.07	68.79	74.00	2680

As shown in the table 3.2, the BiLSTM model gives precision, recall, and F1-score, all higher than 90% in predicting Drug labels, which are good scores. However, the Adverse Event label prediction results all are less than 30%, which are low scores. The main reason is the low number of training examples for the Adverse Event label compared to the Drug label. These results will be further explored in the discussion section.

In the confusion matrix, the summary of the correct and incorrect predictions of each label relative to the other labels is shown in Figure 3.10. All of the correct predictions are on the diagonal, and the prediction errors are outside the diagonal. Here, for example, the prediction percentages for the B-Adverse Event label are as follows: the B-Adverse Event label is predicted correctly 24.6 % of the time (True positive), and the model mispredicted B-Adverse Event as I-Averse Event 4.0% of the time, while it is predicted as O, 67.3 % of the time (False negative). Moreover, the I-Adverse Event is predicted as a B-adverse Event 15.7% of the time (False positive). As the confusion matrix shows for all the labels, the dominant misprediction is the "O" label. The model has trouble distinguishing the B-Indication label from the I-Indication 28.2% of the time, which is a high misprediction percentage.

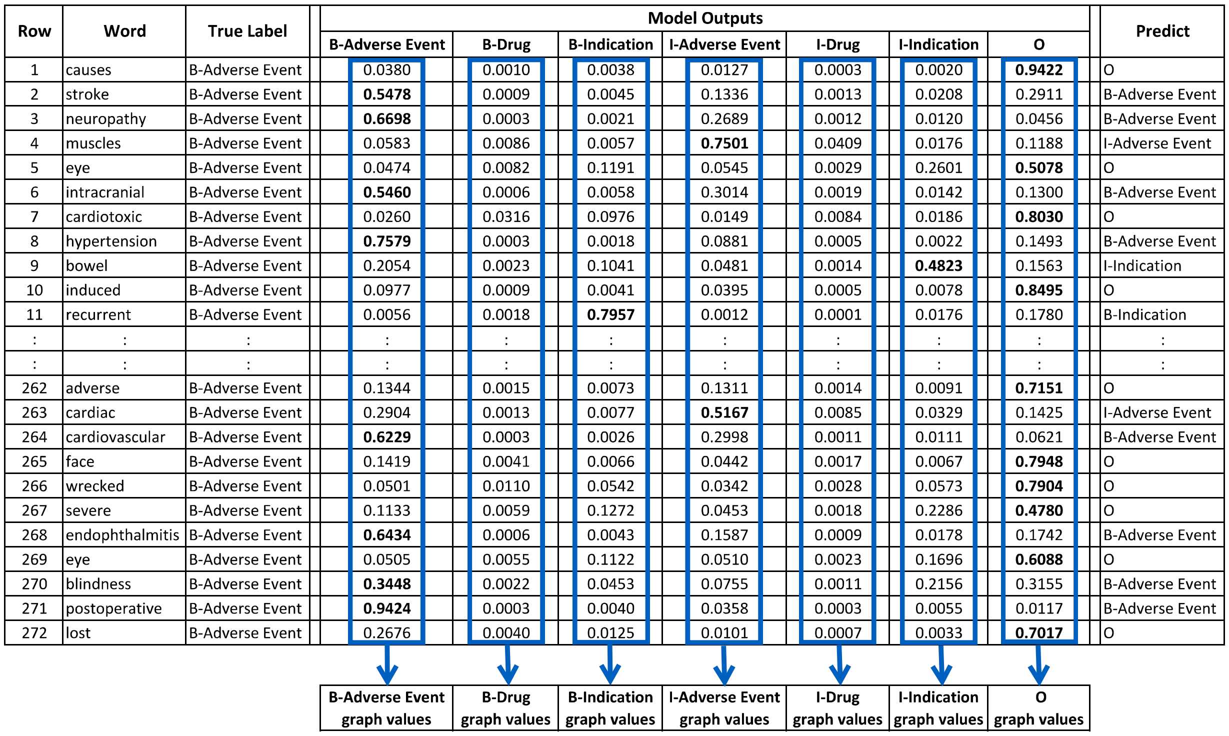


	B-Adverse Event	B-Drug	B-Indication	I-Adverse Event	I-Drug	I-Indication	O
B-Adverse Event	24.6	0.7	1.5	4.0	0.0	1.8	67.3
B-Drug	0.0	91.6	0.1	0.0	0.7	0.2	7.5
B-Indication	0.2	0.1	55.9	0.0	0.1	28.2	15.5
I-Adverse Event	15.7	0.6	0.9	8.8	1.8	4.2	68.0
I-Drug	0.0	8.2	0.0	0.0	38.8	0.0	53.1
I-Indication	0.8	0.0	9.0	0.2	0.0	71.2	18.7
O	0.2	0.4	0.5	0.1	0.1	0.9	97.7

Figure 3.10 The BiLSTM model test result confusion matrix

To better understand the performance of the model, its output will be investigated by creating and representing the density plot of the labels. As mentioned before in the model structure section, for each word in the input sequence, the model produces 7 different values (7 is the number of labels considered in the study) at the output layer. The label related to the output with the highest value will be considered as the model's prediction. In the following, the B-Adverse Event label will be explored as an example.

In the first step, all the output values for the words with the true label of B-Adverse Event in the test dataset will be gathered. As it can be seen in Figure 3.11, the BiLSTM model produces 7 probability values associated with each label (B-Adverse Event, B-Drug, B-Indication, I-Adverse Event, I-Drug, I-Indication, and O) as the output for each word.



Row	Word	True Label	Model Outputs							Predict
			B-Adverse Event	B-Drug	B-Indication	I-Adverse Event	I-Drug	I-Indication	O	
1	causes	B-Adverse Event	0.0380	0.0010	0.0038	0.0127	0.0003	0.0020	**0.9422**	O
2	stroke	B-Adverse Event	**0.5478**	0.0009	0.0045	0.1336	0.0013	0.0208	0.2911	B-Adverse Event
3	neuropathy	B-Adverse Event	**0.6698**	0.0003	0.0021	0.2689	0.0012	0.0120	0.0456	B-Adverse Event
4	muscles	B-Adverse Event	0.0583	0.0086	0.0057	**0.7501**	0.0409	0.0176	0.1188	I-Adverse Event
5	eye	B-Adverse Event	0.0474	0.0082	0.1191	0.0545	0.0029	0.2601	**0.5078**	O
6	intracranial	B-Adverse Event	**0.5460**	0.0006	0.0058	0.3014	0.0019	0.0142	0.1300	B-Adverse Event
7	cardiotoxic	B-Adverse Event	0.0260	0.0316	0.0976	0.0149	0.0084	0.0186	**0.8030**	O
8	hypertension	B-Adverse Event	**0.7579**	0.0003	0.0018	0.0881	0.0005	0.0022	0.1493	B-Adverse Event
9	bowel	B-Adverse Event	0.2054	0.0023	0.1041	0.0481	0.0014	**0.4823**	0.1563	I-Indication
10	induced	B-Adverse Event	0.0977	0.0009	0.0041	0.0395	0.0005	0.0078	**0.8495**	O
11	recurrent	B-Adverse Event	0.0056	0.0018	**0.7957**	0.0012	0.0001	0.0176	0.1780	B-Indication
:	:	:	:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:	:	:	:
262	adverse	B-Adverse Event	0.1344	0.0015	0.0073	0.1311	0.0014	0.0091	**0.7151**	O
263	cardiac	B-Adverse Event	0.2904	0.0013	0.0077	**0.5167**	0.0085	0.0329	0.1425	I-Adverse Event
264	cardiovascular	B-Adverse Event	**0.6229**	0.0003	0.0026	0.2998	0.0011	0.0111	0.0621	B-Adverse Event
265	face	B-Adverse Event	0.1419	0.0041	0.0066	0.0442	0.0017	0.0067	**0.7948**	O
266	wrecked	B-Adverse Event	0.0501	0.0110	0.0542	0.0342	0.0028	0.0573	**0.7904**	O
267	severe	B-Adverse Event	0.1133	0.0059	0.1272	0.0453	0.0018	0.2286	**0.4780**	O
268	endophthalmitis	B-Adverse Event	**0.6434**	0.0006	0.0043	0.1587	0.0009	0.0178	0.1742	B-Adverse Event
269	eye	B-Adverse Event	0.0505	0.0055	0.1122	0.0510	0.0023	0.1696	**0.6088**	O
270	blindness	B-Adverse Event	**0.3448**	0.0022	0.0453	0.0755	0.0011	0.2156	0.3155	B-Adverse Event
271	postoperative	B-Adverse Event	**0.9424**	0.0003	0.0040	0.0358	0.0003	0.0055	0.0117	B-Adverse Event
272	lost	B-Adverse Event	0.2676	0.0040	0.0125	0.0101	0.0007	0.0033	**0.7017**	O

B-Adverse Event graph values	B-Drug graph values	B-Indication graph values	I-Adverse Event graph values	I-Drug graph values	I-Indication graph values	O graph values

Figure 3.11 The BiLSTM model output value probabilities for the words with "B-Adverse Event" true label in the test dataset

In the next step, the distribution of the probability values in each table column will be plotted and shown as the density plot. These steps are done for all 6 labels, and the results are shown in Figure 3.12. In all plots, the red-filled plot shows the main label (the true label that is investigated), and other labels are presented with different colors. The distribution plot with higher peaks and more skewed to the right indicates high probability values for a large number of words. Thus, the more the main label plot (red-filled plot) is skewed to the right, and the other labels' plots are skewed to the left, the less overlap there is with the red-filled plot, which indicates more accurate and more certain predictions for the words with this label.

For instance, in Figure 3.12 (a) the main label is B-Adverse Event, based on the previous explanation and the distribution plot of the O label and its overlap with the B-Adverse Event label, most of the words which must be predicted as B-Adverse Event will be predicted as O label. Figure 3.12 (b) shows that the model will have an accurate and certain prediction for the B-Drug, and a few of the B-Drug labels will be mispredicted as O labels. Moreover, based on the small overlap between the B-Drug plot and I-Indication, very few B-Drug labels will be predicted as I-Indication. In Figure 3.12 (c), it can be seen that there will be some

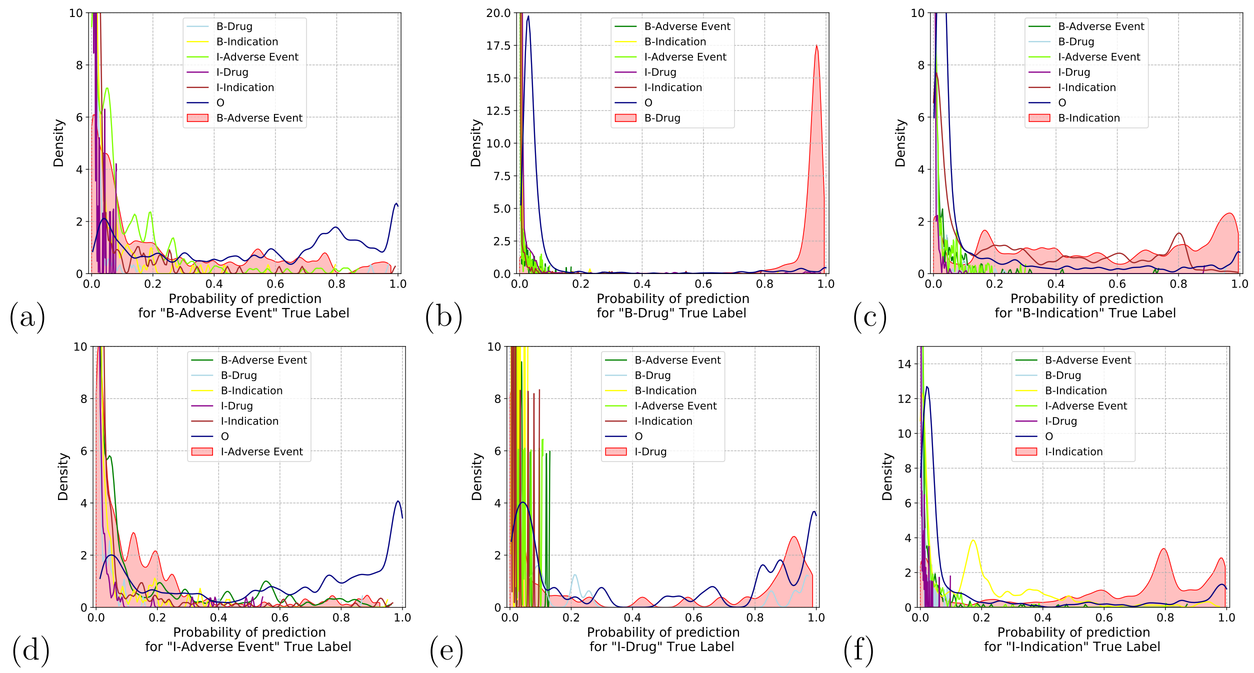


Figure 3.12 BiLSTM model density plot for all labels

misprediction of the main label (B-Indication) by considering some overlap between the B-Indication label and the I-Indication and O labels. Moreover, the distribution of probabilities for the B-Indication label is better than the B-Adverse Event label in Figure 3.12 (a) since it is skewed to the right. Therefore the BiLSTM model will have better prediction score for the B-Indication label. The advantage of exploring density plots in addition to exploring the confusion matrix is that the confusion matrix indicates the additional percentages for correct and incorrect predictions of the model, but in the density plots, the certainty of the prediction is observable as well. It can be seen that the predictions are performed based on the high probability or low probability. The more the distribution plot is skewed to the right and the larger the probability values, the more confident the correct prediction will be, which leads to more confidence in the score results of the model.

BERT models experiment results:

Based on the optimal hyperparameters obtained previously, several experiments were performed, such as BERT first hidden layer, BERT last hidden layer, and dierent combinations of BERT hidden layers: concatenation of the last 3 hidden layers, concatenation of the last 4 hidden layers, and concatenation of the last 5 hidden layers as well as the summation of the values of the last 3 hidden layers, summation of the values of the last 4 hidden layers, and summation of the values of the last 5 hidden layers. The architecture of BERT with 4 last hidden layers concatenation is shown in the Algorithm 3 as an example.

Algorithm 3: Building BERT model (concatenation of 4 last layers) for ADR extraction

```
input        : pre-processed data
                                  ▷ input_dim:[batch size x seq_length]
output       : model output for each review at word-level
InitioalValues: batch size=16, seq_length=120, dropout=0.2, label_number = 7
1. hidden layers=BERT.model (input) ;   ▷ (return the output of all 12 encoder
   of BERT model) hidden layers_dim:[batch size x seq_length x 768]
2. bert-out=Concat (4 last hidden layers) ;   ▷ (concatenation of 4 last layers)
   bert-out_dim:[batch size x seq_length x 3072]
3. drop-out=Dropout (bert-out,dropout)
   ;                         ▷ drop-out_dim:[batch size x seq_length x 3072]
4. outputs=Linear (drop-out,label_number);   ▷ outputs_dim:[batch size x
   seq_length x label_number]
```

The goal is to find a BERT model with a performance better than the original BERT. By comparing the results in Figure 3.13 sorted from the worse to the best Micro average, it is clear that the BERT model concatenating its 4 last layers achieves the best F1-score.

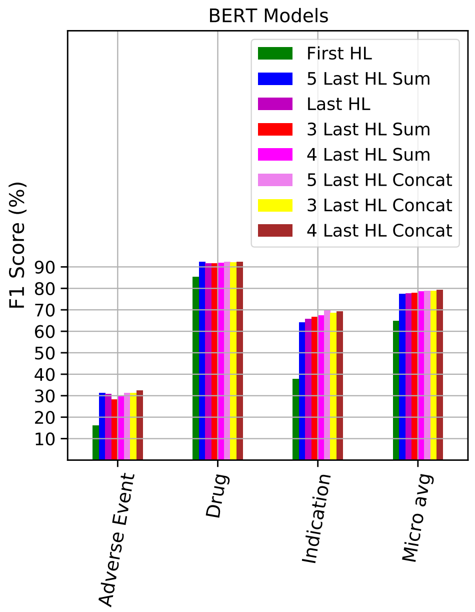


Figure 3.13 Results of different BERT model, HL: hidden layer, Sum: summation, Concat: Concatenation

The detailed results of the BERT model with 4 last hidden layers concatenation are shown in Table 3.3. The Drug label has the highest score, and the model F1-score is 79.37 %.

Table 3.3 Test result of BERT model with 4 last hidden layers concatenation

	Precision(%)	Recall(%)	F1-score(%)	Support
Adverse Event	38.97	27.94	32.55	272
Drug	92.25	92.68	92.46	1502
Indication	71.97	66.89	69.34	906
Micro avg	81.46	77.39	79.37	2680

Exploring the confusion matrix for the BERT model with the 4 last layer concatenated shows that like BiLSTM, the most mispredicted label is the O label with a high percentage such as 49.6%, 48.0%, and 61.2% which is reduced compared to the BiLSTM misprediction percentages. In addition, two labels, I-Adverse Event and I-Drug, are mispredicted as B-Adverse Event and B-Drug by 10.0% and 18.4% respectively, which represent high mispredictions.

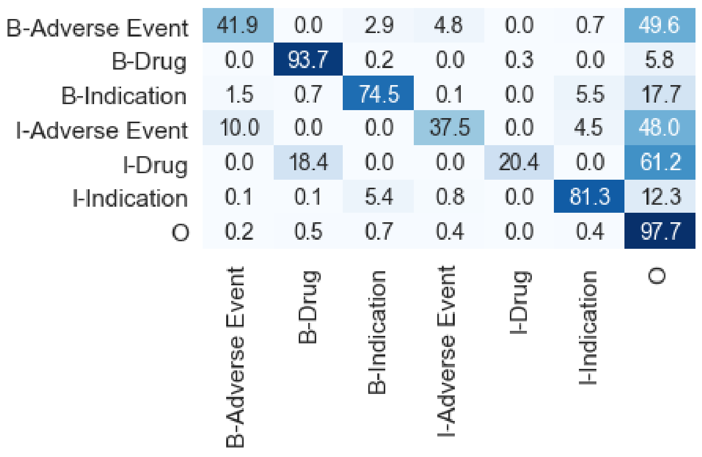
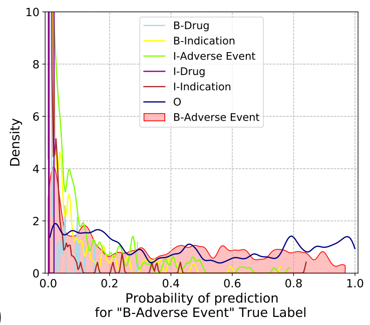
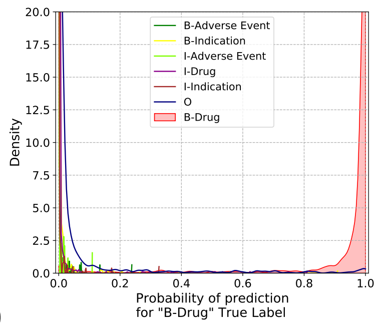


Figure 3.14 The BERT model with 4 last layer concatenation confusion matrix

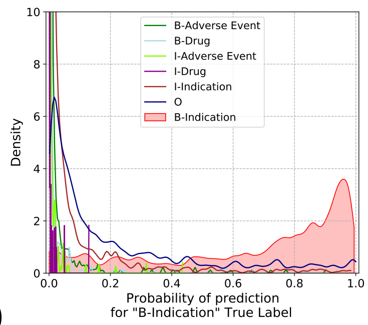
Figure 3.15 shows the output density plots of labels for the BERT model with 4 last hidden layers. Based on the shape of the main label density plots and their lesser overlap with other plots, it is clear that the B-Drug, the B-Indication, and the I-Indication labels are more accurate and produce more certain predictions compared to other labels, and we believe the main reason for this is the higher number of examples for these labels in the dataset.



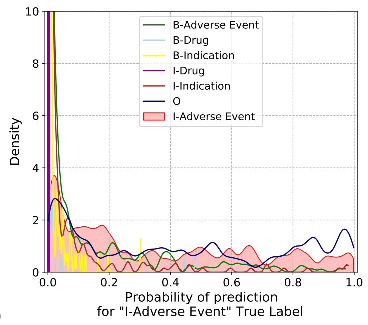
(a)



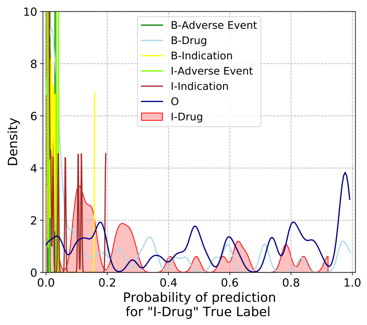
(b)



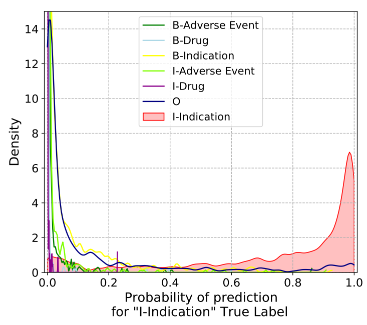
(c)



(d)



(e)



(f)

Figure 3.15 BERT with 4 last hidden layers concatenation model density plot for all labels

Experimental results with the combined BERT and BiLSTM model:

In the final experiments, a combination of the best BERT (Con 4 Last layer) and BiLSTM models with the architecture described in the Algorithm 4 is initialized with optimal hyper-parameters, trained, and tested on the test dataset.

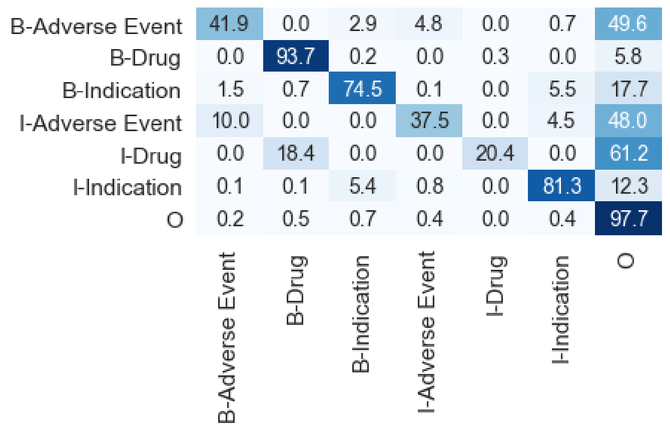
```
Algorithm 4: Building final BERT+BiLSTM model for ADR extraction
input         : pre-processed data
                                  ▷ input_dim:[batch size x seq_length]
output        : model output for each review at word-level
InitioalValues: batch size=16, seq_length=120, dropout=0.3,
                hidden_layer_number=200, label_number = 7
1. hidden layers=BERT.model (input) ;  ▷ (return the output of all 12 encoder
   of BERT model) hidden layers_dim:[batch size x seq_length x 768]
2. bert-out=Concat (4 last hidden layers) ;  ▷ (concatenation of 4 last layers)
   bert-out_dim:[batch size x seq_length x 3072]
3. drop-out=Dropout (bert-out,dropout)
;                          ▷ drop-out_dim:[batch size x seq_length x 3072]
4. lstm-out=LSTM(drop-out,bidirectional=True)
;              ▷ lstm-out_dim:[batch size x seq_length x 2*hidden_layer_number]
5. outputs=Linear (lstm-out,label_number);      ▷ outputs_dim:[batch size x
   seq_length x label_number]
```

The detailed results of the combined model are shown in Table 3.4. The Drug label got the highest score, and the model F1-score is 82.19 %.

Table 3.4 Test result of BERT with 4 last hidden layers concatenation + BiLSTM model

	Precision(%)	Recall(%)	F1-score(%)	Support
Adverse Event	43.56	42.28	42.91	272
Drug	92.32	94.41	93.35	1502
Indication	73.73	76.82	75.24	906
Micro avg	81.23	83.17	82.19	2680

The results of the BERT+BiLSTM model in the confusion matrix keep the same trend as the BERT model, which means that the most mispredicted label is the O label, but this misprediction has been reduced compared to the BERT model and also has been reduced to half compared to the BiLSTM model. The two labels, I-Adverse Event and I-Drug are mispredicted as B-Adverse Event and B-Drug by 14.8% and 18.4%, respectively. It can be seen that the values of true positive results increased, and the values of false negative and false positive results decreased compared to the BERT model implying a better performance for the combination model.



	B-Adverse Event	B-Drug	B-Indication	I-Adverse Event	I-Drug	I-Indication	O
B-Adverse Event	41.9	0.0	2.9	4.8	0.0	0.7	49.6
B-Drug	0.0	93.7	0.2	0.0	0.3	0.0	5.8
B-Indication	1.5	0.7	74.5	0.1	0.0	5.5	17.7
I-Adverse Event	10.0	0.0	0.0	37.5	0.0	4.5	48.0
I-Drug	0.0	18.4	0.0	0.0	20.4	0.0	61.2
I-Indication	0.1	0.1	5.4	0.8	0.0	81.3	12.3
O	0.2	0.5	0.7	0.4	0.0	0.4	97.7

Figure 3.16 The BERT with 4 last hidden layers concatenation + BiLSTM model confusion matrix

In Figure 3.17 it can be seen that the BERT+BiLSTM model gets more certain results for B-Drug, B-Indication and I-Indication compared to the other labels.

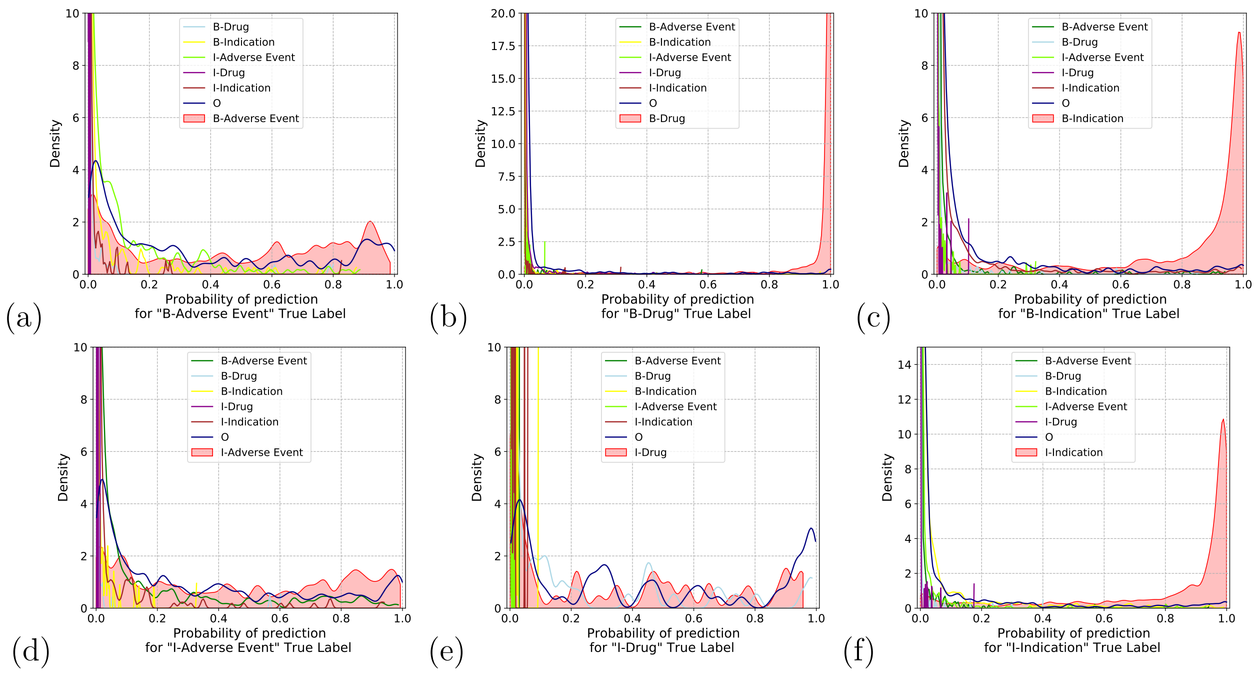


Figure 3.17 combination of BERT with 4 last hidden layers concatenation and BiLSTM model density plot for all labels

Comparison of BERT and BiLSTM models experimental results:

Now it is time to compare the performance of three models: BiLSTM, BERT with 4 last hidden layers concatenation (4Last HL Concat) and BERT(4Last HL Concat)+BiLSTM to find the best model of the study. From here on, during the thesis explanation, BERT will be used for less complexity instead of BERT with 4 last hidden layers concatenation (4Last HL Concat). According to the reported results in Figure 3.18, the final combined model that merges BERT+BiLSTM produces the best results.

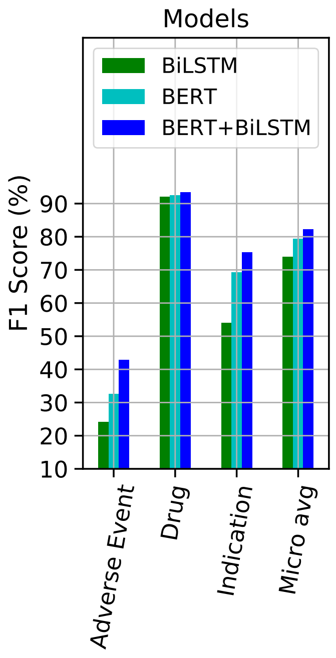
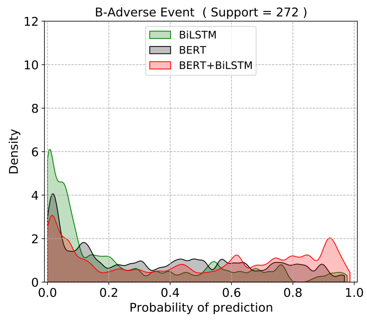
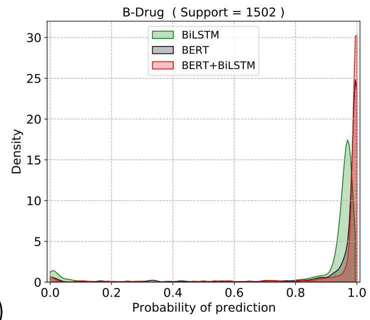


Figure 3.18 Comparison of the three model's performance

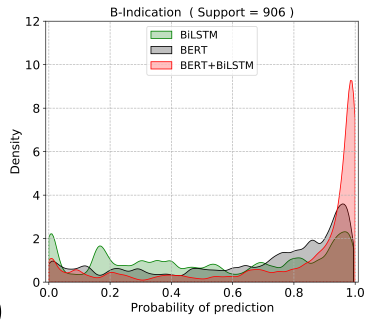
By comparing each label's results for the three models, it can be seen that the Drug label F1- score is the same with high values in all models' predictions. Improving the high value of scores is a challenging task. In the case of the Indication label, the BERT model leads the BiLSTM F1-score of about 15%; however, the BERT+BiLSTM model improved the BERT and BiLSTM model's results, respectively by 6% and 21%. The Adverse Event label witnesses the same trend. The BERT model leads the BiLSTM prediction F1-score by about 9%; however, the BERT+BiLSTM model leads the BERT and BiLSTM models prediction results, respectively 10% and 19%. The reason that the BERT model performs better and is more accurate than the BiLSTM model is probably due to the difference in architecture between the recurrent neural network model (BiLSTM) and the pre-trained (BERT) model. As described in Chapter 2, BiLSTM and BERT have different architectures. The BiLSTM method contains different layers, which have proven to perform well on textual data and sequences. However, they cannot always remember all of the important information that occur during the sequence. The BERT models make use of an attention mechanism that looks at the full sequence at once and detects what information is important. Therefore, the BERT models can have a better understanding of the text, and they are pre-trained models as well.



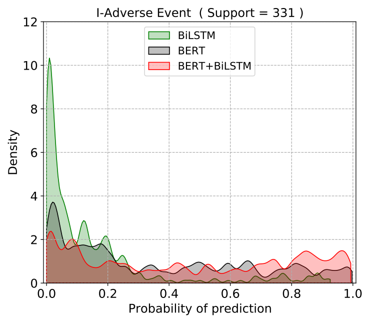
(a)



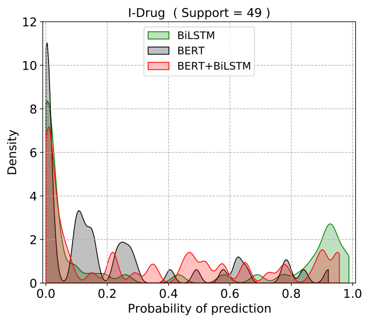
(b)



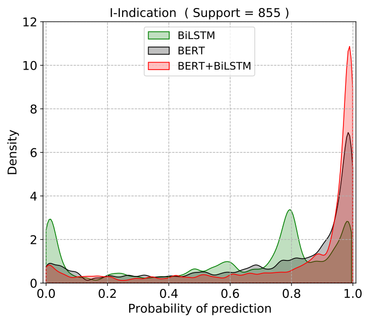
(c)



(d)



(e)



(f)

Figure 3.19 Comparison of the three models density plot for all labels

Figure 3.19 presents the density plots of three models for each label in the same graph. The plots show that both the number of examples and the selected model will affect the output probability values. The better performance of the BERT+BiLSTM model can be seen clearly by exploring the density plot of labels prediction. For each label, it can be seen that the output probability distribution of the BERT+BiLSTM model (with the pink color) is more skewed to the right, closer to 1 value, and has the sharper form. Therefore, the BERT+BiLSTM returns the most certain predictions compared to the other models. As explained before, the BERT model achieves better results than BiLSTM, probably due to the attention mechanism in its architecture. It can be seen that BERT+BiLSTM plots are sharper and more skewed to the right than BERT plots which means that adding BiLSTM layers to a BERT model will increase the value of output probability. The best and most certain prediction is for the B-Drug label, which has the most samples in the dataset. On the other hand, the worst belongs to the I-Drug, with the least number of examples in the dataset for all models.

Training time is another factor for evaluating and comparing the models. As shown in Table 3.5, for the three models of the study, the quickest is the BiLSTM model. It can be seen that there is a considerable difference between BiLSTM and BERT models. The training time for each epoch in the BERT model is almost 7 times longer than BiLSTM. This makes sense since the BERT model has a more complex architecture and has more parameters than the BiLSTM model. The training time of each epoch for the BERT+BiLSTM model is the highest one as expected since it is the combination of the two previous models with more parameters than the other models. Although the training time of each epoch of BERT+BiLSTM is longer than BERT's one, the whole training time for BERT+BiLSTM is less than BERT, due to the positive effect of the combination of BERT and BiLSTM models, which leads to fewer required epochs to achieve better performance.

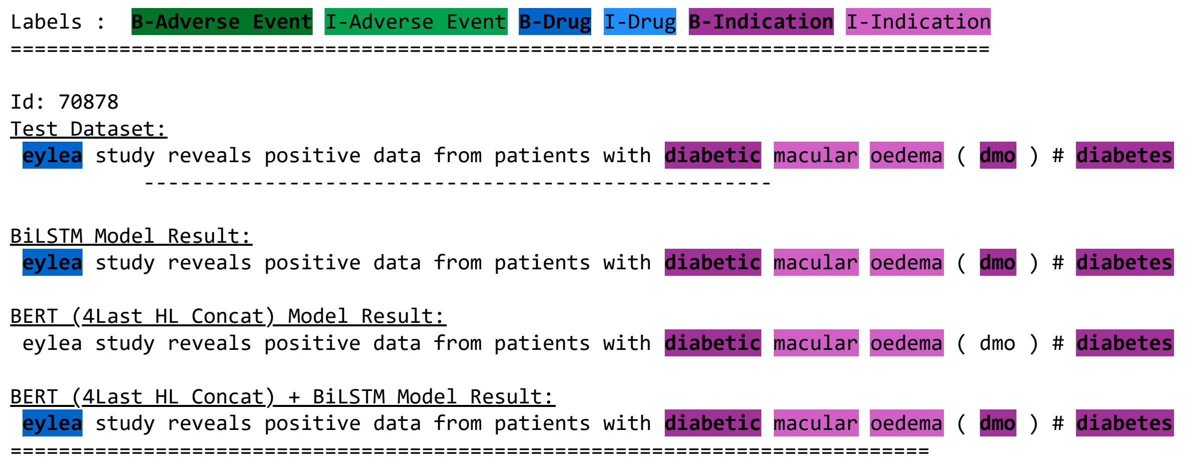
Table 3.5 Training time of models on GPU (GeForce GTX 1650)

Model	Each Epoch Time	Epoch	Training Time
BiLSTM	8 Sec	6	48 Sec
BERT(4 last layer concatenation)	40 Sec	40	26.5 Min
BERT(4 last layer concatenation)+BiLSTM	44 Sec	22	16 Min

3.7 Discussion

In this section, we analyze and explain the obtained results from the models considered in this thesis. It will be done by exploring the samples in the dataset and some example reviews with their predicted labels. In the dataset, there are some token-level and entity-level terms. Token-level consists of one word, and entity-level consists of more than one word. The research analysis will start with the token-level samples. The token-level terms have only B- label, and there are no boundary label for them (there is no I-), for example, token of "eylea" with B-Drug label, "dmo" (diabetic macular oedema) with the label of B-Indication, and "rop" (retinopathy of prematurity) with B-Indication label. By exploring the predicted results, it turns out that the BiLSTM model has better prediction results for the token-level terms with a limited number of examples in the dataset compared to the BERT model. Two predicted examples would illustrate this case from the test dataset in Figure 3.20 and Figure 3.21. In the examples, the Adverse Event labels are shown by green color (B-Adverse Event with dark green and I-Adverse Event with light green), the Drug and the Indication labels are indicated by blue (B-Drug with dark blue and I-Drug with light blue) and purple

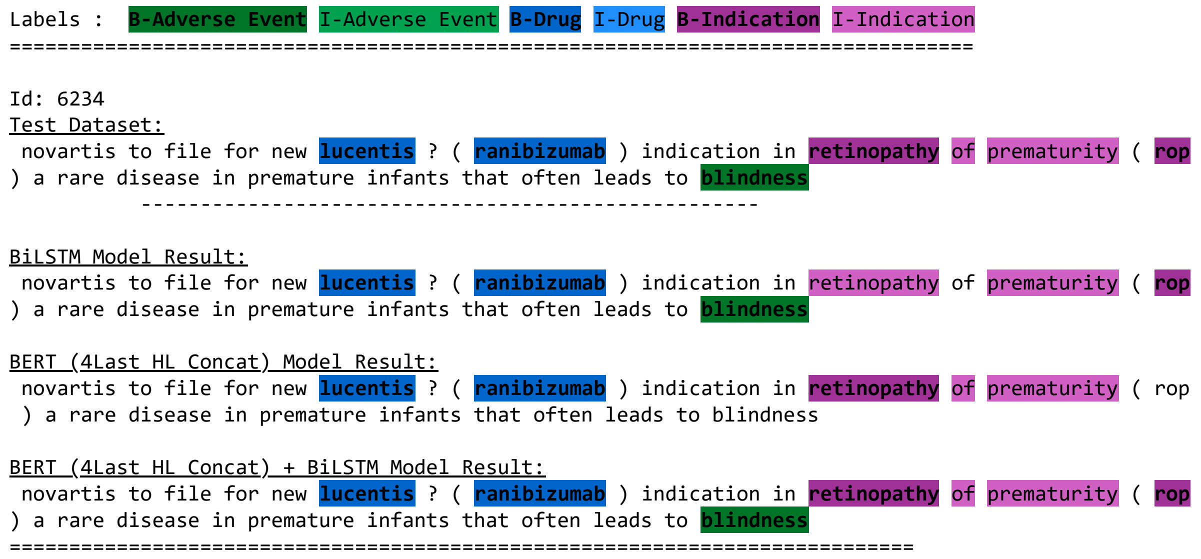
(B-Indication with dark purple and I-Indication with light purple) colors, respectively. The sentence with the Test Dataset title shows the true labels, and in the following, the predicted result of each model are shown. For example, in Figure 3.20, it can be seen that the BERT model missed predicting "eylea" and "dmo" as a drug, but the BiLSTM and BERT+BiLSTM models predicted it correctly.



```
Labels :  B-Adverse Event I-Adverse Event B-Drug I-Drug B-Indication I-Indication
=================================================================================

Id: 70878
Test Dataset:
 eylea study reveals positive data from patients with diabetic macular oedema ( dmo ) # diabetes
           ----------------------------------------------------

BiLSTM Model Result:
 eylea study reveals positive data from patients with diabetic macular oedema ( dmo ) # diabetes

BERT (4Last HL Concat) Model Result:
 eylea study reveals positive data from patients with diabetic macular oedema ( dmo ) # diabetes

BERT (4Last HL Concat) + BiLSTM Model Result:
 eylea study reveals positive data from patients with diabetic macular oedema ( dmo ) # diabetes
============================================================================
```

Figure 3.20 An example for token-level ("eylea" and "dmo") prediction of all models

In Figure 3.21 the BiLSTM and BERT+BiLSTM models tagged the word "rop" as an indication, but BERT did not distinguish this word as an indication.

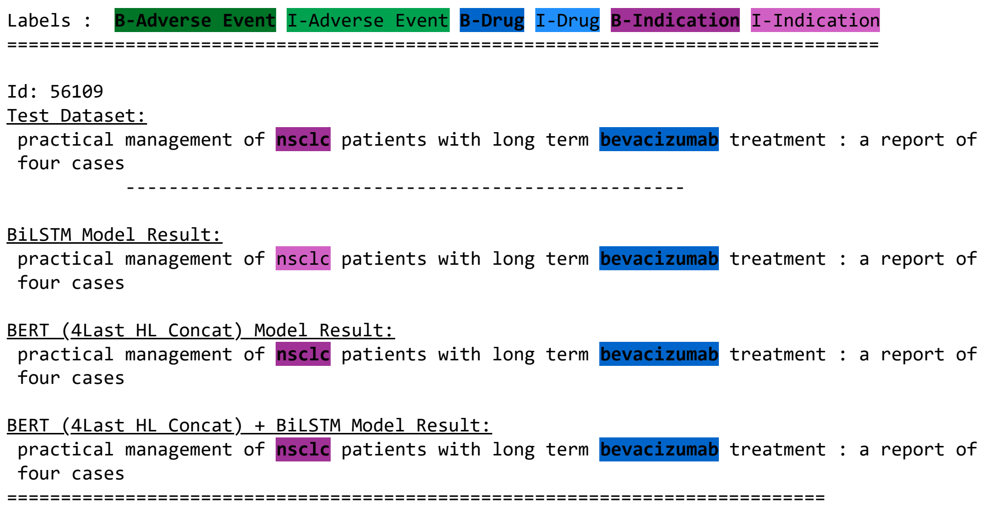


```
Labels :  B-Adverse Event I-Adverse Event B-Drug I-Drug B-Indication I-Indication
=================================================================================

Id: 6234
Test Dataset:
 novartis to file for new lucentis ? ( ranibizumab ) indication in retinopathy of prematurity ( rop
) a rare disease in premature infants that often leads to blindness
           ---------------------------------------------------

BiLSTM Model Result:
 novartis to file for new lucentis ? ( ranibizumab ) indication in retinopathy of prematurity ( rop
) a rare disease in premature infants that often leads to blindness

BERT (4Last HL Concat) Model Result:
 novartis to file for new lucentis ? ( ranibizumab ) indication in retinopathy of prematurity ( rop
 ) a rare disease in premature infants that often leads to blindness

BERT (4Last HL Concat) + BiLSTM Model Result:
 novartis to file for new lucentis ? ( ranibizumab ) indication in retinopathy of prematurity ( rop
) a rare disease in premature infants that often leads to blindness
============================================================================
```

Figure 3.21 An example for token-level ("rop") prediction of all models

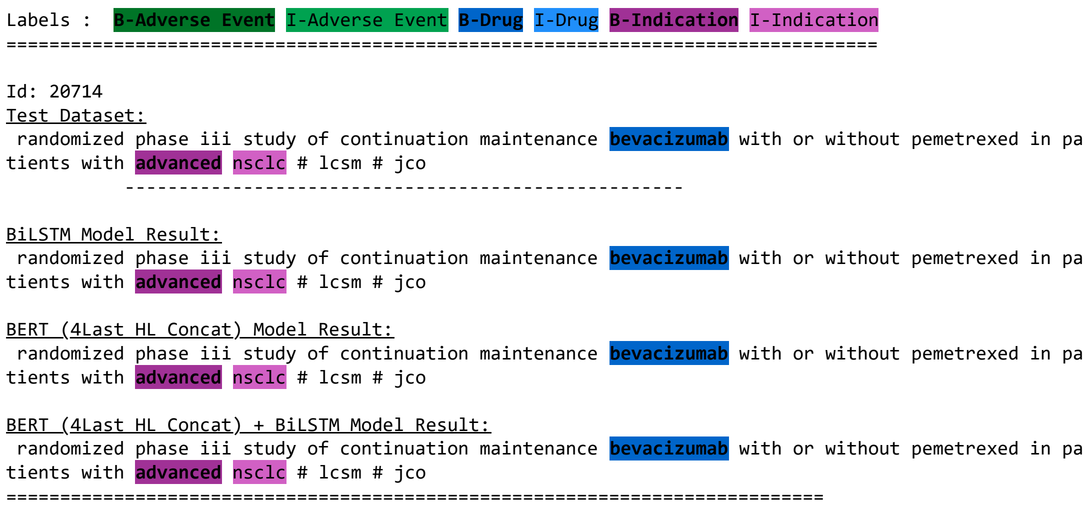
In both examples, the impact of adding the BiLSTM layer to the BERT model is visible. The BiLSTM layer improves the ability of the BERT model for the correct prediction of token-level terms. In the cases that the number of samples for the token-level terms in the training dataset is high, both BiLSTM and BERT models will get good results.

Some words in the training dataset are categorized in token-level as well as entity-level terms. For example, the words like "nsclc", "macular", and "retinopathy" are labeled in some sentences of the training dataset as B-Indication only (since there is one word and no boundaries, so no need for I-), and in some other sentences are labeled as I-Indication (for instance, the word "nsclc" in the term "advanced nsclc", the word "macular" in the term "wet macular degeneration", and the word "retinopathy" in the term "diabetic retinopathy").



```
Labels :  B-Adverse Event I-Adverse Event B-Drug I-Drug B-Indication I-Indication
=================================================================================

Id: 56109
Test Dataset:
 practical management of nsclc patients with long term bevacizumab treatment : a report of
 four cases
           ---------------------------------------------------

BiLSTM Model Result:
 practical management of nsclc patients with long term bevacizumab treatment : a report of
 four cases

BERT (4Last HL Concat) Model Result:
 practical management of nsclc patients with long term bevacizumab treatment : a report of
 four cases

BERT (4Last HL Concat) + BiLSTM Model Result:
 practical management of nsclc patients with long term bevacizumab treatment : a report of
 four cases
============================================================================
```

(a)



```
Labels :  B-Adverse Event I-Adverse Event B-Drug I-Drug B-Indication I-Indication
=================================================================================

Id: 20714
Test Dataset:
 randomized phase iii study of continuation maintenance bevacizumab with or without pemetrexed in pa
tients with advanced nsclc # lcsm # jco
           ---------------------------------------------------

BiLSTM Model Result:
 randomized phase iii study of continuation maintenance bevacizumab with or without pemetrexed in pa
tients with advanced nsclc # lcsm # jco

BERT (4Last HL Concat) Model Result:
 randomized phase iii study of continuation maintenance bevacizumab with or without pemetrexed in pa
tients with advanced nsclc # lcsm # jco

BERT (4Last HL Concat) + BiLSTM Model Result:
 randomized phase iii study of continuation maintenance bevacizumab with or without pemetrexed in pa
tients with advanced nsclc # lcsm # jco
============================================================================
```

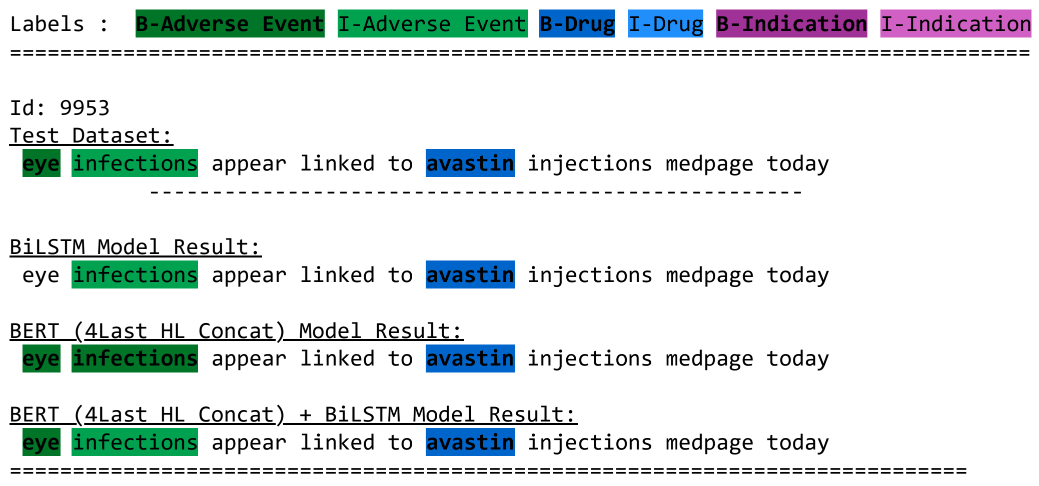
(b)

Figure 3.22 Examples for (a) teken-level ("nsclc") and (b) entity-level ("advance nsclc") prediction of all models

In the BiLSTM model, the prediction of such words depends on the number of times that they are repeated in the training dataset samples. For example, the word nsclc is labeled as B-Indication in 19 sentences and as I-Indication in 23 sentences of the training dataset. Thus, in the mentioned example, the BiLSTM model prediction is all the time I-Indication. However, the BERT model based on the word's position in the sentences and the effects of other words in the sentence makes the correct prediction (Figure 3.22)

As it is shown in Figure 3.22 (a), the "nsclc" is a token-level term with the true label of B-Indication. Still, the BiLSTM model predicts it as I-Indication, and in Figure 3.22 (b), the "nsclc" is a part of an entity-level term, and the BiLSTM correctly predicts it as I-Indication. Therefore, according to the obtained results of the study experiments and the cases stated, for the named entity recognition tasks, which have a limited number of token-level samples, the preferred model is BiLSTM rather than BERT. The reason is that the BiLSTM model will have better prediction and results, and also, it will take much less time for training.

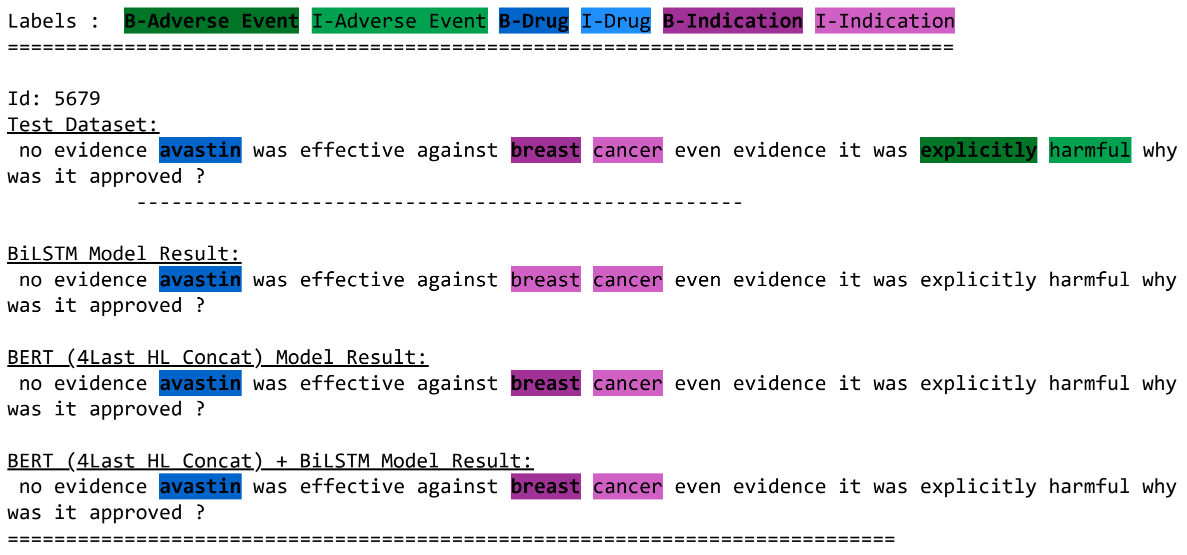
To analyze the entity-level terms clearly, let us do it by examples. Consider the entity-level term of "eye infections" with a B-Adverse Event label for the word eye and an I-Adverse Event label for the word infection. The BiLSTM model does not predict the "eye" word in the "eye infection" term correctly and assigned "O" label to it, since the word eye, which is repeated in the training dataset 115 times, is labeled as B-Adverse Event in 6 times, I-Adverse Event in 9, B-Indication in 10, I-Indication in 19, and O in 71 times of the 115 times. However, since the BERT model uses the context of the sentences and the position of the words, it yields a better prediction. Figure 3.23 shows the issue for the "eye infection" term.



```
Labels :  B-Adverse Event I-Adverse Event B-Drug I-Drug B-Indication I-Indication
==============================================================================

Id: 9953
Test Dataset:
 eye infections appear linked to avastin injections medpage today
           --------------------------------------------------

BiLSTM Model Result:
 eye infections appear linked to avastin injections medpage today

BERT (4Last HL Concat) Model Result:
 eye infections appear linked to avastin injections medpage today

BERT (4Last HL Concat) + BiLSTM Model Result:
 eye infections appear linked to avastin injections medpage today
=========================================================================
```

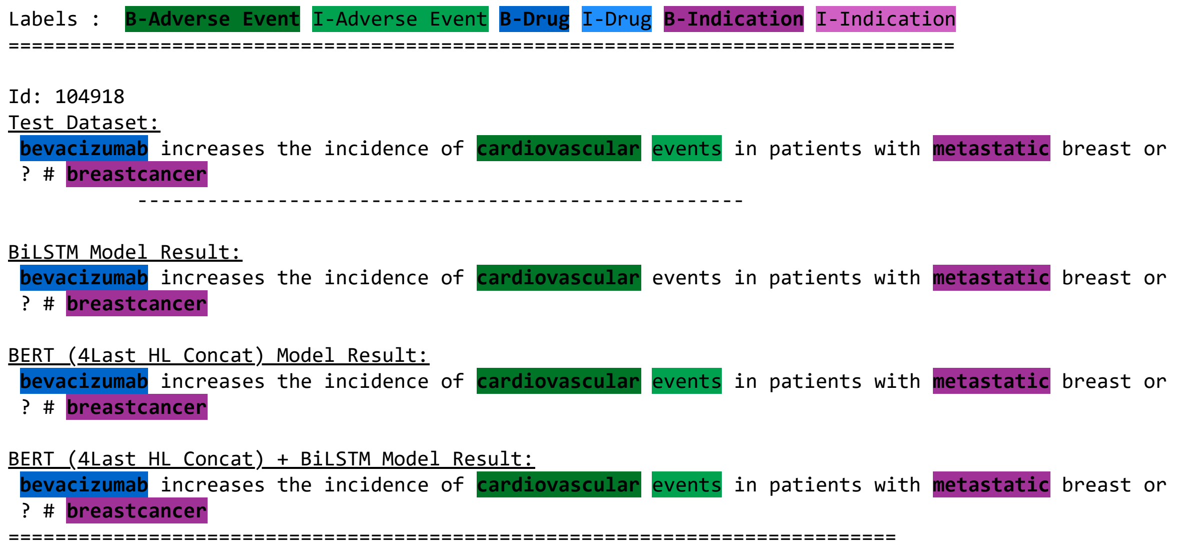
Figure 3.23 An example for entity-level term with different labels in the training dataset

Among the samples in the training dataset, there is an interesting case to address. Some terms consist of two words like "liver cancer", "bowel cancer", "ovarian cancer", and "breast cancer", and is labeled as B-Indication for Breast and I-Indication for cancer. In some examples, these terms are written like "livercancer", "bowelcancer", "ovariancancer", and "breastcancer" without space by users, and they are labeled as B-Indication only. The BiLSTM model predicted all the "breastcancer" words as B-Indication correctly, but it did not correctly predict none of the "breast cancer" terms, and the same for others. However, the BERT model performance for both terms (with and without space terms) was good (Figure 3.24).



```
Labels :  B-Adverse Event I-Adverse Event B-Drug I-Drug B-Indication I-Indication
=================================================================================

Id: 5679
Test Dataset:
 no evidence avastin was effective against breast cancer even evidence it was explicitly harmful why
was it approved ?
           ----------------------------------------------------

BiLSTM Model Result:
 no evidence avastin was effective against breast cancer even evidence it was explicitly harmful why
was it approved ?

BERT (4Last HL Concat) Model Result:
 no evidence avastin was effective against breast cancer even evidence it was explicitly harmful why
was it approved ?

BERT (4Last HL Concat) + BiLSTM Model Result:
 no evidence avastin was effective against breast cancer even evidence it was explicitly harmful why
was it approved ?
============================================================================
```

(a)



```
Labels :  B-Adverse Event I-Adverse Event B-Drug I-Drug B-Indication I-Indication
=================================================================================

Id: 104918
Test Dataset:
 bevacizumab increases the incidence of cardiovascular events in patients with metastatic breast or
 ? # breastcancer
           ----------------------------------------------------

BiLSTM Model Result:
 bevacizumab increases the incidence of cardiovascular events in patients with metastatic breast or
 ? # breastcancer

BERT (4Last HL Concat) Model Result:
 bevacizumab increases the incidence of cardiovascular events in patients with metastatic breast or
 ? # breastcancer

BERT (4Last HL Concat) + BiLSTM Model Result:
 bevacizumab increases the incidence of cardiovascular events in patients with metastatic breast or
 ? # breastcancer
============================================================================
```

(b)

Figure 3.24 Examples for (a) entity-level: "breast cancer" and (b) token-level: "breastcancer" prediction of all models

As it is shown in Figure 3.24 (a), "breast cancer" is an entity-level term with the true label of B-Indication for "breast" and I-Indication label for "cancer". Still, the BiLSTM model does not predict it correctly; rather, it predicts both as I-Indications. in Figure 3.24 (b), "breast cancer" is a token-level term, and the BiLSTM correctly predicts it as B-Indication.

Therefore from all these examples, it can be seen that the BiLSTM model pays more attention to the words, and the BERT model pays attention to the context and the position of the word related to each other. As a result, the combination model of BiLSTM and BERT leads to a better performance compared to each model taken separately.

Exploring the reason for low F1-score values in the three models results

The small size of the dataset is not the only reason for getting the low value of F1-scores. The study training dataset contains 2814 Drug labels related to the 6 different drug names (3 different drugs with their generic and brand names), which means that for each drug label, on average, there are 469 samples in the training dataset. This shows that the drugs' variety is low compared to the number of repetitions in the training dataset, leading to proper training and good F1-scores. The number of Indication labels in the training dataset is about 1631, related to the 58 different indication terms. This means that for each of the indication labels, the number of samples is 32 on average. Compared to the average number for each drug label (469), this number (32) is low, so as a result, the Indication label F1-score does not give a high score like for the Drug label. The number of repetitions of Adverse Event labels in the training dataset is 518. The number of varieties of the Adverse Events is 92, so the sample for each Adverse Event is about 5 samples, which shows the wide variety and low repetition. Due to the fewer training samples for such a variety of Adverse Events the training is bad, leading to a low F1-score compared to other labels.

Exploring the three models results according to their achieved TP, FN, and FP

Figure 3.25 shows the test dataset results of True Positive (TP), False Negative (FN), and False Positive (FP) for three models of study.

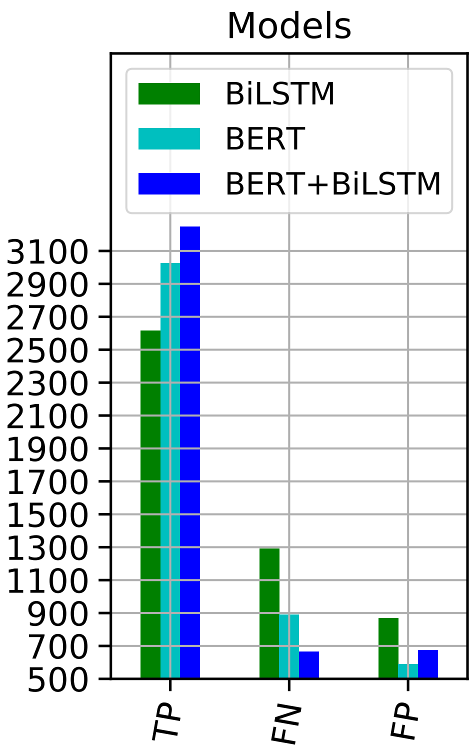
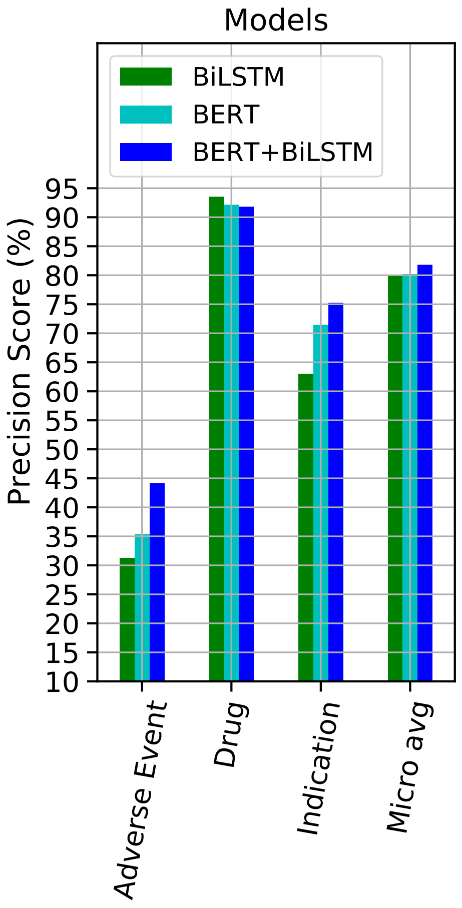
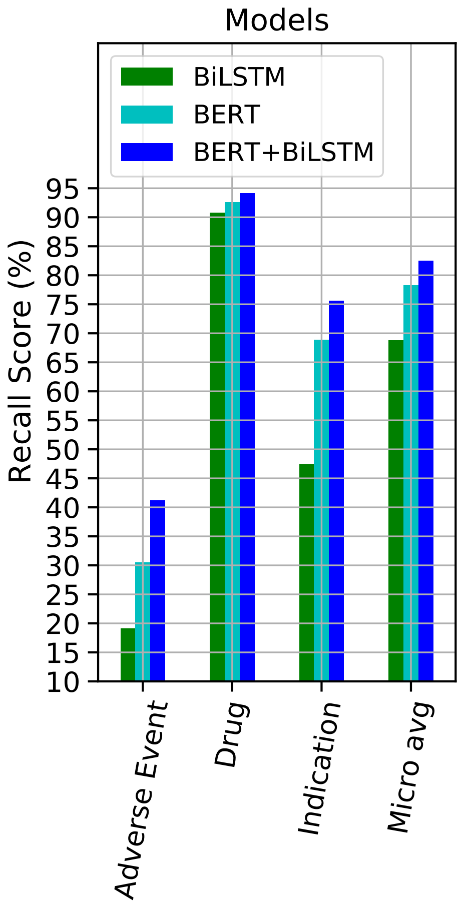


Figure 3.25 Comparison of the three models True Positive, False Negative, and False Positive in the test dataset

As can be seen, the values of FN and FP in the BERT model are reduced compared to the BiLSTM model, which causes an increase in the TP values (it is clear from the definitions of TP and FN, the value of TP+FN for a label is a fixed value). Based on the evaluation metrics formula mentioned in Section 3.3, decreasing FN and increasing TP will increase the recall value. Moreover, decreasing FP and increasing TP will increase the precision value. Based on the F1-score formula, increasing recall and precision increase the F1-score values, which shows the better performance of the BERT model compare to the BiLSTM model. Comparing the BERT model and the BERT+BiLSTM model shows that the FN value of the latter model is less than the former one, which causes an increase in TP. In the process of dataset annotation, some words are not annotated with the correct labels and are labeled as "O" by mistake. For example, words like blindness, cancer, crc, crcsm, and recurrent in some sentences are assigned by their correct labels, but in others, they are labeled as "O" incorrectly. But the BERT+BiLSTM model predicts their correct labels instead of the "O" label. Therefore, although the BERT+BiLSTM model predicts the correct label since the ground troth label in the test dataset is mislabeled, these predictions will be counted as FP and leads to an increase in the FP value in the BERT+BiLSTM model. However, although the value of FP is increased in the BERT+BiLSTM model compared to the BERT model, which should decrease the precision value, the increase of the TP value compensates for that and leads to the higher precision of the BERT+BiLSTM model. Figure 3.26 shows the precision and recall graph of models. The BERT+BiLSTM model got better precision and recall compared to other models.



(a)



(b)

Figure 3.26 (a) Precision results for the three models (b) Recall results for the three models

3.8 Summary

This chapter reported on several experiments performed with our three adopted models based on BiLSTM, BERT, and BiLSTM+BERT. The goal was to find the model with the best performance to extract the three desired drug adverse reactions from Twitter reviews. Some different concatenations of BERT layers were created. Among them, the BERT with the 4 last hidden layer concatenation shows the better result in terms of F1-score compared to other BERT's layer concatenation models, which led to selecting it to be combined with BiLSTM as our third model. After comparing the results of the BiLSTM model with F1-score of 74%, BERT with 4 last hidden layers concatenation model with F1-score of 79.37%, and finally BiLSTM+BERT(with 4 last hidden layers concatenation) model with F1-score of 82.19%, it appears that the last model outperformed the other models with better F1-scores. Therefore, the idea of adding the BiLSTM layer to BERT to continue extracting the features received from BERT's outputs to improve the word representation vector was practical.

CHAPTER 4 ADR DETECTION IMPROVEMENT WITH DATA AUGMENTATION TECHNIQUES

4.1 Detection (sentence-level classification) task introduction

The predicted results of the extraction task determine the name of the adverse drug reactions for drugs of interest. However, if some statistical results, such as the rate of adverse reactions for some specific drugs are more important than the name of the adverse drug reaction, then, the detection method will be more practical to apply. The advantage of the detection task compared to the extraction task is as follows: creating a dataset for the detection task requires less specialized skills and less time, which leads to reduced costs. In addition, the results for the detection task achieve higher precision and recall than the extraction task. Given the above, for gathering the adverse reactions, we suggest starting with applying the detection task and selecting the sentences which contain the adverse drug reaction, and using them to do the extraction task.

4.2 Dataset preparation steps

The data used to create the dataset for the detection task is the same as the data for the extraction task (explained in sections 3.1.1 and 3.1.2). The only difference is their labeling method. Detection is a sentence-level task, and the label will be assigned to the whole sentence instead of each word of the sentence. If each review contains an adverse drug reaction (ADR), this review is labeled as AE-Yes; otherwise, it is assigned an AE-No label. A typical labeled review is provided in Table 4.1

Table 4.1 Examples of ADR detection labeling (AE-Yes: the review contains an ADR, AE-No: the review does not contain any ADR

Row	Sentence	True Label
1	eye infections appear linked to avastin injections medpage today	AE-Yes
2	lucentis shows effect in reducing macular edema in diabetics	AE-No

The distribution of detection tasks for the two AE-Yes and AE-No classes in the dataset are shown in the table 4.2. That is as explained before as follows: in the training set (55%), validation set (15%) and test set (30%).

Table 4.2 Distribution of the main dataset

Samples	**Train**	**Val.**	**Test**
`AE-yes`	411	109	215
`AE-no`	1092	302	606

4.3 Dataset enlargement using different text augmentation techniques

The use of deep learning methods with large size datasets is recommended to improve accuracy. However, the generation of large annotated domain-specific datasets is costly. Therefore, a relevant approach is to increase the dataset size using data augmentation techniques in conjunction with pre-trained language models to deal with small dataset sizes. The main contribution in this chapter is to analyze and evaluate the extent to which some simple data augmentation techniques can improve pre-trained models when we are limited to small size domain-specific datasets.

4.3.1 Our proposed augmentation methods

Inspired by [57] (summarized in Section 2.6), we proposed four simple data augmentation techniques to investigate whether using EDA (see Section 2.6) operations and proposed techniques are effective for boosting the performance of BERT-Based models.

- Regular pattern deletion (RPD): Words are removed from the sentences according to a regular pattern: one word is deleted every m words. Specifically, $\alpha = \frac{1}{m}$ is the fraction of the number of words deleted. This technique was evaluated with m taking values in the range of {10,5,4,3,2} that gives α values of $\{0.1, 0.2, 0.25, 0.33, 0.5\}$

- Random pattern deletion (Pattern deletion Randomly (PDR)): Similar to the previous strategy, the number of words to be deleted are specified, but their positions are selected randomly. The main difference between the proposed random deletion method and EDA is that the fraction of the set of words deleted in a sentence is determined here. However, the EDA random deletion with a fixed value of $p = \alpha$ may delete no word or all words of a sentence.

- Small words deletion (SWD): Words with ch characters or less are deleted. This method was evaluated with ch chosen in$\{1, 2, 3, 4\}$.
- Last characters deletion (LCD): A fraction of the last characters of each word in sentences are removed. This value will be specified by n, which is defined as $n = \alpha l$, l representing the length of each word. This method was evaluated with $\alpha = \{0.1, 0.2, 0.25, 0.33, 0.5\}$.

4.4 Description of models used for ADR detection

Based on the results achieved in Chapter 3 for the extraction task, the model used to implement the detection task is the final model with the best results (BERT with concatenation of 4 last layers + BiLSTM), as shown in Figure 4.1. The only difference is that in the last layer, the number of labels is 1 instead of 7.

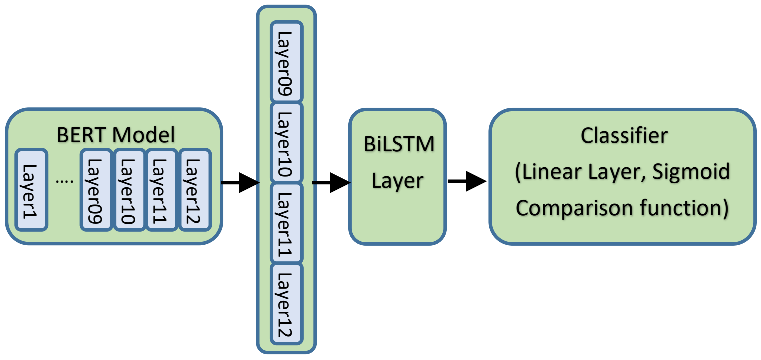


Figure 4.1 Detection task model framework

4.5 Evaluation metrics for ADR detection task

Classic metrics were chosen to compare various experiments. Specifically the F1-score was evaluated for each class (AE-Yes and AE-No) and the total result.

4.6 Implementation of the data augmentation techniques for ADR detection

As was done for the extraction experiments, for the detection task, several experiments were done to find the optimal hyperparameters for all detection experiments. Then, the test dataset was applied to the model with optimal hyperparameters for comparing the results. Figure 4.2 shows the overall framework of the experiments. In this flow graph, the model is the final detection BERT+BiLSTM model with the best performance from Chapter 3. The augmentation technique will only be applied to the training dataset after dividing the dataset into train, validation, and test datasets. The validation and test dataset was kept fixed to have reliable test results. Then, the augmented data was added to the training dataset and used to train the model. Finally, the trained model was fed by the test dataset to evaluate the performance of the model. In this study, different augmentation methods are used, as reported in the following.

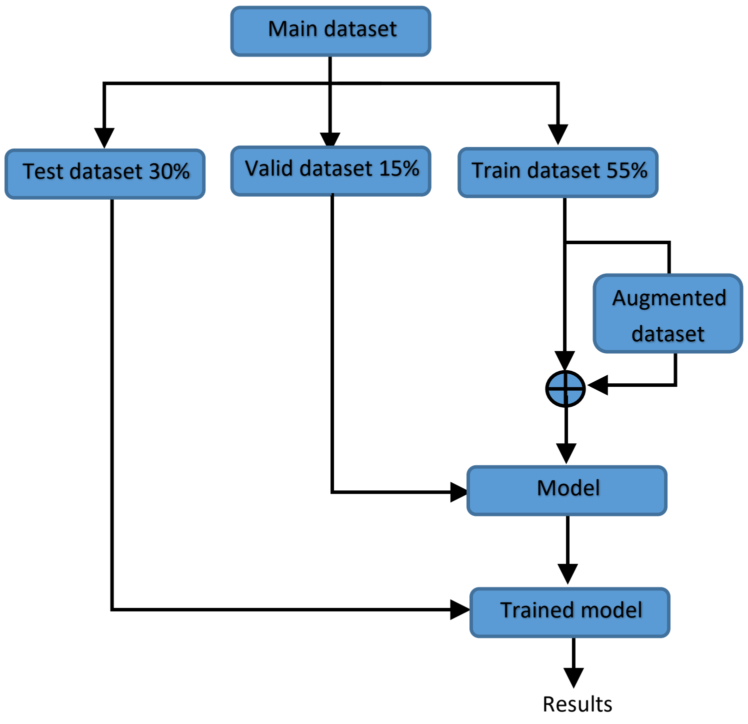


Figure 4.2 Overall experimental framework

The four new proposed augmentation techniques, All Easy Data Augmentation (EDA) and back-translation are utilized to enlarge the training dataset to improve the model's performance. Validation and test datasets are kept fixed in all experiments, and only the training dataset is changed and enlarged based on augmentation techniques.

For back-translation, English is used as the source language. Dutch, French, Italian, and Japanese are considered as target languages. The Dutch language was chosen as it is the most similar to English. Japanese was also chosen as the most different language compared to English. French and Italian are widely used languages after English. Google translation is used as the machine translation service. One augmented sentence per sentence for each language was generated in the training dataset. In the case of EDA, the model is run with different values of α. For each of the single EDA operations (SR, RI, RS, RD, which are explained in Section 2.6) and for each value of $\alpha = \{0.1, 0.2, 0.3, 0.4, 0.5\}$, one augmented sentence per sentence was generated in the training dataset. For each of the individual proposed operations and for each value of $\alpha = \{0.1, 0.2, 0.25, 0.33, 0.5\}$, one augmented sentence per sentence was generated in the training dataset. Then, to evaluate the advantage of combining several data augmentation techniques, a new dataset is created by concatenating the original dataset with augmented versions from the best results of each technique. This new dataset is ten times bigger than the main dataset. Moreover, another large dataset was created by concatenating the two best versions of each technique with the main dataset in order to have a database that is 19 times bigger than the main dataset. Some examples of augmented sentences are shown in appendix C.

4.7 Experimental results and discussion

A BERT+BiLSTM model was implemented and run across the main dataset without augmentation as the baseline. All the considered augmented datasets explore the effects of every single operation of the EDA, the four new techniques, and the back-translation on the model results. F1-score for AE-Yes and AE-No and the micro average (explained in Section 2.5) are shown in all result figures.

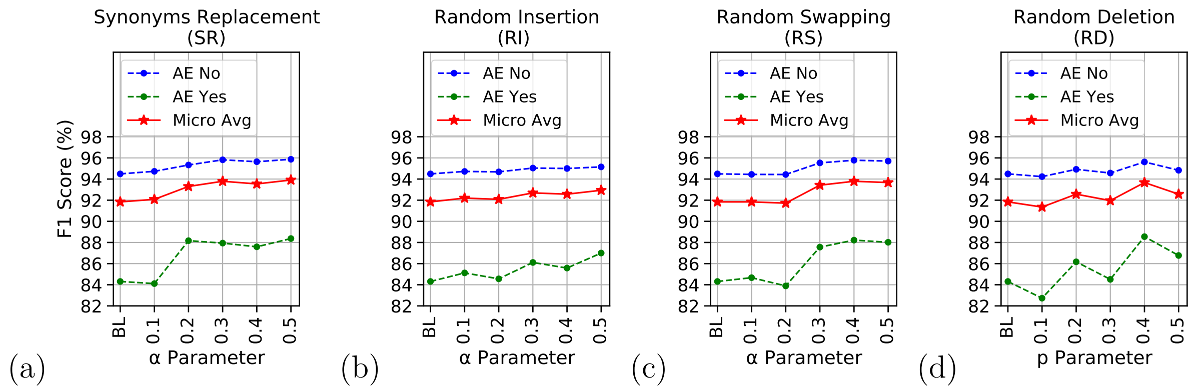


Figure 4.3 ADR detection results for the baseline (BL) and each of the individual EDA techniques

By exploring the results in Figure 4.3 and Figure 4.4, almost all the considered augmentation strategies contribute to boosting the performance of the proposed model. The improvement for AE-Yes is more significant compared to AE-No. The reason could be the high value of the F1-score of AE-No in the baseline model. Improving a high score appears to be more challenging. For SR, Figure 4.3 (a), best results are obtained at $\alpha > 0.2$ as RS Figure 4.3

(c) because both operations insert some noise to the dataset that leads to more diversity in the training dataset. For the RI technique, Figure 4.3 (b), different α values return almost the same result, likely because the adequate words in the sentences and their orders are maintained and introduce less noise and diversity in that case, the best result is obtained at $\alpha = 0.5$. For the RD strategy, Figure 4.3 (d), no clear tendency is observed; this may be due to the randomness of the deletion process. For instance, it is likely that several words in a sentence may be deleted sequentially. On the other hand, a very small number of words may be deleted, leading to the observed inconsistent behavior.

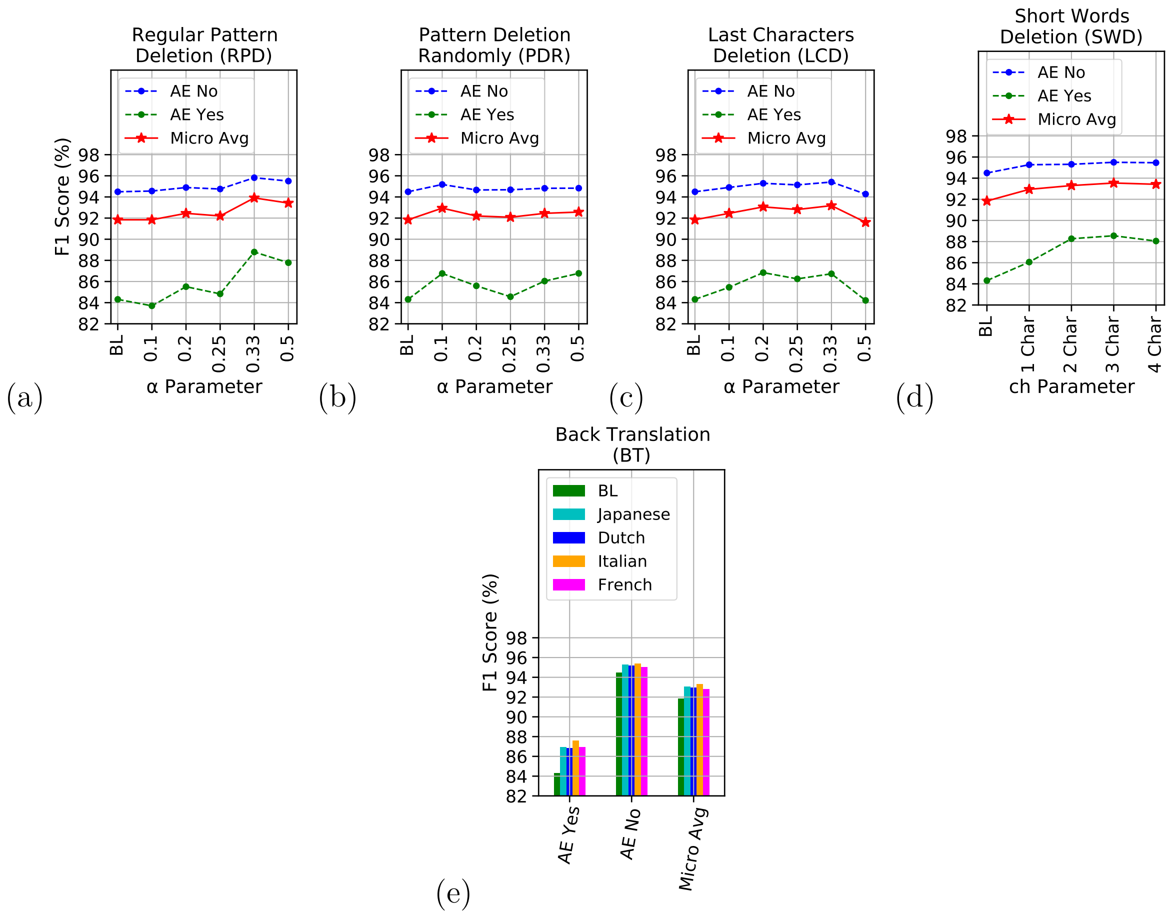


Figure 4.4 ADR detection results for the baseline (BL) and each of the individual proposed augmented and back-translation techniques

With the RPD strategy, Figure 4.4 (a), removing words creates new sentences that enlarge the size of the dataset; more significant improvements were observed for $\alpha > 0.33$ in the PDR technique. It is of interest that the Figure 4.4 (b) was implemented for the same values of α as the RPD technique; the results are different since, in PDR, the words are deleted randomly. In the case of SWD, Figure 4.4 (d), since the adverse drug effects are words with more than four characters, it is probable that using the technique does not delete important and effective words of the sentences. In SWD, the best results were obtained by removing words with three characters or less. With the LCD technique, Figure 4.4 (*c*), low and high deletion rates are not effective, and best results are obtained at α between 0.2 and 0.33. The reason could be that removing a specific number of last characters from words will keep the root of the words and consequently the meaning of the words, but deletion of too many characters changes the identity of the words. In the case of the back-translation (BT) strategy, Figure 4.4 (*e*), almost all translated datasets gave the same results while back-translation to Italian gave the best results. Applying this kind of back-translation increases diversity in the training dataset.

As shown in Figure 4.5, improvements obtained when the model was trained using the concatenated dataset (TrainDataset+9AugBest) outperformed all other augmentation techniques when applying them to the test dataset. An improvement of the baseline accuracy from 91.82 % to 95.25% (about 3.43 %) has been obtained. Moreover, using the concatenated dataset for the AE-Yes score boosted the baseline score from 84.3% to 91.24% (6.93% gain).

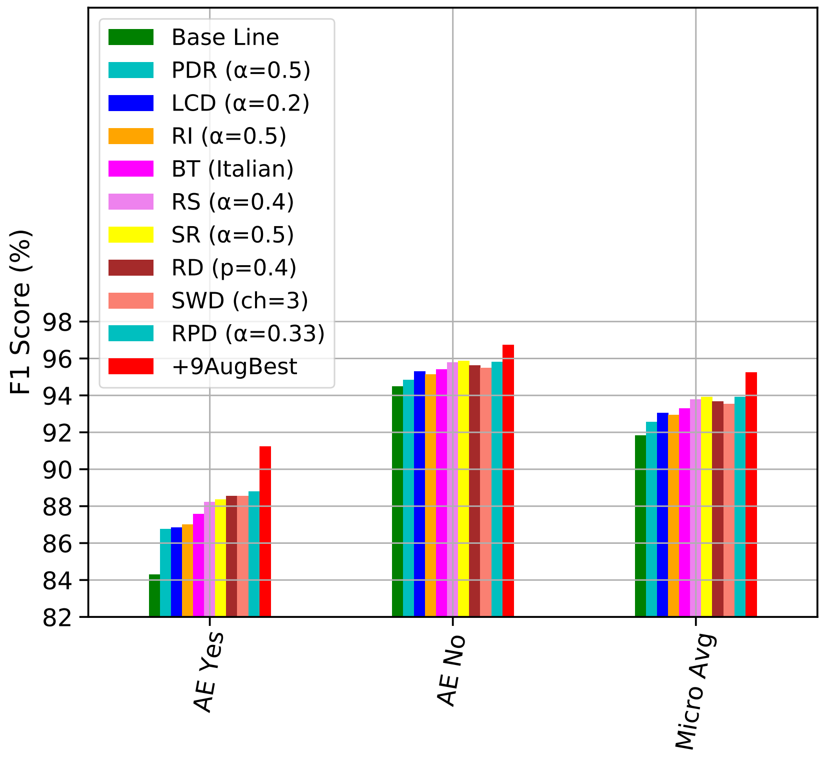
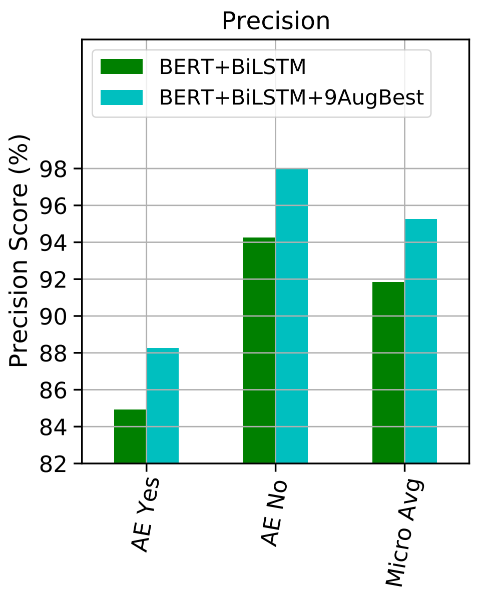
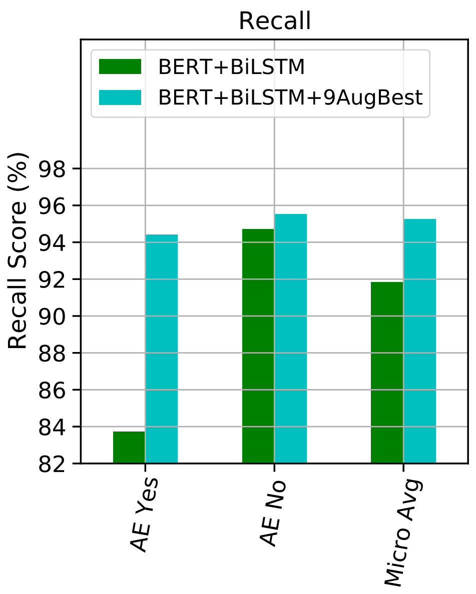


Figure 4.5 Comparison of results for best score obtained with each individual technique and concatenation of them (+9AugBest) with Baseline

In a closer investigation of the results, in Figure 4.6 the precision and recall of the BERT+BiLSTM (Base Line) model and BERT+BiLSTM+9AugBest are explored.



(a)



(b)

Figure 4.6 (a) The precision results for three models (b) The recall results for three models

As can be seen in the figures, using the augmentation method increased precision by 3.5% for both AE Yes and AE No labels. In the case of a recall, this method increased the AE Yes label's recall value by more than 10% and about 1% increase for AE No label. Therefore, the results show that the augmentation method is more effective for the labels with fewer samples in the training dataset. In total, the results show that the augmented dataset leads to an increase in both precision and recall and, as a result, increases the F1-sore of the model.

In Figure 4.7, we show the final result of the experiments, which created a dataset by selecting the two best results of each augmentation technique and by concatenating them. The results obtained with Dataset+9AugBest were better than those obtained with Dataset+18AugBest.

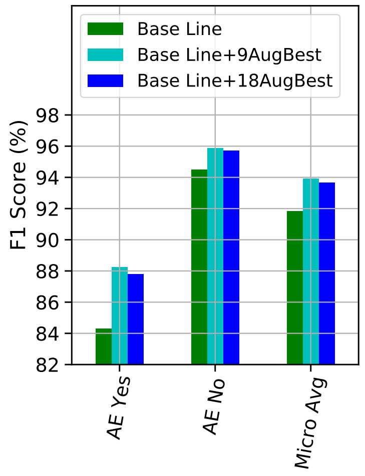


Figure 4.7 Comparison of results for the Train Dataset,Train Dataset+9AugBest and Train Dataset+18AugBest

However, [57] noted that applying EDA when using pre-trained models may not result in significant improvements. By contrast, our experiments show that using EDA and the proposed augmentation techniques improve the pre-trained model. Since our baseline model consists of BiLSTM and BERT, we performed another experiment without BiLSTM in our baseline model. We applied an augmented dataset to BERT to see how it would contribute to the improvement. These augmentation strategies have a main impact on the BERT part, as illustrated in Figure 4.8.

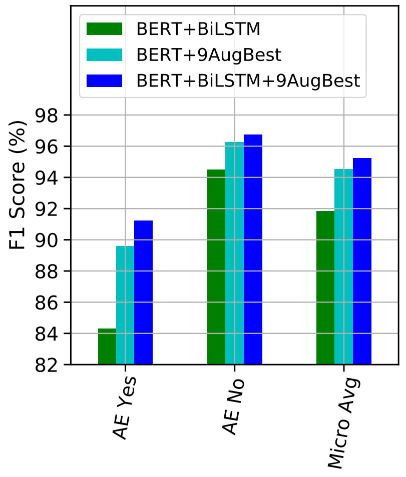


Figure 4.8 Comparison of results for the BERT+BiLSTM as baseline model, BERT and BERT+BiLSTM models with Train Dataset+9AugBest

4.8 Summary

In this chapter, four simple data augmentation techniques are proposed, and their effectiveness is examined along with EDA and back-translation techniques for text classification tasks. The reported results show that using text data augmentation methods can boost the results of pre-trained models such as BERT, and this could be a reliable solution for annotated data scarcity in domain-specific datasets. By comparing the proposed research results with other studies in this field, it is concluded that the improvements obtained with data augmentation techniques depend on the type and structure of the datasets.

CHAPTER 5 CONCLUSION

5.1 Summary of work performed

The unwanted effect of drug use called ADRs often leads to hospitalization, and some severe impact might lead to death, so early detection can prevent these issues from happening. In this thesis, the performance of three different models for automatic detection and extraction of ADR were explored. The goal is to find a model with better performance. One of the fast and reliable sources for gathering ADR data is Twitter. After gathering Twitter user reviews, pre-possessing and annotating it by human experts, these reviews were used as training and test datasets for the study.

By developing Natural Language Processing and introducing Deep Learning and transfer learning methods, some of the limitations of previously reported methods like lexicon and traditional machine learning methods are solved. The limitations include the need for periodic updates of the lexicon for a lexicon-based approach and manual feature creation for old machine learning methods. Therefore, this thesis investigates the performance of deep learning and transfer learning in detecting and extracting ADRs. The Deep learning models consist of word embedding, BiLSTM, and classifier layers. For the transfer learning model, the new state-of-the-art language model BERT is used. Some different experiments were performed to find the best performing model, such as summation and concatenation of the last 3, 4, and 5 last hidden layers of the BERT model. The BERT model with the concatenation of 4 last layers achieved the best performance among the others. To investigate the possibility of improving the performance, a combination of BERT and BiLSTM was created, trained, and tested in the dataset. Comparing all the final results in Figure 3.18 reveals that the F1-score of the BERT model is higher than the BiLSTM model, 79.37% compared to 74.0% of F1-score, and the BERT+BiLSTM model outperformed both two other models and got 82.19% of F1-score. This makes sense since BERT as a pre-trained model is already trained on vast amounts of text data and also, by using the attention mechanism, has a better conception of contexts compare to BiLSTM. For better results of BERT+BiLSTM, the reason could be that the added BiLSTM layer to BERT continued to extract the features received from BERT's output and improved the word representation vectors, leading to better prediction and better performance scores. Interestingly, these pre-trained models, trained on Wikipedia and official texts, performed well on tweets, although tweets often contain misspellings and slang. Moreover, if more annotated data was available, the models could get higher performances.

The study dataset is considered a small dataset. By exploring the examples and true labels in the dataset, it can be seen that for labels like "Drug", the number of samples was in the range that the model could get 93% F1-score. If the dataset was larger, maybe the F1-score could improve. Moreover, if the variety of the terms for the two labels of Adverse Event and Indication in the dataset were reduced, the model was able to get a better score in the result. Therefore, to spend a reasonable cost on preparing the dataset for the desired tasks, it is better to estimate the variety of labeled words and entities and, to gather the proper number of samples to create the dataset. By investigating the F1-score results for all labels, it can be seen that the Adverse Event is lower than the others. The main reason for that is the limited number of Adverse Event examples compared to other labels in the training dataset. Another reason is the long length of entity terms of Adverse Event compared to other labels, entity terms such as "loss of vision including peripheral", "reduced cognitive function and quality of life", and "outbreaks of endophthalmitis with blindness", that if one of the words in the entity is mispredicted, the whole entity will be considered misprediction and leads to a low F1-score value.

According to the result in this study, for the small dataset with token-level samples, BiLSTM might be a better choice than BERT since it makes more accurate predictions for these kinds of samples, and also the training time is smaller. The reason probably could be that when a word is not in the BERT vocabulary, the model uses the word-piece technique, and for accurate prediction, the model needs more samples for training.

For the detection task, the efficiency of using the augmentation method to enlarge the training dataset was investigated. Four proposed simple data augmentation techniques, along with EDA and back-translation techniques were applied to the small dataset of the study. After exploring the results, it was found that compared to the baseline model (BERT+BiLSTM without training in the augmented dataset), the +9AugBest model, which is trained with the augmented dataset, got better scores. Therefore using the augmentation technique was an effective way for improving the result, and this could be a reliable solution for annotated data scarcity in domain-specific datasets. However, excessively enlarging the dataset by the mentioned method will not necessarily improve the result. The reason might be that applying the augmentation technique in some of the samples can change the meaning and concept of the sentence in such a way that the true label is no longer correct for that sentence, which will cause incorrect training for that label and decrease performance.

The scenario of using the trained model of detection and extraction of ADR is as follows: in the first step, some new reviews with keywords of our three mentioned drugs are gathered. These reviews are then fed to our trained detection model, and the model predicts and classify these reviews as AE-Yes or AE-No. In the next step, we filter the reviews with AE-Yes labels and feed them to our trained extraction model as input. Finally, the output of the model will indicate ADR names for us.

5.2 Limitations

The most critical limitation in this study was finding a proper dataset for the desired tasks of the research, so for implementing the research, a specific dataset of the desired drugs is created and annotated. The dataset used for the study is small; for example, for some of the Adverse Events labels, there are only 1 or 2 examples that is not sufficient to train the models. To get higher scores, large Twitter corpus are needed but their production is expert-dependent, expensive and time-consuming.

5.3 Future research

To obtain higher scores, a large corpus to train the model should be developed. Applying the augmentation technique, used to enlarge the dataset for the detection task might also help make a large dataset for the extraction task and improve the performance. To expand the dataset, another method could be pseudo-labeling. It might be beneficial to test this method for both detection and extraction tasks and check if the results improve or not. Pseudo-labeling is the process in which a trained model of the project will predict some unlabelled data. Then the predicted results will be added to the dataset as the annotated data to enlarge the training dataset. Another way to improve the results might be to extend the model through additional layers like CNN or CRF, creating a new combination for the detection and extraction models. In addition, using more recent pre-trained language models such as the XLNet model instead of the BERT model might be another subject for future work of this research. Finally, all these suggested methods should be applied to the BERT+BiLSTM model to check their effectiveness.

REFERENCES

[1] A. J. Weiss *et al.*, “Adverse drug events in u.s. hospitals, 2010 versus 2014,” 2010. [Online]. Available: www.ahrq.gov/sites/default/files/publications/files/interimhacrate2014_2.pdf.

[2] M. C. Course, “Embeddings: Translating to a lower-dimensional space.” [Online]. Available: https://developers.google.com/machine-learning/crash-course/embeddings/translating-to-a-lower-dimensional-space

[3] A. Azzouni, R. Boutaba, and G. Pujolle, “Neuroute: Predictive dynamic routing for software-defined networks,” 09 2017.

[4] C. Olah, “Understanding lstm networks – colah’s blog,” 2015. [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[5] A. Vaswani *et al.*, “Attention is all you need,” *Advances in Neural Information Processing Systems*, vol. 2017-December, pp. 5999–6009, 6 2017. [Online]. Available: https://arxiv.org/abs/1706.03762v5

[6] N. Adaloglou, “How transformers work in deep learning and nlp: an intuitive introduction | ai summer.” [Online]. Available: https://theaisummer.com/transformer/

[7] J. Devlin *et al.*, “Bert: Pre-training of deep bidirectional transformers for language understanding,” vol. 1, 2019.

[8] A. M. Ghosh, “Early stopping with pytorch to restrain your model from overfitting,” 2020. [Online]. Available: https://medium.com/analytics-vidhya/early-stopping-with-pytorch-to-restrain-your-model-from-overfitting-dce6de4081c5

[9] A. ADRCanada, “Adverse drug reaction canada – working to prevent canada’s 4th leading cause of death,” 2021. [Online]. Available: https://adrcanada.org/

[10] X. Liu and H. Chen, “A research framework for pharmacovigilance in health social media: Identification and evaluation of patient adverse drug event reports,” *Journal of Biomedical Informatics*, vol. 58, pp. 268–279, 12 2015.

[11] R. Harpaz *et al.*, “Novel data mining methodologies for adverse drug event discovery and analysis,” *Clinical pharmacology and therapeutics*, vol. 91, p. 1010, 6 2012.

[Online]. Available: /pmc/articles/PMC3675775//pmc/articles/PMC3675775/?report=abstracthttps://www.ncbi.nlm.nih.gov/pmc/articles/PMC3675775/

[12] C. M *et al.*, “Why clinicians don’t report adverse drug events: Qualitative study,” *JMIR Public Health Surveill 2018;4(1):e21 https://publichealth.jmir.org/2018/1/e21*, vol. 4, p. e9282, 2 2018. [Online]. Available: https://publichealth.jmir.org/2018/1/e21

[13] Shannon and D. Cohen, “Can online consumers contribute to drug knowledge? a mixed-methods comparison of consumer-generated and professionally controlled psychotropic medication information on the internet,” *J Med Internet Res 2011;13(3):e53 https://www.jmir.org/2011/3/e53*, vol. 13, p. e1716, 7 2011. [Online]. Available: https://www.jmir.org/2011/3/e53

[14] L. Wu, T. S. Moh, and N. Khuri, “Twitter opinion mining for adverse drug reactions,” *Proceedings - 2015 IEEE International Conference on Big Data, IEEE Big Data 2015*, pp. 1570–1574, 12 2015.

[15] L. Hazell and S. Shakir, “Under-reporting of adverse drug reactions,” *Drug Safety*, vol. 29, 05 2006.

[16] A. Nikfarjam *et al.*, “Pharmacovigilance from social media: mining adverse drug reaction mentions using sequence labeling with word embedding cluster features,” *Journal of the American Medical Informatics Association*, vol. 22, pp. 671–681, 3 2015. [Online]. Available: https://academic.oup.com/jamia/article/22/3/671/776531

[17] D. Weissenbacher and G. Gonzalez-Hernandez, Eds., *Proceedings of the Fourth Social Media Mining for Health Applications (#SMM4H) Workshop & Shared Task*. Florence, Italy: Association for Computational Linguistics, Aug. 2019. [Online]. Available: https://aclanthology.org/W19-3200

[18] C. C. Freifeld *et al.*, “Digital drug safety surveillance: Monitoring pharmaceutical products in twitter,” *Drug Safety 2014 37:5*, vol. 37, pp. 343–350, 4 2014. [Online]. Available: https://link.springer.com/article/10.1007/s40264-014-0155-x

[19] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature 2015 521:7553*, vol. 521, pp. 436–444, 5 2015. [Online]. Available: https://www.nature.com/articles/nature14539

[20] M. E. Peters *et al.*, “Deep contextualized word representations,” *NAACL HLT 2018 - 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference*, vol. 1, pp. 2227–2237, 2 2018. [Online]. Available: https://arxiv.org/abs/1802.05365v2

[21] A. R. Openai *et al.*, “Improving language understanding by generative pre-training,” 2015. [Online]. Available: https://gluebenchmark.com/leaderboard

[22] J. Devlin *et al.*, “Bert: Pre-training of deep bidirectional transformers for language understanding,” *NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference*, vol. 1, pp. 4171–4186, 10 2018. [Online]. Available: https://arxiv.org/abs/1810.04805v2

[23] M. Schuster and K. Paliwal, “Bidirectional recurrent neural networks,” *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[24] M. Kuhn *et al.*, “A side effect resource to capture phenotypic effects of drugs,” *Molecular Systems Biology*, vol. 6, 2010. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/20087340/

[25] Q. Zeng-Treitler *et al.*, “Estimating consumer familiarity with health terminology: A context-based approach,” *Journal of the American Medical Informatics Association*, vol. 15. [Online]. Available: https://academic.oup.com/jamia/article/15/3/349/729054

[26] P. Mozzicato, “Meddra: An overview of the medical dictionary for regulatory activities,” pp. 65–75, 8 2009. [Online]. Available: https://link.springer.com/article/10.1007/BF03256752

[27] O. Bodenreider, “The unified medical language system (umls): Integrating biomedical terminology,” *Nucleic Acids Research*, vol. 32, 1 2004.

[28] R. Leaman *et al.*, “Towards internet-age pharmacovigilance: Extracting adverse drug reactions from user posts to health-related social networks,” pp. 117–125, 2010. [Online]. Available: http://www.hc-sc.gc.ca/dhp-mps/medeff/index-eng.php

[29] A. R. D. Canada, “Adverse reaction database - canada.ca.” [Online]. Available: https://www.canada.ca/en/health-canada/services/drugs-health-products/medeffect-canada/adverse-reaction-database.html

[30] A. Benton *et al.*, “Identifying potential adverse effects using the web: A new approach to medical hypothesis generation,” *Journal of Biomedical Informatics*, vol. 44, p. 989, 12 2011. [Online]. Available: https://repository.upenn.edu/oid_papers/41

[31] A. Yates and N. Goharian, “Adrtrace: Detecting expected and unexpected adverse drug reactions from user reviews on social media sites.” [Online]. Available: http://pewinternet.org/Commentary/2011/November/Pew-Internet-Health.aspx

[32] X. Liu and H. Chen, “Azdrugminer: An information extraction system for mining patient-reported adverse drug events in online patient forums,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8040 LNCS, pp. 134–150, 8 2013. [Online]. Available: https://arizona.pure.elsevier.com/en/publications/azdrugminer-an-information-extraction-system-for-mining-patient-r

[33] H. Gurulingappa, A. Mateen-Rajput, and L. Toldo, “Extraction of potential adverse drug events from medical case reports,” *Journal of Biomedical Semantics*, vol. 3, p. 15, 12 2012. [Online]. Available: /pmc/articles/PMC3599676//pmc/articles/PMC3599676/?report=abstracthttps://www.ncbi.nlm.nih.gov/pmc/articles/PMC3599676/

[34] I. Alimova and E. Tutubalina, “Automated detection of adverse drug reactions from social media posts with machine learning,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10716 LNCS, pp. 3–15, 2017. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-73013-4_1

[35] L. M *et al.*, “Large-scale prediction of adverse drug reactions using chemical, biological, and phenotypic properties of drugs,” *Journal of the American Medical Informatics Association : JAMIA*, vol. 19, 6 2012. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/22718037/

[36] A. Cocos, A. G. Fiks, and A. J. Masino, “Deep learning for pharmacovigilance: recurrent neural network architectures for labeling adverse drug reactions in twitter posts,” *Journal of the American Medical Informatics Association*, vol. 24, pp. 813–821, 7 2017. [Online]. Available: https://academic.oup.com/jamia/article/24/4/813/3041102

[37] K. Lee *et al.*, “Adverse drug event detection in tweets with semi-supervised convolutional neural networks,” 2017. [Online]. Available: http://dx.doi.org/10.1145/3038912.3052671

[38] T. Huynh *et al.*, “Adverse drug reaction classification with deep neural networks,” pp. 877–887, 2016. [Online]. Available: https://aclanthology.org/C16-1084

[39] W. S *et al.*, “Adverse drug event detection from electronic health records using hierarchical recurrent neural networks with dual-level embedding,” *Drug safety*, vol. 42, pp. 113–122, 1 2019. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/30649736/

[40] F. Li *et al.*, “A neural joint model for entity and relation extraction from biomedical text,” *BMC Bioinformatics 2017 18:1*, vol. 18, pp. 1–11, 3 2017. [Online]. Available: https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-017-1609-9

[41] A. Radfort *et al.*, “Improving language understanding by generative pre-training,” *OpenAI*, 2018.

[42] S. Chen *et al.*, “Hitsz-icrc: A report for smm4h shared task 2019-automatic classification and extraction of adverse effect mentions in tweets,” pp. 47–51, 9 2019. [Online]. Available: https://aclanthology.org/W19-3206

[43] A. Breden and L. Moore, “Detecting adverse drug reactions from twitter through domain-specific preprocessing and bert ensembling,” 5 2020. [Online]. Available: https://arxiv.org/abs/2005.06634v1

[44] R. Grishman and B. M. Sundheim, “Message understanding conference- 6: A brief history,” 1996. [Online]. Available: https://aclanthology.org/C96-1079

[45] G. Developers, “Google developers blog: Introducing tensorflow feature columns.” [Online]. Available: https://developers.googleblog.com/2017/11/introducing-tensorflow-feature-columns.html

[46] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature 1986 323:6088*, vol. 323, pp. 533–536, 1986. [Online]. Available: https://www.nature.com/articles/323533a0

[47] M. M. Department, M. M. Department, and M. C. Mozer, “Induction of multiscale temporal structure,” *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS 4*, pp. 275–282, 1992. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.6.2785

[48] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, pp. 1735–1780, 11 1997.

[49] D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 9 2014. [Online]. Available: https://arxiv.org/abs/1409.0473v7

[50] C. Sun *et al.*, “Revisiting unreasonable effectiveness of data in deep learning era,” *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2017-October, pp. 843–852, 12 2017.

[51] Y. LeCun *et al.*, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, pp. 2278–2323, 1998.

[52] R. Galinsky, A. Alekseev, and S. I. Nikolenko, "Improving neural network models for natural language processing in russian with synonyms," 2017.

[53] G. Rizos, K. Hemker, and B. Schuller, "Augment to prevent," 2019.

[54] A. W. Yu *et al.*, "Qanet: Combining local convolution with global self-attention for reading comprehension," 2018.

[55] J. Ma and L. Li, "Data augmentation for chinese text classification using back-translation," vol. 1651, 2020.

[56] M. Fadaee, A. Bisazza, and C. Monz, "Data augmentation for low-resource neural machine translation," vol. 2, 2017.

[57] J. Wei and K. Zou, "Eda: Easy data augmentation techniques for boosting performance on text classification tasks," 2020.

[58] Y. Chen *et al.*, "Twitter sentiment analysis via bi-sense emoji embedding and attention-based lstm," *MM 2018 - Proceedings of the 2018 ACM Multimedia Conference*, pp. 117–125, 10 2018. [Online]. Available: https://doi.org/10.1145/3240508.3240533

[59] Pablo, "Tweepy," 2013. [Online]. Available: https://www.tweepy.org/

[60] E. F. T. K. Sang and S. Buchholz, "Introduction to the conll-2000 shared task: Chunking," pp. 127–132, 9 2000. [Online]. Available: https://arxiv.org/abs/cs/0009008v1

[61] N. Alshammari and S. Alanazi, "The impact of using different annotation schemes on named entity recognition," *Egyptian Informatics Journal*, 11 2020.

[62] H. Nakayama *et al.*, "doccano: Text annotation tool for human," 2018, software available from https://github.com/doccano/doccano. [Online]. Available: https://github.com/doccano/doccano

[63] T. Wolf *et al.*, "Huggingface's transformers: State-of-the-art natural language processing," *arXiv preprint arXiv:1910.03771*, 10 2019. [Online]. Available: https://arxiv.org/abs/1910.03771v5

APPENDIX A DATASET DIVIDING RATIO EXPLANATION

Here is an example to show that with less testing data, the performance will have higher variance. Considering the B-Adverse Event with 100 instances in the test dataset and $TP = 40$, $FN = 60$ and $FP = 30$. Utilizing the formula in Section 2.5 the evaluation metrics for this example will be as follows:

$Recall = 40\%$, $Precision = 57.14\%$, $F1 - score = 47.06\%$

Now by changing only one prediction, $TP = 39$, $FN = 61$ and $FP = 30$ the results will change as follows:

$Recall = 39\%$, $Precision = 56.52\%$, $F1 - score = 46.15\%$

It can be seen that just by one change in prediction values, the F1-score changed about 1%.

Now the number of B-Adverse Event is tripled and is changed to 300 and the ratio for TP, FN and FP is kept constant as follows:$TP = 120$, $FN = 180$ and $FP = 90$. the evaluation metrics will be as follows:

$Recall = 40\%$, $Precision = 57.14\%$, $F1 - score = 47.06\%$

Now by changing only one prediction, $TP = 119$, $FN = 181$ and $FP = 90$ the results will change as follows:

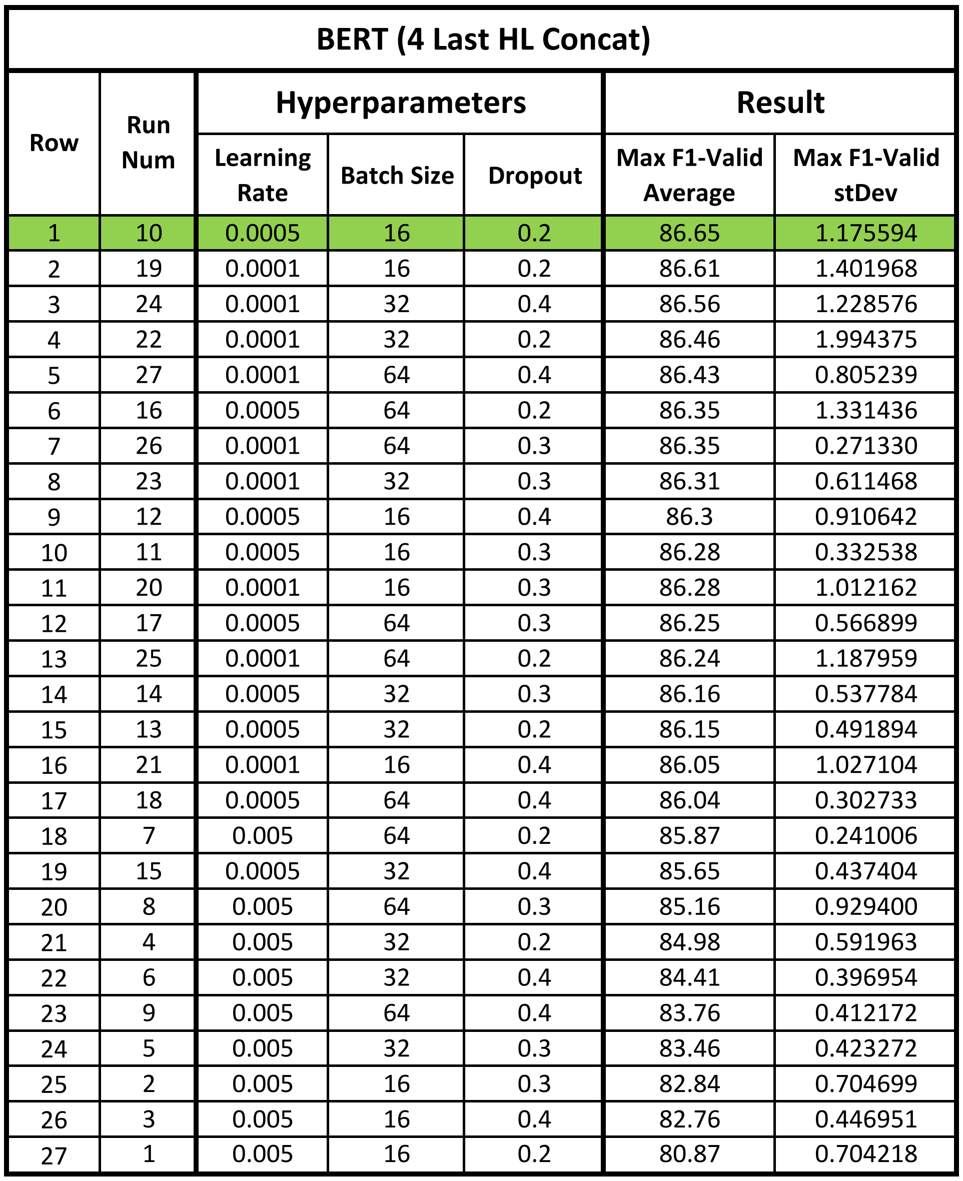
$Recall = 39.67\%$, $Precision = 56.94\%$, $F1 - score = 46.76\%$

It can be seen that just by one change in prediction values, the F1-score changed about 0.3% which shows that by having more testing data, the F1-score will have less variance and are more reliable for assessing the performance of the model.

APPENDIX B EXPERIMENT RESULTS FOR FINDING OPTIMAL HYPERPARAMETERS

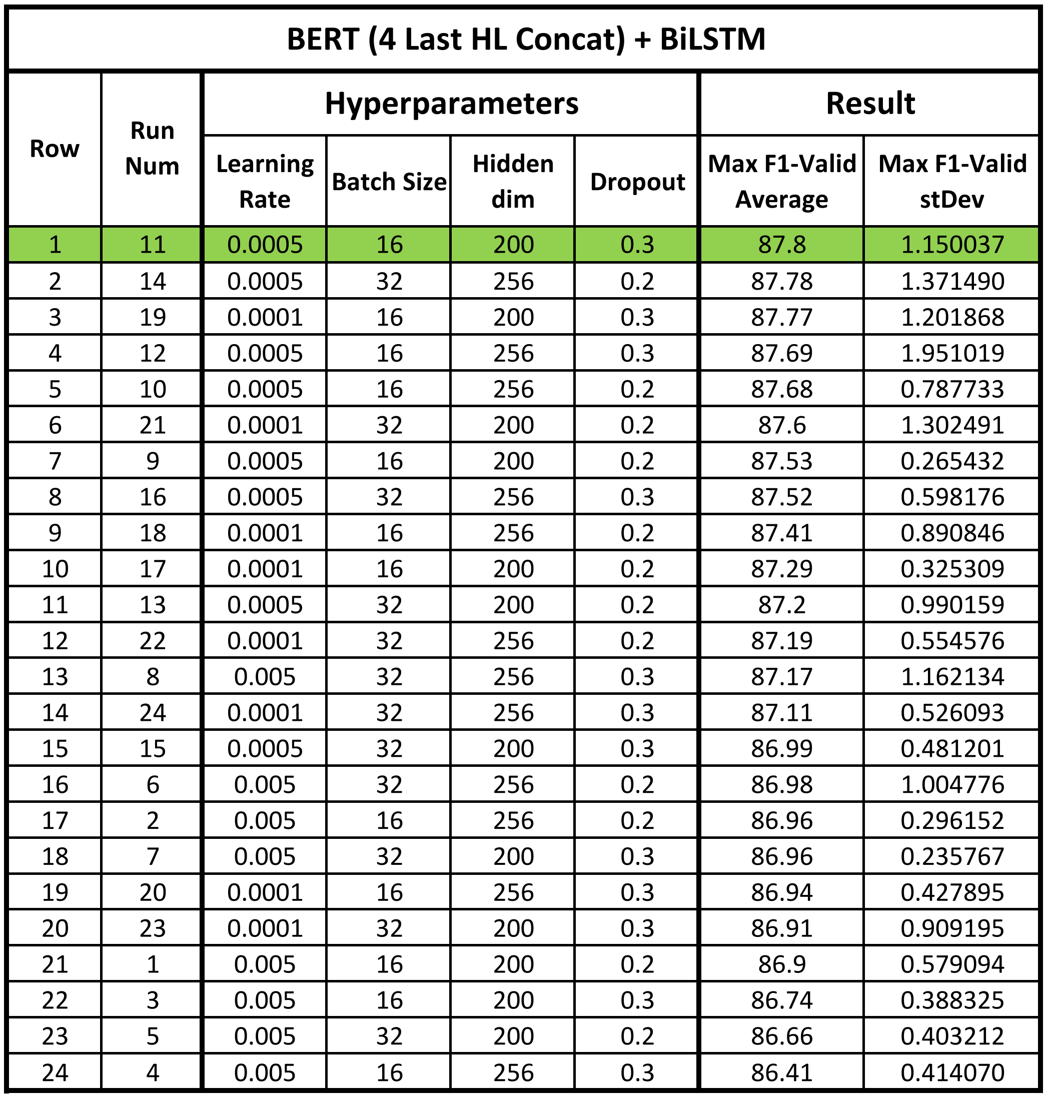
BiLSTM								
Row	Run Num	Hyperparameters					Result	
		Learning Rate	Batch Size	Embedding dim	Hidden dim	Dropout	Max F1-Valid Average	Max F1-Valid stDev
1	23	0.005	16	512	256	0.2	74.064	2.300075
2	24	0.005	16	512	256	0.3	73.954	2.742981
3	18	0.005	16	256	256	0.3	73.644	2.403735
4	17	0.005	16	256	256	0.2	73.256	3.902038
5	20	0.005	16	512	150	0.3	73.212	1.575467
6	11	0.005	16	100	256	0.2	73.028	2.604983
7	178	0.0005	32	100	200	0.3	72.872	0.530863
8	45	0.005	32	512	200	0.2	72.778	1.196351
9	34	0.005	32	100	200	0.3	72.774	1.781691
10	47	0.005	32	512	256	0.2	72.684	0.650618
11	15	0.005	16	256	200	0.2	72.668	1.980317
12	33	0.005	32	100	200	0.2	72.668	1.109151
13	22	0.005	16	512	200	0.3	72.664	2.324268
14	105	0.001	32	100	200	0.2	72.662	1.052186
15	31	0.005	32	100	150	0.2	72.582	0.962401
16	26	0.005	32	10	150	0.3	72.548	2.009551
17	43	0.005	32	512	150	0.2	72.486	0.592304
18	115	0.001	32	512	150	0.2	72.444	0.471534
19	46	0.005	32	512	200	0.3	72.438	0.855790
20	7	0.005	16	100	150	0.2	72.434	1.818390
21	39	0.005	32	256	200	0.2	72.41	1.158188
:	:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:	:
200	61	0.005	64	256	150	0.2	69.72	1.224614
201	145	0.0005	16	10	150	0.2	69.718	2.650807
202	171	0.0005	32	10	200	0.2	69.692	1.419808
203	141	0.001	64	512	200	0.2	69.688	1.218481
204	69	0.005	64	512	200	0.2	69.684	1.515514
205	213	0.0005	64	512	200	0.2	69.656	1.706008
206	123	0.001	64	10	200	0.2	69.644	1.120350
207	196	0.0005	64	10	200	0.3	69.534	0.600087
208	194	0.0005	64	10	150	0.3	69.504	1.406252
209	70	0.005	64	512	200	0.3	69.488	1.269636
210	169	0.0005	32	10	150	0.2	69.294	0.796507
211	63	0.005	64	256	200	0.2	69.292	1.296848
212	216	0.0005	64	512	256	0.3	69.23	0.358162
213	195	0.0005	64	10	200	0.2	69.22	1.858892
214	68	0.005	64	512	150	0.3	69.128	1.439255
215	203	0.0005	64	100	256	0.2	69.08	1.555236
216	197	0.0005	64	10	256	0.2	69.022	1.544052

Figure B.1 The BiLSTM model experiment results for finding optimal hyperparameters



BERT (4 Last HL Concat)						
Row	Run Num	Hyperparameters			Result	
		Learning Rate	Batch Size	Dropout	Max F1-Valid Average	Max F1-Valid stDev
1	10	0.0005	16	0.2	86.65	1.175594
2	19	0.0001	16	0.2	86.61	1.401968
3	24	0.0001	32	0.4	86.56	1.228576
4	22	0.0001	32	0.2	86.46	1.994375
5	27	0.0001	64	0.4	86.43	0.805239
6	16	0.0005	64	0.2	86.35	1.331436
7	26	0.0001	64	0.3	86.35	0.271330
8	23	0.0001	32	0.3	86.31	0.611468
9	12	0.0005	16	0.4	86.3	0.910642
10	11	0.0005	16	0.3	86.28	0.332538
11	20	0.0001	16	0.3	86.28	1.012162
12	17	0.0005	64	0.3	86.25	0.566899
13	25	0.0001	64	0.2	86.24	1.187959
14	14	0.0005	32	0.3	86.16	0.537784
15	13	0.0005	32	0.2	86.15	0.491894
16	21	0.0001	16	0.4	86.05	1.027104
17	18	0.0005	64	0.4	86.04	0.302733
18	7	0.005	64	0.2	85.87	0.241006
19	15	0.0005	32	0.4	85.65	0.437404
20	8	0.005	64	0.3	85.16	0.929400
21	4	0.005	32	0.2	84.98	0.591963
22	6	0.005	32	0.4	84.41	0.396954
23	9	0.005	64	0.4	83.76	0.412172
24	5	0.005	32	0.3	83.46	0.423272
25	2	0.005	16	0.3	82.84	0.704699
26	3	0.005	16	0.4	82.76	0.446951
27	1	0.005	16	0.2	80.87	0.704218

Figure B.2 The BERT (4 last layer concatenation) model experiment results for finding optimal hyperparameters



BERT (4 Last HL Concat) + BiLSTM							
Row	Run Num	Hyperparameters				Result	
		Learning Rate	Batch Size	Hidden dim	Dropout	Max F1-Valid Average	Max F1-Valid stDev
1	11	0.0005	16	200	0.3	87.8	1.150037
2	14	0.0005	32	256	0.2	87.78	1.371490
3	19	0.0001	16	200	0.3	87.77	1.201868
4	12	0.0005	16	256	0.3	87.69	1.951019
5	10	0.0005	16	256	0.2	87.68	0.787733
6	21	0.0001	32	200	0.2	87.6	1.302491
7	9	0.0005	16	200	0.2	87.53	0.265432
8	16	0.0005	32	256	0.3	87.52	0.598176
9	18	0.0001	16	256	0.2	87.41	0.890846
10	17	0.0001	16	200	0.2	87.29	0.325309
11	13	0.0005	32	200	0.2	87.2	0.990159
12	22	0.0001	32	256	0.2	87.19	0.554576
13	8	0.005	32	256	0.3	87.17	1.162134
14	24	0.0001	32	256	0.3	87.11	0.526093
15	15	0.0005	32	200	0.3	86.99	0.481201
16	6	0.005	32	256	0.2	86.98	1.004776
17	2	0.005	16	256	0.2	86.96	0.296152
18	7	0.005	32	200	0.3	86.96	0.235767
19	20	0.0001	16	256	0.3	86.94	0.427895
20	23	0.0001	32	200	0.3	86.91	0.909195
21	1	0.005	16	200	0.2	86.9	0.579094
22	3	0.005	16	200	0.3	86.74	0.388325
23	5	0.005	32	200	0.2	86.66	0.403212
24	4	0.005	16	256	0.3	86.41	0.414070

Figure B.3 The BERT (4 last layer concatenation) +BiLSTM model experiment results for finding optimal hyperparameters

APPENDIX C EXAMPLE OF AUGMENTED SENTENCES

Original Text = "my father had a stroke in the part of his brain right behind the eye that got avastin injection."

α= 0.3 L = 19 n= int(l) = int (0.3x19) = 5

Table C.1 Examples of applying EDA augmentation technique to a sample of the dataset

Data augmentation technique	Sentence length(l)	Augmented text
RS	19	"avastin father had the stroke in got behind his of brain right part a eye that the my injection."
SR	20	"my father indiana had a stroke in the brainiac middle part of his brain right virgule behind the eye that got avastin powerful injection."
RD	14	"father had a in the part of his right behind the eye that got"
RI	24	"my male parent had a stroke in the division of his brainpower good fundament the eye that got avastin injection."

Word_List = ["my" "father" "had" "a" "stroke" "in" "the" "part" "of" "his" "brain" "right" "behind" "the" "eye" "that" "got" "avastin" "injection."]

α= 0.33 m = 1/α=1/0.33=3

Table C.2 Examples of applying 4 proposed augmentation technique to a sample of the dataset

Data augmentation technique	Sentence length(l)	Augmented text
RPD	13	"my father a stroke the part his brain behind the that got injection."
DPR	13	"father had a in the part of his right the that got injection."
LCD	19	"my fath had a stro in the part of his bra rig behi the eye that got avas inject"
SWD (ch=2)	15	"father had stroke the part his brain right behind the eye that got avastin injection."

APPENDIX D RESEARCH MODELS CODES

The training python codes for the BERT+BiLSTM are shown as the following:

```
In [ ]:
##### Training #####

startA = datetime.now()
print("Now Time : ",startA)
show_confusion_after_any_epoch = True
show_classification_report_after_any_epoch = True
Digits = 4

validat_loss = 0
min_valid_loss = 1000
df_score = pd.DataFrame()

max_valid_accuracy = 0
max_valid_F1Score = 0

yLow_loss = 0.001
yHigh_loss = 2.0
yLow_accuracy = 0
yLow_f1_score = 0.0

Tr_f1Score,Tr_RecallScore,Tr_PreciScore,Tr_Accure =[],[],[],[]
Va_f1Score,Va_RecallScore,Va_PreciScore,Va_Accure =[],[],[],[]

min_valid_loss_epoch = 0
max_valid_F1Score_epoch = 0

epoch = 1
Average = 'micro'
mode_score = 'strict'
Flag_Traning_Countinue = True
Epochs_Max = 60
while Flag_Traning_Countinue == True:
    start = datetime.now()
    print ("Epoch :", epoch,"                             ")

    #------------ train part ------------------------
    train_loss, valid_loss = [], []
    predictions = []
    true_labels = []
    data2 =[]
    ii = 0
    model.train()
    for data,target, mask in trainloader:

        optimizer.zero_grad()

        target = torch.tensor(target, dtype=torch.long)

        if n_gpu == 1:
            data = data.to('cuda')
            target = target.to('cuda')
            mask = mask.to('cuda')

        output= model(data,mask)
        loss = loss_function(output.flatten(start_dim=0, end_dim=1),target.flatten())

        ##  backward propagation
        loss.backward()
        ##  weight optimization
        torch.nn.utils.clip_grad_norm_(parameters=model.parameters(), max_norm=1.0)
        optimizer.step()

        train_loss.append(loss.item())

        label_ids = target.to('cpu').numpy()
        true_labels.extend(label_ids)
        output2 = output.detach().cpu().numpy()

        data_tmp = data.to('cpu').numpy()
        data2.extend(data_tmp)

        predictions.extend([list(p) for p in np.argmax(output2, axis=2)])
        ii +=1
        print("train batch:  %.f / %.f " %(ii,len(train_ids)/BatchSize),end='\r')
    print("                                ",end='\r')
    training_loss = np.mean(train_loss)

    train_pred_tags = [[tags_name[l_i] for ii,l_i in enumerate(l)  if data2[i][ii] != 0
and data2[i][ii] != 101
                        and data2[i][ii] != 102] for i,l in enumerate(predictions)]
    train_tags = [[tags_name[l_i] for ii,l_i in enumerate(l)  if data2[i][ii] != 0 and
data2[i][ii] != 101
                        and data2[i][ii] != 102] for i,l in enumerate(true_labels)]

    Train_Acc = accuracy_score(train_tags, train_pred_tags)*100.0
    train_F1Score = f1_score(train_tags, train_pred_tags,average=Average,scheme=IOB2, m
ode= mode_score )
    train_precision_score = precision_score(train_tags, train_pred_tags,average=Average
,scheme=IOB2, mode= mode_score)
    train_recall_score = recall_score(train_tags, train_pred_tags,average=Average,schem
e=IOB2, mode= mode_score)

    ## --------------------evaluation part -------------------------
    model.eval()
    predictions = []
    true_labels = []
    ii = 0
    data2 =[]
    for data, target, mask in validloader:
        target = torch.tensor(target, dtype=torch.long)
        if n_gpu == 1:
            data = data.to('cuda')
            target = target.to('cuda')
            mask = mask.to('cuda')

        with torch.no_grad():
            output= model(data,mask)

        loss = loss_function(output.flatten(start_dim=0, end_dim=1),target.flatten())

        valid_loss.append(loss.item())

        label_ids = target.to('cpu').numpy()
        true_labels.extend(label_ids)
        output2 = output.detach().cpu().numpy()

        data_tmp = data.to('cpu').numpy()
        data2.extend(data_tmp)
        predictions.extend([list(p) for p in np.argmax(output2, axis=2)])
        ii += 1
        print("valid batch:  %.f / %.f " %(ii,len(valid_ids)/BatchSize),end='\r')

    validat_loss = np.mean(valid_loss)
    valid_pred_tags = [[tags_name[l_i] for ii,l_i in enumerate(l)  if data2[i][ii] != 0 
and data2[i][ii] != 101
                        and data2[i][ii] != 102] for i,l in enumerate(predictions)]
    valid_tags = [[tags_name[l_i] for ii,l_i in enumerate(l)  if data2[i][ii] != 0 and 
data2[i][ii] != 101
                        and data2[i][ii] != 102] for i,l in enumerate(true_labels)]

    Valid_Acc = accuracy_score(valid_tags,valid_pred_tags)*100.0
    valid_F1Score = f1_score(valid_tags,valid_pred_tags,average=Average,scheme=IOB2, mo
de= mode_score )
    valid_precision_score = precision_score(valid_tags,valid_pred_tags,average=Average,
scheme=IOB2, mode= mode_score)
    valid_recall_score = recall_score(valid_tags,valid_pred_tags,average=Average,scheme
=IOB2, mode= mode_score)

    #----------- Svae model  in Max F1 Validation------------------

    if valid_F1Score > max_valid_F1Score:
        torch.save(model,'./model/Valid_Best_F1_Score_Model.pt')
        max_valid_F1Score = valid_F1Score
        max_valid_F1Score_epoch = epoch

    # ------------------- Report & Graph -------------------------
    if epoch == 1:
        temp = pd.DataFrame({
                             'Tr_accur':[0],
                             'Va_accur':[0],
                             'Tr_loss' : [0],
                             'Va_loss' : [0],
                             'Tr_F1':[0],
                             'Va_F1':[0],
                             'Tr_precis':[0],
                             'Va_precis':[0],
                             'Tr_recall':[0],
                             'Va_recall':[0]
                             })
        df_score = df_score.append(temp, ignore_index = True)
    temp = pd.DataFrame({
                             'Tr_accur':[Train_Acc],
                             'Va_accur':[Valid_Acc],
                             'Tr_loss' : [training_loss],
                             'Va_loss' : [validat_loss],
                             'Tr_F1':[train_F1Score],
                             'Va_F1':[valid_F1Score],
                             'Tr_precis':[train_precision_score],
                             'Va_precis':[valid_precision_score],
                             'Tr_recall':[train_recall_score],
                             'Va_recall':[valid_recall_score]
                             })
    df_score = df_score.append(temp, ignore_index = True)

    print (Fore.BLUE +"            Accuracy |     Loss    |   Precision |    Recall    | 
F1-score   |"+Fore.RESET)
    print('Training  :  %.2f %% |    %.4f    |    %.4f    |    %.4f    |    %.4f    |'
          %(Train_Acc,training_loss,train_precision_score,train_recall_score,train_F1S
core))
```

```
    print('Validation:  %.2f %% |    %.4f    |    %.4f    |    %.4f    |    %.4f    |'
           %(Valid_Acc,validat_loss,valid_precision_score,valid_recall_score,valid_F1Sc
ore))

    plot_loss_accuracy(df_score,0, Epochs_Max, yLow_loss,yHigh_loss, yLow_accuracy, 1)
    plot_F1_Score(df_score,0, Epochs_Max, yLow_f1_score, 1)

    if show_confusion_after_any_epoch :
        print('\nTraining-------------------------------------------------------------
----------------------')
        Make_Confusion_Graph_Tabel(sum(train_tags,[]), sum(train_pred_tags,[]), Labels=
TagLabel, Font_Scale = 1.4,
                                                                     Prefix = True, Show
Percent= True, Sum_O_Zero= True)
        print('\n\nValidation---------------------------------------------------------
------------------------')
        Make_Confusion_Graph_Tabel(sum(valid_tags,[]), sum(valid_pred_tags,[]), Labels=
TagLabel, Font_Scale = 1.4,
                                                                     Prefix = True, Show
Percent= True, Sum_O_Zero= True)
        print('-----------------------------------------------------------------------
--------------------')

    if show_classification_report_after_any_epoch:
        print('\nTraining-------------------------------------------------------------
----------------------')
        print(classification_report(train_tags, train_pred_tags, digits = Digits,scheme
=IOB2, mode= mode_score))

        print('\n\nValidation---------------------------------------------------------
------------------------')
        print(classification_report(valid_tags,valid_pred_tags, digits = Digits,scheme=
IOB2, mode= mode_score))
        print('-----------------------------------------------------------------------
----------------------')

    end = datetime.now()
    print("")
    print("Time : ", (end - start))

    print("_____________________________________________________________________________
____________________________")

#------------------------- Trainning Countine Condition Check ---------------------
    if validat_loss < min_valid_loss :
        min_valid_loss = validat_loss
        min_valid_loss_epoch = epoch

    if epoch - min_valid_loss_epoch > 5 :
        Flag_Traning_Countinue = False
        Epochs_Max = epoch
    else:
        epoch += 1
#-----------------------------------------------------------------------------------
plot_loss_accuracy(df_score,0, Epochs_Max, yLow_loss,yHigh_loss, yLow_accuracy, 1)
plot_F1_Score(df_score,0, Epochs_Max, yLow_f1_score, 1)

print("Sum Times : ", (end - startA))
```