Multilingual Hate Speech Detection

A thesis submitted in partial fulfillment of the requirements for the award of the degree of

B.Tech

in

Computer Science and Engineering

By

Satwick Sen Sarma (Roll No. 11200118037)

Samarpan Mandal (Roll No. 11200118040)

Munsi Afif Aziz (Roll No. 11200118045)

Adrish Kumar Ghosh (Roll No. 11200118062)

Project Guide :*Prof. Brijesh Srivastava*

COMPUTER SCIENCE AND ENGINEERING
GOVERNMENT COLLEGE OF ENGINEERING AND LEATHER TECHNOLOGY
SALT LAKE SECTOR III – 700096

Maulana Abul Kalam Azad University of Technology

June 2022

In memory of

Dr. A. P. J. Abdul Kalam "Missile Man"

Former President of India

BONAFIDE CERTIFICATE

This is to certify that the project titled **Multilingual Hate Speech Detection** is a bonafide record of the work done by

Satwick Sen Sarma (Roll No. 11200118037)

Samarpan Mandal (Roll No. 11200118040)

Munsi Afif Aziz (Roll No. 11200118045)

Adrish Kumar Ghosh (Roll No. 11200118062)

in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering** of the **GOVERNMENT COLLEGE OF ENGINEERING AND TECHNOLOGY, KOLKATA**, during the year 2021-22.

Prof. Brijesh Srivastava	**Prof. Shantanu Halder**
Guide	Head of the Department

Project Viva-voce held on: **14.6.2022**

Internal Examiner	**External Examiner**

Abstract

Detection of hate speech in user-generated online content has become an issue of increasing importance in recent years and the urgent need for effective countermeasures have drawn significant investment from governments, companies, and researchers. Text classification for online content is a bit challenging task due to the natural language complexity and hastily generated online user microblogs including a plethora of informality and mistakes.A large number of methods have been developed for automated hate speech detection online. A large number of methods have been developed for automated hate speech detection online. This aims to classify textual content into non-hate or hate speech, in which case the method may also identify the targeting characteristics (i.e., types of hate, such as race, and religion) in the hate speech. This work introduces a system to classify tweets into two classes (i.e., Non Hate-Offensive, Hate and Offensive) for Sub-task A and three fine-grained classes(i.e., Hate speech, Offensive, Profane) for Sub-task B in English and German language.In both of these tasks, we have implemented state of the art NLP architecture - Transformers to get an semantic encoding of the texts which was later evaluated by the model for classification. We perform the task by taking advantage of transfer learning models. We observed that the pre-trained BERT model and the multilingual-BERT model gave the best results and have used the same in our model. Our method is evaluated on a collection of hate speech datasets based on Twitter.

Keywords : Hate speech, classification, neural network, BERT, transfer learning, deep learning, natural language processing

ACKNOWLEDGEMENTS

We would like to express our deepest gratitude to the following people for guiding us through this course and without whom this project and the results achieved from it would not have reached completion.

Prof. Brijesh Srivastava, Assistant Head of the Department, Department of Computer Science and Engineering, for helping us and guiding us in the course of this project. Without his guidance, we would not have been able to successfully complete this project. His patience and genial attitude is and always will be a source of inspiration to us.

Prof. Shantanu Halder, the Head of the Department, Department of Computer Science and Engineering, for allowing us to avail the facilities at the department.

We are also thankful to the faculty and staff members of the Department of Computer Science and Engineering, our individual parents and our friends for their constant support and help.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1
Introduction

The widespread adoption of social media platforms has made it possible for users to express their opinions easily in a manner that is visible to a huge audience. These platforms provide a large step forward for freedom of expression. Social media has been used extensively for various purposes, such as advertising, business, news, etc. The idea of allowing users to post anything at any time on social media contributed to the existence of inappropriate content on social media. As a result, these platforms become a fertile environment for this type of content. Hate speech is the most common form of destructive content on social media, and it can come in the form of text, photographs, or video. It is defined as an insult directed at a person or group based on characteristics such as color, gender, race, sexual orientation, origin, nationality, religion, ethinicity or other characteristics (Weber, 2009). Hate speech poses a significant threat to communities, either by instilling hatred in young people against others or by instigating criminal activity or violence against others.

Social media users often employ abbreviations and ordinary words(not hateful) to express their hate intent implicitly that known as code words to evade from being detected (e.g., using Google to refer to dark skin people), which adds extra difficulties to detect hate speech. Many studies have proposed machine learning models to handle this problem by utilizing a wide range of features set and machine learning algorithms for classification. These methods often utilize features that require considerable effort and time to be extracted, such as text-based, profile-based, and community-based features. Other studies have worked on linguistic-based features (e.g., word frequency) and deep learning for classification , or distributional based features (e.g., word embed-ding) and machine learning classifier.

1.1 Motivation

Hate speech on the internet is on the rise around the world, with approximately 60% of the global population (4:54 billion) using social media to communicate (Ltd, 2020). According to studies, approximately 53 percent of Americans have encountered online harassment and hatred (League, 2019). This score is 12 points higher than the findings of a similar survey performed in 2017 (Duggan, 2017). According to Clement (2019), 21%. of students frequently encounter hate speech on social

media. Thus, the detection of hate content on social media is an essential and necessary requirement for social media platforms. Social media providers work hard to get rid of this content for a safer social environment. Detecting hateful content is considered one of the challenging NLP tasks as the content might target/attack individuals or groups based on various characteristics using different hate terms and phrases.
Offensive language such as insulting, hurtful, derogatory or obscene content directed towards people might suppress meaningful discussions. As there are no restrictions on expressing peoples opinions on such platforms, it might lead to the defaming of personalities. So it is the platform's responsibility to restrain such content. Hate speech mainly involves discriminating against people based on religion, community, race, nationality, gender or any other identity factors. Even though manual moderation of hate speech is always precise, it isn't recommended considering the huge volumes of data that is being pumped into social media. So there is a constant need for automated techniques to suppress such hateful content where the ages of all the groups are exposed to . As we have seen advances in computing capabilities, machine learning algorithms have gained their importance in tasks that involve understanding natural language.
The main obstacle with hate speech is, it is difficult to classify based on a single sentence because most of the hate speech has context attached to it, and it can morph into many different shapes depending on the context. Another obstacle is that humans cannot always agree on what can be classified as hate speech. Hence it is not very easy to create a universal machine learning algorithm that would detect it. Also, the datasets used to train models tend to "reflect the majority view of the people who collected or labeled the data". To perform hate speech detection, a corpus in the targeted language is required. In essence, a corpus is a body of text acquired from various sources, with labels assigned to them across one or more dimensions . The corpus plays a vital role in the model building process, as it is used to train models to perform specific tasks, which in this case, is to classify the body of texts into hate speech or non-hate speech. As the literature on hate speech detection is mostly focused on the English language, most of the corpora available in the Literature is in English . This results in a major impediment in the field of multilingual hate speech detection,as corpora for non-English languages are limited or unavailable.

1.2 Defining Hate Speech

There is no universal definition of hate speech, perhaps as argued by Berger (2018) because of the far-right's proximity to political power. In general terms, hate

speech can be seen as the communicative production of human inferiority. It is a heterogeneous phenomenon that does not necessarily always involve attitudes of hatred. In legal terms, the definition of hate speech depends on the regulations in each individual country. This is a challenge in light of removing online content. For example,the Encyclopedia of the American Constitution defines hate speech as "any communication that disparages a person or a group on the basis of some characteristic such as race, color, ethnicity, gender, sexual orientation, nationality, religion, or other characteristic". It is rarely penalized in the US, where freedom of speech is absolute and protected by the First Amendment . Exceptions can include defamation, fighting words ("personal epithets hurled face-to-to face at another individual which are likely to cause the average addressee to fight"), threats of unlawful harm, and incitement to violence or other unlawful actions.
In Germany, hate speech is more likely to be penalized. Its legislation is representative of the EU approach to hate speech regulation, which counterbalances freedom of expression with values such as human dignity . More concretely, in case of a conflict, US legislation tends to give precedence to freedom of speech while EU nations tend to give precedence to human dignity. Additionally, EU legislation also penalizes hate speech that targets groups, whereas US legislation is more restricted to penalizing speech that targets individuals . This is also reflected in the EU's Code of Conduct on countering illegal hate speech online, which defines hate speech as "the public incitement to violence or hatred directed to groups or individuals on the basis of certain characteristics, including race, colour, religion, descent and national or ethnic origin" German legislation in particular shows a "heightened sensitivity toward individual dignity and the potential harms of violating these values", which evolved from past incidences during the Nazi regime. This includes, specifically, the prohibition of Holocaust denial. In general, §5(1) of the German Grundgesetz protects freedom of speech and prohibits censorship, but it is overruled when personal dignity is in danger Infringements of personality rights are illegal and specified in German criminal law, such as incitement to criminal behavior, incitement of the masses , insult, defamation, slander, coercion, and intimidation. Also relevant is defamation or slander of a politician . Tweets qualifying as an offence as per StGB can be legally pursued. It is this kind of illegal hate speech (as well as fake news) that the Network Enforcement Act , effective in Germany as of October 1, 2017, pertains to. Reported illegal content must now be systematically reviewed within 24 hours. One type of content that is problematic here is satire (irony, ridicule), for which various cases of overblocking have been observed. A common argument against NetzDG is that it constitutes censorship, by restricting freedom of speech. However, NetzDG only fosters the removal of content that was not protected by freedom of speech even before its introduction.

Many instances of online group-focused enmity are currently in limbo between legally acceptable and illegal. For example, while StGB clearly differentiates between simply spreading defamatory information, and knowing that this information is incorrect (slander), the distinction is harder to make in reality. With respect to European hate speech legislation, "the application and interpretation of the existing criminal provisions on 'hate speech' is generally inconsistent" . We would argue to understand hate speech beyond its purely legal application, not only because of the lack of a clear-cut distinction between legal and illegal, but also because content moderation in social networks extends beyond what is illegal. For example, Facebook is known to apply its own rules, which only in part correspond to what should be removed by NetzDG. Hate speech is now widely studied and considered a problem not so much because of the (relatively few) illegal cases, but because it is an indicator of society's polarization. The analysis thus includes "'hate speech' that is lawful [...], but which nevertheless raises concerns in terms of intolerance and discrimination"– tweets that are not illegal but 'merely' offensive, i.e., constituting either an insult or abuse. According to Ruppenhofer, Siegel, Wiegand , this means messages that convey disrespect and contempt, by ascribing qualities to people that portray them as unworthy or unvalued (insult), or constitute group-based offense by ascribing a social identity to a person that is judged negatively by society or part of society (abuse). Many tweets in the dataset involve a tendentiously framed dissemination of news, as people interpret news by connecting events to personal experience and world views. By framing we mean 'select[ing] some aspects of perceived reality and mak[ing] them more salient in a communicating text'. For example, a crime committed by a single refugee may be framed as something that all refugees do all the time. This interpretation is often explicitly stated, as in "Dieser Mehrfach-Straftäter ist der Beweis: Multikulti funktioniert... Nicht!" ('this repeat offender proves that multiculturalism does not work'). Here, one criminal act is used to imply that multiculturalism automatically leads to an increased crime rate. The reported event may either be real news or fake news which falls into a very seperate domain,so we restrict ourselves here.

1.3 Main Achievements

With our increasing diversity, ethinicity, culture, political views, sexual orientation, beliefs and heavy usages of social media platforms day in day out people are abused and getting open to hateful allegations. Thus it creates a very massive problem and all the platform giants are looking forward to a more efficient and automated hate speech detector. Our work is just a descendent of a long line of work of researchers work on this same topic from very early. Our objective was to detect hate speech in more than one language and we successfully did this by detecting hate speech in both German and English language. The efficiency and complexity will be described later as we go on, but for now the same model works for both German and English with slight changes. Thanks to the huge amount of data generated by applications like Facebook, Twitter etc were able to have a very efficient model. We tried our model to be as flexible and as simple as possible which makes the model very likely to extend to new enhancements.

1.4 About the Report

This section gives a overview of the report. Till now we discussed our motivation, how we define the hate speech to implement in a mathematical model. The rest of the report goes as follows :

- **Background** The Background section covers past work on hate speech detection and evolution of different models. It also covers Sentiment Analysis and Emotion Analysis
- **Abstract Model** The Abstract Model section talks about the formulation of the problem and gives a thorough description of the transformer model. And then it gives us the idea of how we mapped the real world problem mathematically.
- **Project Methodologies** The project methodologies covers the collection and processing of the data and design of the model using coding paradigms.
- **Conclusion** Conclusion as it is named wrap up the over all report and possible future work scenarios with improvements.
- **Code Attachments** Code attachments is the documentation of the coding paradigms and its output.

CHAPTER 2
Background

This section gives an overview of hate speech detection and provides information for both featuires and classifiers.

2.1 Overview of Hate speech detection and the classifiers

Several research have attempted to solve the problem of detecting hate speech in general by differentiating hate and non-hate speech. Others have tackled the issue of recognizing certain types of hate speech, such as anti-religious hate speech, jihadist, sexist and racist,there are also ethnicity based hate speech ,colour related like Nigga, monkey; some are also based on body shaming like fat ass,fat bitch etc. Several platforms have been used to collect datasets from various online resources such as websites or online forums (e.g., 4Chan, DailyStorm), or recent social media platforms (e.g., Twitter, Facebook). Hate speech has been applied] also on different languages (e.g., English, Arabic, German)

2.2 Word Embedding

Word embedding is a promi-nent natural language processing technique that seeks to convey the semantic meaning of a word. It provides a useful numerical description of the term based on its context. The words are represented by a dense vector that can be used in estimating the similarities between the words. The word is represented by an N-dimensional vector appropriate to represent the word meaning in a specific language. The word embedding has been widely used in many recent NLP tasks due to its efficiency such as text classification, document clustering , part of speech tagging, named entity recognition, sentiment analysis, and many other problems. The most common pretrained word embedding models are Google Word2Vec, Stanford GloVe and they are described as follow:

2.2.1 Word2Vec

Word2Vec is one of the most recently used word embedding models. It is provided by the Google research team. Word2Vec associates each word with a vector-based on its surrounding context from a large corpus. The training process for extracting the word vector has two types, the continuous bag of words model

(CBOW), which predicts the target word from its context, and the Skip-Gram model (SG), which predicts the target context from a given word. The feature vector of the word is manipulated and updated accord-ing to each context the word appears in the cor-pus. If the word embedding is trained well, similar words appear close to each other in the dimensional space. The word similarities between the words are measured by the cosine distance between their vectors. Google released a vector model called Google Word2Vec that has been trained on a massive corpus of over 100 billion words.

2.2.2 GloVe

Jeffrey Pennington on his work Glove: Global vectors for word representation provides another popular word embedding model named GloVe (Global Vectors for Word Representation). GloVe learns embeddings using an unsupervised learning algorithm that is trained on a corpus to create the distributional feature vectors. During the learning process, a statistics-based matrix is built to represent the words to words co-occurrence of the corpus. This matrix represents the word vectors. The learning process requires time and space for the matrix construction, which is a highly costly process. The difference between GloVe and Word2Vec is in the learning process, Word2Vec is a prediction based model, and GloVe is a count-based model. The GloVe is learned from Wikipedia, web data, Twitter, and each model is available with multiple vector dimensions

2.3 Bidirectional Long Short-Term Memory (BiLSTM)

In the paper Long Short Term Memory (LSTM), Sepp Hochreiter and Jurgen Schmidhuber proposed the idea, which is an enhanced version of the recurrent neural network, which is one of the deep learning models that is designed to capture information from a sequence of information. It differs from the feed-forward neural network in that it has a backward connection. RNN suffers from a vanishing gradient problem that happens when the weights are not updated anymore due to the small value of the received from error function in respect to the current weights in the iteration. The value is vanishing in very long sequences and becomes close to zero. This problem stops RNN from training. LSTM solves this problem by adding an extra interaction cell to preserve long sequence dependencies. Thus, LSTM saves data for long sequences, but it saves the data only from left to right. However, to save sequence data from both directions, a Bidirectional LSTM is used. BiLSTM consist of two LSTM, one process the data from left to right and the other in opposite direction then concatenates and flattens both forward and backward LSTM to improve the knowledge of the surrounding context.

2.4 BERT Pre-trained Language Model

Bidirectional Encoder Representations from Transformers (BERT) proposed in 2018 is a language model trained on very huge data based on contextual representations. BERT consists of feature extraction layers, which consist of word embedding and layer for the model (e.g., Classification, Question Answering, Named Entity Recognition). BERT is the most recent language model and provides state of the art results in comparison to other language models for various NLP tasks. BERT training procedure of word embedding differs from other word embedding models. It creates a bidirectional representation of words that may be learned from both left and right directions. Word embed-ding approaches like Word2Vec and GloVe only examine one direction (either left to right or right to left), resulting in static word representations that do not alter with context. If the word's meaning varies depending on the context, GloVe and Word2Vec map the word to only one embedding vector. As a result, Word2Vec and GloVe are referred to as context-free models. BERT is also different from previous language models (e.g., ELMo stands for Embeddings from Language Models) in that it manipulates the context in all layers in both directions (left and right). Instead of shal-low combining processes such as concatenating, it use cooperatively conditioning to combine both left and right context. BERT is trained on Books Cor-pus (800M words) and English Wikipedia (2,500M words).

2.5 Emotion Analysis

Emotion analysis from text (EA) consists of mapping textual units to a predefined set of emotions, for instance basic emotions, as they have been proposed by Ekman (anger, fear, sadness, joy, disgust, surprise), the dimensional model of Plutchik (adding trust and anticipation),or the discrete model proposed by Shaver et al. [12] (anger, fear, joy, love, sadness, surprise). Great efforts have been conducted in the last years by the NLP community in a variety of emotion research tasks including emotion intensity prediction , emotion stimulus or cause detection [16, 17, 18, 19], or emotion classification [20, 21]. Studying patterns of human emotions is essential in various applications such as the detection of mental disorders, social media mining, dialog systems, business intelligence, or e-learning. An important application is the detection of HOF, since it is inextricably linked to the emotional and psychological state of the speaker [22]. Negative emotions such as anger, disgust and fear can be conveyed in the form of HOF. For example, in the text "I am sick and tired of this stupid situation" the author feels angry and at the same time is using offensive language to express that emotion. Therefore, the detection of negative emotions can be a clue to detect this type of behavior on the web. An important aspect of EA is the creation of annotated corpora to train machine learning models. The availability of emotion corpora is highly fragmented, not only because of the different emotion theories, but also because emotion classification appears to be genre- and domain-specific . We will limit the discussion of corpora in the following to those we use in this paper. The Twitter Emotion Corpus (TEC) was annotated with labels corresponding to Ekman's model of basic emotions (anger, disgust, fear, joy, sadness, and surprise) and consists of 21,051 tweets. It was automatically labeled with the use of hashtags that the authors selfassigned to their posts. The grounded emotions corpus created by Liu et al. is motivated by the assumption that emotions are grounded in contextual experiences. It consists of 2,557 instances, labeled by domain experts for the emotions of happiness and sadness. EmoEvent, on the contrary, was labeled via crowdsourcing via Amazon Mechanical Turk. It contains a total of 8,409 tweets in Spanish and 7,303 in English, based on events related to different topics such as entertainment, events, politics, global commemoration, and global strikes. The labels.

2.6 Sentiment Analysis

Sentiment analysis (SA) has emerged as one of the most well-known areas in NLP due to its significant implications in social media mining. Construed broadly, the task includes sentiment polarity classification, identifying the sentiment target or topic, opinion holder identification, and identifying the sentiment of one specific aspect (e.g., a product, topic, or organization) in its context sentence [7, 26]. Sentiment analysis is a stricter sense, i.e., polarity classification, is often modeled as a two-class (positive, negative) or three-class (positive, negative, neutral) categorization task. For instance, the opinionated expression "The movie was terrible, I wasted my time watching it" is clearly negative. A negative sentiment can be an indicator of the presence of offensive language, as previous studies have shown. Sentiment analysis and the identification of HOF share common discursive properties. Considering the example shown in previous section, "I am sick and tired of this stupid situation", in addition to expressing anger, conveys a negative sentiment along with the presence of expletive language targeted to a situation. Therefore, both sentiment and emotion features can be used as useful information in the NLP systems to benefit the task of HOF detection in social media. Note that sentiment analysis is not a "simplified" version of emotion analysis – sentiment analysis is about the expression of an opinion, while emotion analysis is about inferring an emotional private state of a user. These tasks are related, but at least to some degree complementary . Unlike EA, as SA classification is one of the most studied tasks due to its broader applications, a larger number of corpora annotated with sentiments is available, particularly from Twitter. For instance, one of the most well-known datasets is the Stanford Sentiment Treebank. It contains movie reviews in English from Rotten Tomatoes. Another popular dataset was released in SemEval 2016 for Task 4 is labeled with positive, negative, or neutral sentiments and includes a mixture of entities (e.g., Gaddafi, Steve Jobs), products (e.g., kindle, android phone), and events (e.g., Japan earthquake, NHL playoffs) [30]. The same year, another dataset was released in SemEval for Task 6, the Twitter stance and sentiment corpus which is composed of 4,870 English tweets labeled with positive and negative sentiments. For a more detailed overview, we refer the reader to recent surveys on the topic

CHAPTER 3
Abstract Model

3.1 Defining the Problem

After considering the potential of a sentence to be considered as a hate speech, we decided to fine tune it more. We used the simple divide and conquer technique to make our identification task easier. So ultimately our task is the given sentence falls under which categories which is discussed below.

Sub-task A focuses on Hate speech and Offensive language identification offered for English and German Languages. Sub-task A is coarse-grained binary classification in which participating system are required to classify tweets into two class, namely: Hate and Offensive (HOF) and Non- Hate and offensive (NOT).

- (NOT) Non Hate-Offensive - Post does not contain any Hate speech, profane, offensive content.
- ((HOF) Hate and Offensive - Post contains Hate, offensive, and profane content.

Sub-task B discriminates between Hate, profane and offensive posts.This subtask is a fine-grained classification offered for English and German. Hate-speech and offensive posts from the sub-task A are further classified into three categories.

- (HATE) Hate speech :- Posts under this class contain Hate speech content.
- (OFFN) Offensive :- Posts under this class contain offensive content.
- (PRFN) Profane :- Post contains profane words. This typically concerns the usage of swearwords and cursing.

3.2 From Real World to Abstract World

The Machine Learning algorithms cannot understand the classification rules from the raw text. These algorithms need numerical features to understand classification rules. Hence, in text classification one of the key steps is feature engineering. This step is used for extracting the key features from raw text and representing the extracted features in numerical form. To identify or classify user-generated content, text features indicating hate must be extracted. Obvious features are individual words or phrases (n-grams, i.e., sequence of n consecutive words). To improve the matching of features, words can be stemmed to obtain only the root removing morphological differences. The bag-of-words assumption is commonly used in text categorization. Under this assumption, a post is represented simply as a set of words or n-grams without any ordering. Some deep learning architectures, such as recurrent and transformer neural networks, challenge the bag-of-words assumption by modeling the ordering of the words by processing over a sequence of word embeddings [Kuncoro A, Dyer C, Hale J, Yogatama D, Clark S, Blunsom P. LSTMs Can Learn Syntax-Sensitive Dependencies Well, But Modeling Structure Makes Them Better. In: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Melbourne, Australia: Association for Computational Linguistics; 2018. p. 1426–1436.]. In this model, pretrained BERT tokenizer has been implemented for feature extraction from the texts. The input embeddings from the BERT tokenizer are the sum of the token embeddings, the segmentation embeddings and the position embeddings[Devlin J, Chang MW, Lee K, Toutanova K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:181004805 [cs]. 2018.].

The input embeddings serve as input for the fine-tuned pre-trained Bidirectional Encoder Representations from Transformers (BERT) which acts as the encoding layer of the model for retrieves semantic relationships among words . The output from the final transformer layer from the BERT transformer stack is given as input to a fully connected neural network, and the softmax activation function is applied to the neural network to classify the given sentence.

3.3 The Transformer Model

The core idea behind a transformer model is self-attention—the ability to attend to different positions of the input sequence to compute a representation of that sequence. Transformer creates stacks of self-attention layers and is explained below in the sections Scaled dot product attention and Multi-head attention.

A transformer model handles variable-sized input using stacks of self-attention layers instead of RNNs or CNNs. This general architecture has a number of advantages:

- It makes no assumptions about the temporal/spatial relationships across the data. This is ideal for processing a set of objects .
- Layer outputs can be calculated in parallel, instead of a series like an RNN.

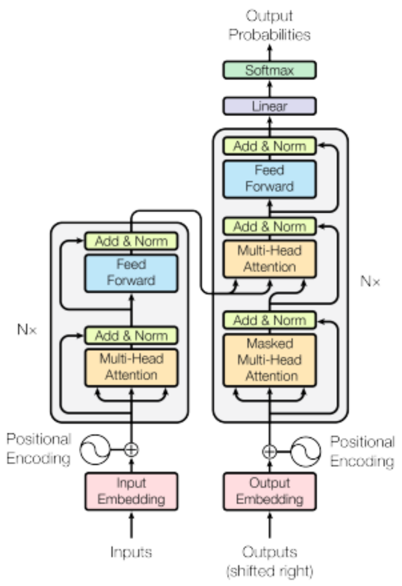
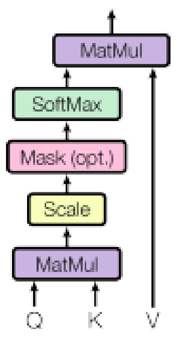


Figure 3.1: The Transformer - model architecture.



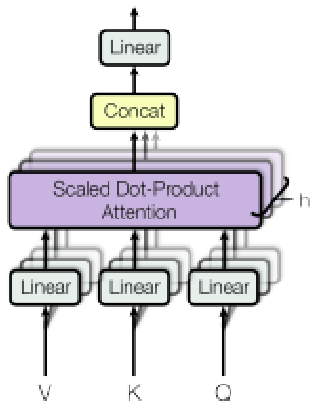


Figure 3.2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

- Distant items can affect each other's output without passing through many RNN-steps, or convolution layers . It can learn long-range dependencies. This is a challenge in many sequence tasks.

Now let's walk through figure 3.1:

- The lefthand part is the encoder. Just like earlier, it takes as input a batch of sentences represented as sequences of word IDs (the input shape is [batch size, max input sentence length]), and it encodes each word into a 512-dimensional representation (so the encoder's output shape is [batch size, max input sentence length, 5121). Note that the top part of the encoder is stacked N times (in the paper. N-6).

- The righthand part is the decoder. During training, it takes the target sentences input (also represented as a sequence of word IDs), shifted one time step to the right (Le, a start-of-sequence token is inserted at the beginning). It also receives the outputs of the encoder (ie, the arrows coming from the left side). Note that the top part of the decoder is also stacked N times, and the encoder stack's fin outputs are led to the decoder at each of these N levels. Just like earlier, the decoder outputs a probability for each possible next word, at each time step (its is output shape is [*batch size, max output sentence length, vocabulary length*])

- During inference, the decoder cannot be fed targets, so we feed it the previously output words (starting with a start-of-sequence token). So the model needs to be called repeatedly, predicting one more word at every round (which is fed to the decoder at the next round, until the end-of-sequence token is output).

- Looking more closely, you can see that you are already familiar with most components: there are two embedding layers, **5 x N skip connections**, each of the followed by a layer of **normalization layer**, **2 x N "Feed Forward" modules** that are composed of two dense layers each (the first one using the ReLU activation function, the second with no activation function), and finally the output layer is a dense layer using the softmax activation function. All of these layers are time distributed, so each word is treated independently of all the others. But how can we translate a sentence by only looking at one word at a time? Well, that's where the new components come in:

  1. The encoder's **Multi-Head Attention layer** encodes each word's relationship with every other word in the same sentence, paying more attention to the most relevant ones. For example, the output of this layer for the word "Queen" in the sentence "They welcomed the Queen of the United Kingdom will depend on all the words in the sentence, but it will probably pay more attention to the words "United" and "Kingdom" than to the words "They" or "welcomed." This attention mechanism is called self-attention (the sentence is paying attention to itself). We will discuss exactly how it works shortly. The decoder's Masked Multi-Head Attention layer does the same thing, but each word is only allowed to attend to words located before it. Finally, the decoders upper Multi-Head Attention layer is where the decoder pays attention to the words in the input sentence. For example, the decoder will probably pay close attention to the word "Queen" in the input sentence when it is about to output this word's translation.

2. The **positional embeddings** are simply dense vectors (much like word embed dings) that represent the position of a word in the sentence. The positional embedding is added to the word embedding of the nth word in each sentence This gives the model access to each word's position, which is needed because the Multi-Head Attention layers do not consider the order or the position of the words: they only look at their relationships. Since all the other layers are time-distributed, they have no way of knowing the position of each word (either relative or absolute). Obviously, the relative and absolute word positions are important, so we need to give this information to the Transformer somehow, and positional embeddings are a good way to do this.

The next sections covers both these novel components of the Transformer architecture starting with the positional embeddings.

3.3.1 Positional Embeddings

A positional embedding is a dense vector that encodes the position of a word within a sentence: the positional embedding is simply added to the word embedding of the word in the sentence. These positional embeddings can be learned by the model. but in the paper the authors preferred to use fixed positional embeddings, defined using the sine and cosine functions of different frequencies. The positional embed ding matrix P is defined in the next equation and represented at the bottom of Figure 3.3 (transposed), where $P_{p,i}$, is the ith component of the embedding for the word located at the pth position in the sentence.

$$P_{p,2i} = sin(p/10000^{2i/d})$$
$$P_{p,2i+1} = cos(p/10000^{2i/d})$$

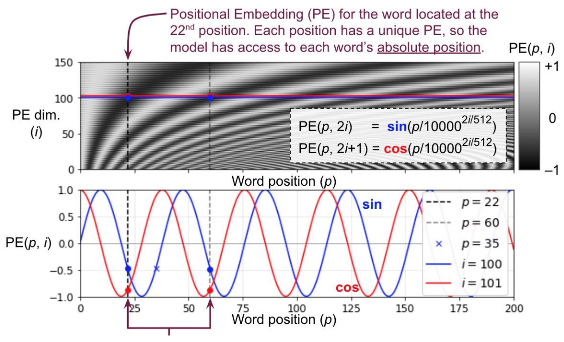


Figure 3.3: Sine/cosine positional embedding matrix(transposed top) with a focus on two values of i(bottom)

This solution gives the same performance as learned positional embeddings d can extend to arbitrarily long sentences, which is why it's favored. After the p

embeddings are added to the word embeddings, the rest of the model has acces the absolute position of each word in the sentence because there is a unique tional embedding for each position (e.g., the positional embedding for the word in ted at the 22nd position in a sentence is represented by the vertical dashed line at the bottom left of Figure 3.3, and you can see that it is unique to that position More over, the choice of oscillating functions (sine and cosine) makes it possible for the model to learn relative positions as well. For example, words located 38 words an (e.g., at positions $p = 22$ and $p = 60$) always have the same positional embedding ues in the embedding dimensions i = 100 and i = 101, as you can see in Figure 3.3 This explains why we need both the sine and the cosine for each frequency: if we only used the sine (the blue wave at i= 100), the model would not be able to distinguish positions p = 25 and p = 35 (marked by a cross).

There is no Positional Embedding layer in TensorFlow, but it is easy to create one For efficiency reasons, we precompute the positional embedding matrix in the constructor (so we need to know the maximum sentence length, max_steps, and the number of dimensions for each word representation, max_dims). Then the call!) method crops this embedding matrix to the size of the inputs, and it adds it to the inputs. Since we added an extra first dimension of size 1 when creating the positional embedding matrix, the rules of broadcasting will ensure that the matrix gets added to every sentence in the inputs

3.3.2 Multihead Attention Layer

To understand how a Multi-Head Attention layer works, we must first understand the Scaled Dot-Product Attention layer, which it is based on. Let's suppose the encoder analyzed the input sentence ″They played chess, and it managed to understand that the word ″They″ is the subject and the word ″played is the verb, so it encoded this information in the representations of these words, Now suppose the decoder has already translated the subject, and it thinks that it should translate the verb next. For this, it needs to fetch the verb from the input sentence. This is analog to a dictionary lookup: it's as if the encoder created a dictionary subject″: ″They″, ″verb″ ″played″, ...and the decoder wanted to look up the value that corresponds to the key ″verb However, the model does not have discrete tokens to represent the keys (like ″subject″ or ″verb″); it has vectorized representations of these concepts (which it learned during training), so the key it will use for the lookup (called the query) will not perfectly match any key in the dictionary. The solution is to compute a similarity measure between the query and each key in the dictionary, and then use the softmax function to convert these similarity scores to weights that add up to 1. If the key that represents the verb is by far the most similar to the query, then that key's weight will be close to 1. Then the model can compute a weighted sum of the corresponding values, so if the weight of the ”verb” key is close to 1, then the weighted sum will be very close to the representation of the word ”played.” In short, you can think of this whole process as a differentiable dictionary lookup. The similarity measure used by the Transformer is just the dot product, like in Luong attention. In fact, the equation is the same as for Luong attention, except for a scaling factor. The equation is shown in Equation, in a vectorized form.

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_{keys}}})$$

In this equation:

- Q is a matrix containing one row per query. Its shape is $[n_{queries}d_{Keys}]$ where $n_{queries}$ is the number of queries and d_{Keys} is the number of dimensions of each query and each key.
- K is a matrix containing one row per key. Its shape is $[n_{keys}d_{Keys}]$ where n_{keys} is the number of keys and values.
- V is a matrix containing one row per value. Its shape is $[n_{keys}d_{values}]$ where d_{values} is the number of each value.
- The shape of QK^T is $[n_{queries}n_{Keys}]$ it contains one similarity score for each query/key pair. The output of the softmax function has the same shape, but rows sum up to 1. The final output has a shape of $[n_{queries}d_{values}]$: there is one row per query, where each row represents the query result (a weighted sum of the values).
- he scaling factor scales down the similarity scores to avoid saturating the w max function, which would lead to tiny gradients.
- It is possible to mask out some key/value pairs by adding a very large negative value to the corresponding similarity scores, just before computing the softmax This is useful in the Masked Multi-Head Attention layer.

n the encoder, this equation is applied to every input sentence in the batch, with Q K. and V all equal to the list of words in the input sentence (so each word in the sentence will be compared to every word in the same sentence, including itself). Similarly, in the decoder’s masked attention layer, the equation will be applied to every target sentence in the batch, with Q, K, and V all equal to the list of words in the tar get sentence, but this time using a mask to prevent any word from comparing itself to words located after it (at inference time the decoder will only have access to the words it already output, not to future words, so during training we must mask out future output tokens). In the upper attention layer of the decoder, the keys K and values V are simply the list of word encodings produced by the encoder, and the queries Q are the list of word encodings produced by the decoder.The keras. layers. Attention layer implements Scaled Dot-Prod attention, efficiently applying above equation to multiple sentences in a batch. Its are just like Q, K, and V, except with an extra batch dimension (the first dimension)

3.3.3 Implementing the Transformer Model to the problem

Here, we used the pre-trained Bidirectional Encoder Representations from Transformers (BERT) transformer model for hate speech and offensive content detection. BERT's model architecture is a multi-layer bidirectional Transformer encoder based on the original implementation described in [Vaswani et al. (2017)] and released in the tensor2tensor library.BERT is conceptually simple and empirically powerful. It obtains new state-of-the-art results on eleven natural language processing tasks, including pushing the GLUE score to 80.5% (7.7% point absolute improvement), MultiNLI accuracy to 86.7% (4.6% absolute improvement), SQuAD v1.1 question answering Test F1 to 93.2 (1.5 point absolute improvement) and SQuAD v2.0 Test F1 to 83.1 (5.1 point absolute improvement). It shows clear benefits of using pre-trained models (trained on huge datasets) and transfer learning independent of the downstream tasks - in our case for Hate Speech Detection and Classification. The [CLS] output produced by the 12^{th} and final layer of $\text{BERT}_{\text{Base}}$ is the encoded output of the input text. The output represents a rich encoding of the text in the form of a vector .This output acts as the input for the classifier feed forward neural network which classifies the sentences into the given categories.

CHAPTER 4
Project Methodologies and Results

4.1 Data Collection and Data Processing

In this work, we collected publicly available hate speech tweets dataset.In this dataset, the tweets are labeled into three distinct classes, namely, hate speech(HATE), profane(PRFN), and offensive(OFFN) if the first label is HOF(i.e. Hate Speech) and none if the first label is NOT(i.e. Not Hate Speech). The dataset was subsequently sampled from Twitter and partially from Facebook for English and German languages. The tweets were acquired using hashtags and keywords that contained offensive content. The statistics of datasets are given in the following Table.

Table 4.1: English Dataset

	NONE	**HATE**	**PRFN**	**OFFN**
Train Sentences	1852	158	1377	321
Test Sentences	414	25	293	82

Table 4.2: German Dataset

	NONE	**HATE**	**PRFN**	**OFFN**
Train Sentences	1700	146	387	140
Test Sentences	378	24	88	36

In our model we implemented the Bert Tokenizer for feature extraction . The BERT tokenizer applies WordPiece tokenization [Devlin et al., 2018]. An example of word piece tokenization is that "composes" will produce two tokens, namely "compose" and "##s". The double # character indicates that this token is part of the previous word. This ability of BERT tokenizer allows it to be very descriptive taking into account its very small vocabulary size.The words that cannot be represented by the BERT tokenizer are resulting to UNK tokens, which are being removed from the resulting text[Impact of Tokenization, Pretraining Task, and Transformer Depth on Text Ranking Jaap Kamps Nikolaos Kondylidis David Rau University of Amsterdam].The total number of tokens kept in the sequence is fixed at a maximum length of 512.If the text input is longer than the Sequence length, the end of the text will be ignored. If the text input is smaller, the sequence will be padded with PAD tokens.

The first token of every sequence is always a special classification token ([CLS]). [SEP] token has to be inserted at the end of a single input. The first step is to use the BERT tokenizer to first split the word into tokens. Then, we add the special tokens needed for classification (these are [CLS] at the first position, and [SEP] at the end of the sentence). For a given token, its input representation is constructed by summing

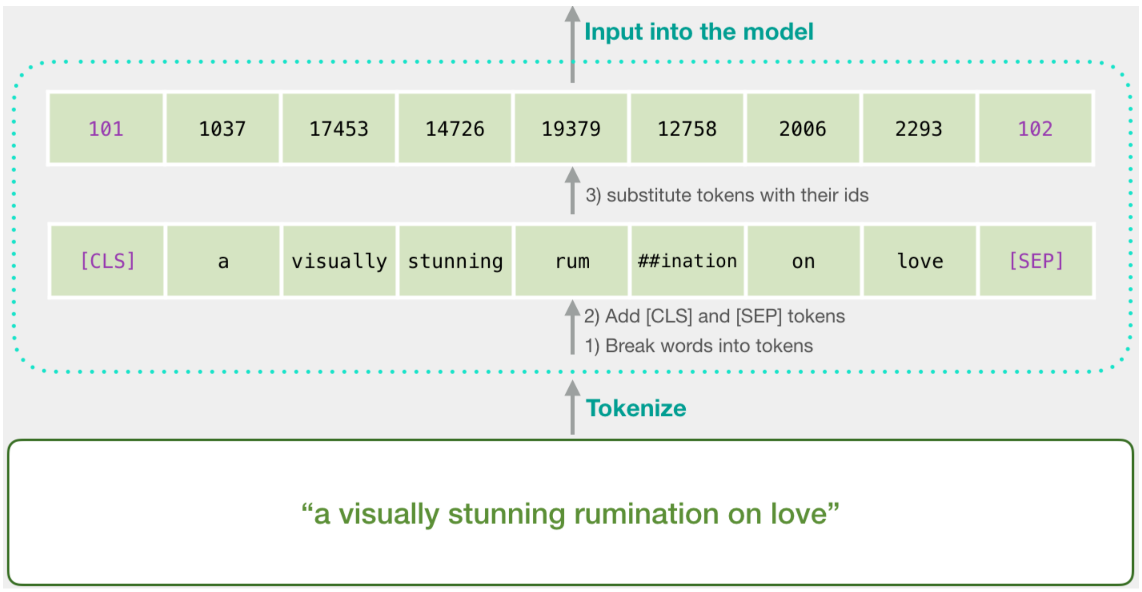

Input into the model
101
1037
17453
14726
19379
12758
2006
2293
102
3) substitute tokens with their ids
[CLS]
a
visually
stunning
rum
##ination
on
love
[SEP]
2) Add [CLS] and [SEP] tokens
1) Break words into tokens
Tokenize
"a visually stunning rumination on love"

Figure 4.1: BERT Tokenization

the corresponding token, segment, and position embeddings. A visualization of tokenization can be seen in Figure 4.1.

4.2 Developing the Classifier Model

Here, we use the pre-trained BERT transformer model for hate speech and offensive content detection. Figure 4.2 depicts the abstract view of BERT model that is used for hate speech detection and offensive language identification. Bidirectional Encoder Representations from Transformers (BERT) is a transformer Encoder stack trained on the large English corpus. It has 2 models, $\text{BERT}_{\text{Base}}$ and $\text{BERT}_{\text{Large}}$ .In this work, we denote the number of layers (i.e., Transformer blocks) as L, the hidden size as H, and the number of self-attention heads as A.The model architectures are: **BERTBASE** (L=12, H=768, A=12, Total Parameters=110M) and **BERTLARGE** (L=24, H=1024,A=16, Total Parameters=340M). These model sizes have a large number of transformer layers. The $\text{BERT}_{\text{Base}}$ version has 12 transformer layers and the $\text{BERT}_{\text{Large}}$ has 24. These also have larger feed-forward networks with 768 and

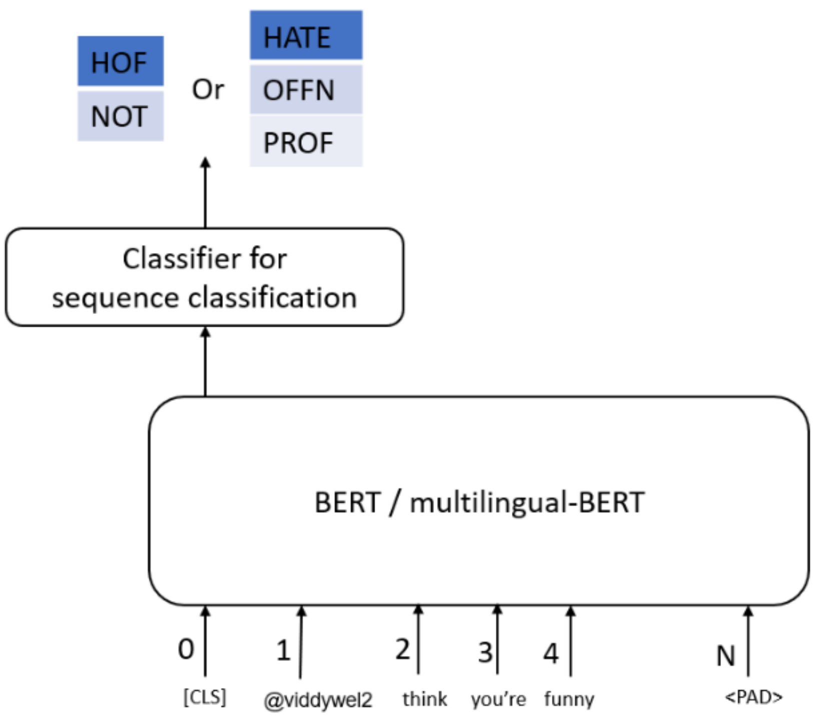


Figure 4.2: Block Diagram of Model for sequence classification on Hate Speech Data

1024 hidden representations, and attention heads are 12 and 16 for the respective models.Like the vanilla transformer model [8], BERT takes a sequence of words as input. Each layer applies self-attention, passes its results through a feed-forward network, and then hands it off to the next encoder. Embeddings from B have 768 hidden units. The BERT configuration model takes a sequence of words/tokens at a maximum length of 512 and produces an encoded representation of dimensionality 768. The pre-trained BERT models have a better word representation as they are trained on a large Wikipedia and book corpus. Recent empirical improvements due to transfer learning with language models have demonstrated that rich, unsupervised pre-training is an integral part of many language understanding systems.As the pre-trained BERT model is trained on generic corpora, we need to fine-tune the model for the downstream tasks.For finetuning, the BERT model is first initialized with the pre-trained parameters, and all of the parameters are fine-tuned using labeled data from the downstream tasks. During fine-tuning, the pre-trained BERT model parameters are updated when trained on the labeled hate speech and offensive content dataset. When fine-tuned on the downstream sentence classification task, changes are applied to the $\text{BERT}_{\text{Base}}$ configuration. In this architecture, only the [CLS] (classification) token output provided by BERT is used. The [CLS] output is the output of the 12th transformer encoder with a dimensionality of 768. It is given as input to a fully connected neural network, and the softmax activation function.

For the implementation, we used the transformers library provided by HuggingFace. The HuggingFace transformers package is a python library providing pre-trained and configurable transformer models useful for a variety of NLP tasks. It contains the pre-trained BERT and multilingual BERT, and other models suitable for downstream tasks. As the implementation environment, we use the PyTorch library that supports GPU processing. The BERT models were run on NVIDIA graphics card . We trained our classifier with a batch size of 64 for 5 to 10 epochs based on our experiments. The dropout is set to 0.1, and the Adam optimizer is used with a learning rate of 2e-5. We used the hugging face transformers pre-trained BERT tokenizer for tokenization. We used the BertForSequenceClassification module provided by the HuggingFace library during finetuning and sequence classification.

4.3 Project Results: Performance and Efficiency

To compute the accuracy of our models we used *macro F1* accuracy. The *macro F1* can be defined as The macro-averaged F1 score (or macro F1 score) is computed by taking the arithmetic mean (aka unweighted mean) of all the per-class F1 scores, in this method all classes are treated equally regardless of their support values.

Table 4.3: macro F1 and Accuracy on English Subtasks A and B

	macro F1	**Accuracy**
Hate speech Detection	88.33%	88.33%
Offensive Content Identification	54.44%	81.57%

Table 4.4: macro F1 and Accuracy on German Subtasks A and B

	macro F1	**Accuracy**
Hate speech Detection	77.91%	82.51%
Offensive Content Identification	47.78%	80.42%

The results are tabulated in Tables 4.3 and 4.4 . We evaluated the performance of the method using macro F1 and accuracy. The BERT model performed well compared to other contemporary ELMO and SVM models. The SVM model stands for support vetor machine and the ELMO stands for Embeddings from Language Model, which is a word embedding method for representing a sequence of words as a corresponding sequence of vectors The SVM model has 81.56% and 45.54%

value for macro F1, and using both ELMO and SVM gives 82.43% and 45.94% for English Sub task A and B; and German Sub tasks A and B respectively. It shows the pre-trained BERT model's capability, which learnt better text representations from the generic data. The Bert Transfomer model is much more efficient than SVM and ELMO.The state of the art transformer architecture used in the BERT model helped the model learn better parameter weights in hate speech and offensive content detection.

CHAPTER 5
Conclusion

We used pre-trained bi-directional encoder representations using transformers (BERT) and multilingualBERT for hate speech and offensive content detection for English, German, and Hindi languages. We compared the BERT with other machine learning and neural network classification methods. Our analysis showed that using the pre-trained BERT and multilingual BERT models and finetuning it for downstream hate-speech text classification tasks showed an increase in macro F1 score and accuracy metrics compared to traditional word-based machine learning approaches. The given data has both hate speech and offensive content labeled for a given same sentence. It implies that both tasks are related. In such a scenario, we can use joint learning models to help obtain a strong relationship between the two tasks. Which, in turn, helps a deep joint classification model to understand the given datasets better. We observe that frozen embeddings give better results by retaining rich token representations from the pre-trained model. Moreover, averaging over sentence representations has helped the model in understanding the context better while trying to classify the current tweet. There are many scopes for future work in this project , for example to extend this model to detect sarcasm, which is not possible for our model, but the flexibility gives us the advantage of doing so.

APPENDIX A
CODE ATTACHMENTS

English_Task_A

June 13, 2022

[1]:
```
import torch

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

[2]:
```
import pandas as pd
import math
from sklearn import preprocessing

#2020 train and test files

file = "hasoc_2020_en_train_new_a.xlsx"
file_test = "english_test_1509.csv"


df_train = pd.read_excel(file,index_col=0)

df_train = df_train.dropna()

df_test = pd.read_csv(file_test)

task = 'task1'
task_2019 = 'task_1'

#2019 datasets also

file_2019_1 = pd.read_csv("2019/english_dataset/english_dataset/
 ↪hasoc2019_en_test-2919.tsv",sep='\t')
file_2019_2 = pd.read_csv("2019/english_dataset/english_dataset/english_dataset.
 ↪tsv",sep="\t")


sentences_2019_1 = list(file_2019_1['text'].values)
sentences_2019_2 = list(file_2019_2['text'].values)
```

```python
labels_2019_1 = list(file_2019_1[task_2019].values)
labels_2019_2 = list(file_2019_2[task_2019].values)

print(len(df_train))
print(df_train.head())

total_sentences = list(df_train['text'].values)
total_labels = list(df_train[task].values)

total_sentences.extend(sentences_2019_1)
total_sentences.extend(sentences_2019_2)

total_labels.extend(labels_2019_1)
total_labels.extend(labels_2019_2)

test_sentences = list(df_test['text'].values)
test_labels = list(df_test[task].values)

def clean_text(sentences):
    for index,line in enumerate(sentences):
        if "\n" in line:
            sentences[index] = line.replace("\n","")
    return sentences

total_sentences = clean_text(total_sentences)
test_sentences = clean_text(test_sentences)

def clean_labels(labels):
    new_list= []
    for value in labels:
        new_list.append(value.strip())
    return new_list

total_labels = clean_labels(total_labels)
test_labels = clean_labels(test_labels)

le = preprocessing.LabelEncoder()
le.fit(total_labels)
encoded_labels = le.transform(total_labels)
encoded_test_labels = le.transform(test_labels)
print(set(encoded_labels))

print(len(total_sentences),len(encoded_labels),len(test_sentences),len(encoded_test_labels))

print(df_test)
```

```
3708
```

```
                                                                    text task1  \
tweet_id
1123757263427186690  hate wen females hit ah nigga with tht bro  ,…   HOF
1123733301397733380  RT @airjunebug: When you're from the Bay but y…   HOF
1123734094108659712  RT @DonaldJTrumpJr: Dear Democrats: The Americ…   NOT
1126951188170199049  RT @SheLoveTimothy: He ain’t on drugs he just …   HOF
1126863510447710208  RT @TavianJordan: Summer ‘19 I’m coming for yo…   NOT

                    task2                  ID
tweet_id
1123757263427186690  PRFN  hasoc_2020_en_2574
1123733301397733380  PRFN  hasoc_2020_en_3627
1123734094108659712  NONE  hasoc_2020_en_3108
1126951188170199049  PRFN  hasoc_2020_en_3986
1126863510447710208  NONE  hasoc_2020_en_5152
{0, 1}
10713 10713 814 814
                tweet_id                                               text  \
0    1130081762154090497  RT @delmiyaa: Samini resetting the show and mo…
1    1130048316807491584           @Swxnsea how do you know that he’s left?
2    1123657766143504386  Tried to get Divock Origi on a free seeing as …
3    1126782963042013186  RT @nutclusteruwu: that…is yalls stupid whi…
4    1130159113529434113  &amp; IT DID. But a bitch got big girls things…
..                   …                                               …
809  1127061607433900032  RT @nytmike: At least twice in the past month,…
810  1123685826074951681  @ThreeDailey Omg he’s so gross! This just turn…
811  1126882552587927552  RT @RFERL: Two alleged #GRU agents and 12 othe…
812  1130294488859996160  RT @ShefVaidya: In Modi 2.0, I do hope the two…
813  1130111650780991493  RT @SSTweeps: #UAE BO #May16-18 wknd:\n\n#DeDe…

    task1 task2                  ID
0     NOT  NONE  hasoc_2020_en_2713
1     HOF  NONE  hasoc_2020_en_3874
2     NOT  NONE   hasoc_2020_en_281
3     HOF  PRFN  hasoc_2020_en_2026
4     HOF  PRFN  hasoc_2020_en_4023
..    …    …                  …
809   NOT  NONE  hasoc_2020_en_1212
810   HOF  OFFN  hasoc_2020_en_3435
811   NOT  NONE  hasoc_2020_en_3987
812   NOT  NONE  hasoc_2020_en_1176
813   HOF  NONE  hasoc_2020_en_1937

[814 rows x 5 columns]
```

[3]:
```python
from transformers import BertTokenizer
```

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

max_length = 0
for sentence in total_sentences:
    #print(sentence)
    length = len(tokenizer.tokenize(sentence))
    if length > max_length:
        max_length  = length
print("max token length is: ",max_length)
# max token length obtained is 50
# bert tokens are limited to 514 bytes.
```

```
max token length is:  399
```

[4]:
```
def encoder_generator(sentences,labels):

    sent_index = []
    input_ids = []
    attention_masks =[]

    for index,sent in enumerate(sentences):

        sent_index.append(index)

        encoded_dict = tokenizer.encode_plus(sent,
                                    add_special_tokens=True,
                                    max_length=128,
                                    pad_to_max_length=True,
                                    truncation = True,
                                    return_attention_mask=True,
                                    return_tensors='pt')
        input_ids.append(encoded_dict['input_ids'])

        attention_masks.append(encoded_dict['attention_mask'])

    input_ids = torch.cat(input_ids,dim=0)
    attention_masks = torch.cat(attention_masks,dim=0)
    labels = torch.tensor(labels)
    sent_index = torch.tensor(sent_index)

    return sent_index,input_ids,attention_masks,labels

sent_index,input_ids,attention_masks,encoded_label_tensors =␣
 ↪encoder_generator(total_sentences,encoded_labels)
test_sent_index,test_input_ids,test_attention_masks,encoded_test_label_tensors␣
 ↪= encoder_generator(test_sentences,encoded_test_labels)
```

```
print('Original: ', total_sentences[0])
print('Token IDs:', input_ids[0])
#print(encoded_label_tensors)
#print(encoded_test_label_tensors)
```

```
Original:  hate wen females hit ah nigga with tht bro  , I’m tryna make u my la
sweety , fuck ah bro
Token IDs: tensor([  101,  5223, 19181,  3801,  2718,  6289,  9152, 23033,
2007, 16215,
         2102, 22953,   100,  1010,  1045,  1521,  1049,  3046,  2532,  2191,
         1057,  2026,  2474,  4086,  2100,  1010,  6616,  6289, 22953,   102,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0])
```

[5]:
```
from torch.utils.data import TensorDataset,random_split

dataset = TensorDataset(input_ids,attention_masks,encoded_label_tensors)
test_dataset =␣
 ↪TensorDataset(test_sent_index,test_input_ids,test_attention_masks,encoded_test_label_tensor

train_size = int(0.75*len(dataset))

val_size = len(dataset)-train_size

train_dataset,val_dataset = random_split(dataset,[train_size,val_size])

print('train data samples is {}'.format(len(train_dataset)))
print("valid data samples is {}".format(len(val_dataset)))
print("test data samples is {}".format(len(test_dataset)))
```

```
train data samples is 8034
valid data samples is 2679
test data samples is 814
```

[6]:
```
from torch.utils.data import DataLoader,RandomSampler,SequentialSampler

bs=8

train_data_loader = DataLoader(train_dataset,
                              sampler=RandomSampler(train_dataset),
                              batch_size=bs)
valid_data_loader = DataLoader(val_dataset,
                              sampler=SequentialSampler(val_dataset),
                              batch_size=bs)
test_data_loader = DataLoader(test_dataset,
                            sampler=SequentialSampler(test_dataset),
                            batch_size=bs)
```

[7]:
```
from transformers import BertForSequenceClassification, AdamW

model = BertForSequenceClassification.from_pretrained('bert-base-uncased',
                                                     num_labels=len(le.
 ↪classes_),
                                                     output_attentions=False,
                                                     output_hidden_states=False,
                                                     )
model.cuda()
```

```
Some weights of the model checkpoint at bert-base-uncased were not used when
initializing BertForSequenceClassification: ['cls.predictions.bias',
'cls.predictions.transform.dense.weight',
'cls.predictions.transform.dense.bias', 'cls.predictions.decoder.weight',
'cls.seq_relationship.weight', 'cls.seq_relationship.bias',
'cls.predictions.transform.LayerNorm.weight',
'cls.predictions.transform.LayerNorm.bias']
- This IS expected if you are initializing BertForSequenceClassification from
the checkpoint of a model trained on another task or with another architecture
(e.g. initializing a BertForSequenceClassification model from a
BertForPretraining model).
- This IS NOT expected if you are initializing BertForSequenceClassification
from the checkpoint of a model that you expect to be exactly identical
(initializing a BertForSequenceClassification model from a
BertForSequenceClassification model).
Some weights of BertForSequenceClassification were not initialized from the
model checkpoint at bert-base-uncased and are newly initialized:
['classifier.weight', 'classifier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
```

[7]:
```
BertForSequenceClassification(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0): BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (1): BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (2): BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (3): BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
```

```
  )
)
(4): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(5): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
```

```
)
(6): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(7): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
```

```
(8): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(9): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(10): BertLayer(
      (attention): BertAttention(
        (self): BertSelfAttention(
          (query): Linear(in_features=768, out_features=768, bias=True)
          (key): Linear(in_features=768, out_features=768, bias=True)
          (value): Linear(in_features=768, out_features=768, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (output): BertSelfOutput(
          (dense): Linear(in_features=768, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
      (intermediate): BertIntermediate(
        (dense): Linear(in_features=768, out_features=3072, bias=True)
      )
      (output): BertOutput(
        (dense): Linear(in_features=3072, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (11): BertLayer(
      (attention): BertAttention(
        (self): BertSelfAttention(
          (query): Linear(in_features=768, out_features=768, bias=True)
          (key): Linear(in_features=768, out_features=768, bias=True)
          (value): Linear(in_features=768, out_features=768, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (output): BertSelfOutput(
          (dense): Linear(in_features=768, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
      (intermediate): BertIntermediate(
        (dense): Linear(in_features=768, out_features=3072, bias=True)
      )
      (output): BertOutput(
        (dense): Linear(in_features=3072, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
  )
)
```

```
      (pooler): BertPooler(
        (dense): Linear(in_features=768, out_features=768, bias=True)
        (activation): Tanh()
      )
    )
    (dropout): Dropout(p=0.1, inplace=False)
    (classifier): Linear(in_features=768, out_features=2, bias=True)
  )
```

[8]:
```
optimizer = AdamW(model.parameters(),lr=2e-5,eps=1e-8)

from transformers import get_linear_schedule_with_warmup

epochs=10
total_steps = len(train_data_loader) * epochs

scheduler = get_linear_schedule_with_warmup(optimizer,
                                            num_warmup_steps=0,
                                            num_training_steps=total_steps)
```

[9]:
```
import numpy as np

def predictions_labels(preds,labels):
    pred = np.argmax(preds,axis=1).flatten()
    label = labels.flatten()
    return pred,label
```

[10]:
```
import random
import numpy as np
import time
from sklearn.metrics import classification_report,accuracy_score,f1_score

total_t0 = time.time()

seed_val = 42

random.seed(seed_val)
np.random.seed(seed_val)
torch.manual_seed(seed_val)
torch.cuda.manual_seed_all(seed_val)
```

[11]:
```
def categorical_accuracy(preds, y):
    """
    Returns accuracy per batch, i.e. if you get 8/10 right, this returns 0.8,␣
 ↪NOT 8
    """
    max_preds = preds.argmax(dim = 1, keepdim = True) # get the index of the␣
↪max probability
    correct = max_preds.squeeze(1).eq(y)
    return correct.sum() / torch.FloatTensor([y.shape[0]])

def predictions_labels(preds,labels):
    pred = np.argmax(preds,axis=1).flatten()
    label = labels.flatten()
    return pred,label
```

[12]:
```
def train():
    total_train_loss = 0
    total_train_acc = 0

    model.train() # set model in train mode for batchnorm and dropout layers in␣
↪bert model

    for step,batch in enumerate(train_data_loader):
        b_input_ids = batch[0].to(device)
        b_input_mask = batch[1].to(device)
        b_labels = batch[2].to(device)
        model.zero_grad()

        loss,logits = model(b_input_ids,
                            attention_mask=b_input_mask,
                            labels=b_labels.long())

        total_train_loss+=loss.item()
        total_train_acc+=categorical_accuracy(logits,b_labels).item()

        loss.backward()

        torch.nn.utils.clip_grad_norm_(model.parameters(),1.0)

        optimizer.step()

        scheduler.step() #go ahead and update the learning rate

    avg_train_loss = total_train_loss/len(train_data_loader)
    avg_train_acc = total_train_acc/len(train_data_loader)

    return avg_train_loss,avg_train_acc
```

[13]:
```
def evaluate():
    model.eval()

    total_eval_accuracy = 0
    total_eval_loss = 0
    number_of_eval_steps= 0

    all_true_labels = []
    all_pred_labels = []

    for batch in valid_data_loader:
        b_input_ids = batch[0].to(device)
        b_input_mask = batch[1].to(device)
        b_labels = batch[2].to(device)

        with torch.no_grad():

            loss, logits = model(b_input_ids,
                                 attention_mask= b_input_mask,
                                 labels = b_labels.long())
        total_eval_loss+=loss.item()

        logits = logits.detach().cpu().numpy()

        label_ids = b_labels.to('cpu').numpy()

        pred,true = predictions_labels(logits,label_ids)

        all_pred_labels.extend(pred)
        all_true_labels.extend(true)

    print(classification_report(all_pred_labels,all_true_labels))
    avg_val_accuracy = accuracy_score(all_pred_labels,all_true_labels)
    macro_f1_score = f1_score(all_pred_labels,all_true_labels,average='macro')

    avg_val_loss = total_eval_loss/len(valid_data_loader)

    print("accuracy = {0:.2f}".format(avg_val_accuracy))

    return avg_val_loss,avg_val_accuracy,macro_f1_score
```

[14]:
```
import time
def epoch_time(start_time, end_time):
    elapsed_time = end_time - start_time
    elapsed_mins = int(elapsed_time / 60)
    elapsed_secs = int(elapsed_time - (elapsed_mins * 60))
    return elapsed_mins, elapsed_secs
```

[15]:
```
epochs = 10

best_macro_f1 = float('0')
```

```python
for epoch in range(epochs):

    start_time = time.time()
    train_loss,train_acc = train()
    valid_loss,valid_acc,macro_f1 = evaluate()

    end_time = time.time()

    epoch_mins, epoch_secs = epoch_time(start_time, end_time)

    if macro_f1 > best_macro_f1:
        best_macro_f1 = macro_f1
        torch.save(model,'model_english_task_a.pt')

    print(f'Epoch: {epoch+1:02} | Epoch Time: {epoch_mins}m {epoch_secs}s')
    print(f'\tTrain Loss: {train_loss:.3f} | Train Acc: {train_acc*100:.2f}%')
    print(f'\t Val. Loss: {valid_loss:.3f} |  Val. Acc: {valid_acc*100:.2f}%')
```

```
              precision    recall  f1-score   support

           0       0.79      0.69      0.74      1245
           1       0.76      0.84      0.80      1434

   micro avg       0.77      0.77      0.77      2679
   macro avg       0.78      0.77      0.77      2679
weighted avg       0.77      0.77      0.77      2679

accuracy = 0.77
Epoch: 01 | Epoch Time: 3m 36s
        Train Loss: 0.518 | Train Acc: 75.25%
         Val. Loss: 0.470 |  Val. Acc: 77.27%
              precision    recall  f1-score   support

           0       0.71      0.77      0.74      1001
           1       0.86      0.81      0.83      1678

   micro avg       0.80      0.80      0.80      2679
   macro avg       0.78      0.79      0.79      2679
weighted avg       0.80      0.80      0.80      2679

accuracy = 0.80
Epoch: 02 | Epoch Time: 3m 34s
        Train Loss: 0.421 | Train Acc: 81.53%
         Val. Loss: 0.450 |  Val. Acc: 79.66%
              precision    recall  f1-score   support
           0       0.62      0.81      0.70       841
           1       0.90      0.78      0.83      1838

   micro avg       0.79      0.79      0.79      2679
   macro avg       0.76      0.79      0.77      2679
weighted avg       0.81      0.79      0.79      2679

accuracy = 0.79
Epoch: 03 | Epoch Time: 3m 33s
        Train Loss: 0.325 | Train Acc: 87.89%
         Val. Loss: 0.566 |  Val. Acc: 78.69%
              precision    recall  f1-score   support

           0       0.62      0.79      0.69       850
           1       0.89      0.77      0.83      1829

   micro avg       0.78      0.78      0.78      2679
   macro avg       0.75      0.78      0.76      2679
weighted avg       0.80      0.78      0.78      2679

accuracy = 0.78
Epoch: 04 | Epoch Time: 3m 32s
        Train Loss: 0.220 | Train Acc: 93.93%
         Val. Loss: 1.106 |  Val. Acc: 77.90%
              precision    recall  f1-score   support

           0       0.68      0.75      0.71       984
           1       0.85      0.79      0.82      1695

   micro avg       0.78      0.78      0.78      2679
   macro avg       0.76      0.77      0.76      2679
weighted avg       0.78      0.78      0.78      2679

accuracy = 0.78
Epoch: 05 | Epoch Time: 3m 29s
        Train Loss: 0.126 | Train Acc: 96.77%
         Val. Loss: 1.375 |  Val. Acc: 77.68%
              precision    recall  f1-score   support

           0       0.66      0.77      0.71       929
           1       0.86      0.79      0.82      1750

   micro avg       0.78      0.78      0.78      2679
   macro avg       0.76      0.78      0.77      2679
weighted avg       0.79      0.78      0.78      2679

accuracy = 0.78
Epoch: 06 | Epoch Time: 3m 30s
```

```
        Train Loss: 0.076 | Train Acc: 98.35%
         Val. Loss: 1.566 |  Val. Acc: 77.94%
              precision    recall  f1-score   support

           0       0.73      0.71      0.72      1124
           1       0.79      0.81      0.80      1555

   micro avg       0.77      0.77      0.77      2679
   macro avg       0.76      0.76      0.76      2679
weighted avg       0.76      0.77      0.77      2679

accuracy = 0.77
Epoch: 07 | Epoch Time: 3m 29s
        Train Loss: 0.059 | Train Acc: 98.76%
         Val. Loss: 1.639 |  Val. Acc: 76.56%
              precision    recall  f1-score   support

           0       0.71      0.71      0.71      1093
           1       0.80      0.80      0.80      1586

   micro avg       0.77      0.77      0.77      2679
   macro avg       0.76      0.76      0.76      2679
weighted avg       0.77      0.77      0.77      2679

accuracy = 0.77
Epoch: 08 | Epoch Time: 3m 40s
        Train Loss: 0.044 | Train Acc: 99.13%
         Val. Loss: 1.715 |  Val. Acc: 76.67%
              precision    recall  f1-score   support

           0       0.71      0.72      0.72      1071
           1       0.81      0.80      0.81      1608

   micro avg       0.77      0.77      0.77      2679
   macro avg       0.76      0.76      0.76      2679
weighted avg       0.77      0.77      0.77      2679

accuracy = 0.77
Epoch: 09 | Epoch Time: 3m 42s
        Train Loss: 0.029 | Train Acc: 99.29%
         Val. Loss: 1.889 |  Val. Acc: 77.19%
              precision    recall  f1-score   support

           0       0.72      0.71      0.71      1117
           1       0.79      0.81      0.80      1562

   micro avg       0.76      0.76      0.76      2679
   macro avg       0.76      0.76      0.76      2679
```

```
weighted avg       0.76      0.76      0.76      2679

accuracy = 0.76
Epoch: 10 | Epoch Time: 3m 42s
        Train Loss: 0.018 | Train Acc: 99.56%
         Val. Loss: 1.949 |  Val. Acc: 76.45%
```

[15]:
```
del model
import gc
gc.collect()

model = torch.load('model_english_task_a.pt')
model = model.to(device)
```

[16]:
```
def evaluate_test():
    model.eval()

    total_eval_accuracy = 0
    total_eval_loss = 0
    number_of_eval_steps= 0

    all_true_labels = []
    all_pred_labels = []

    all_sentence_id=[]

    for batch in test_data_loader:
        b_sentence_id = batch[0].to(device)
        b_input_ids = batch[1].to(device)
        b_input_mask = batch[2].to(device)
        b_labels = batch[3].to(device)

        sent_ids = b_sentence_id.to('cpu').numpy()
        all_sentence_id.extend(sent_ids)

        with torch.no_grad():

            loss, logits = model(b_input_ids,
                                 attention_mask= b_input_mask,
                                 labels = b_labels.long())
        total_eval_loss+=loss.item()

        logits = logits.detach().cpu().numpy()

        label_ids = b_labels.to('cpu').numpy()
        pred,true = predictions_labels(logits,label_ids)

        all_pred_labels.extend(pred)

        all_true_labels.extend(true)

    print(classification_report(all_pred_labels,all_true_labels))
    avg_val_accuracy = accuracy_score(all_pred_labels,all_true_labels)

    avg_val_loss = total_eval_loss/len(valid_data_loader)

    print("accuracy = {0:.2f}".format(avg_val_accuracy))

    return avg_val_loss,avg_val_accuracy,all_sentence_id,all_pred_labels

valid_loss,valid_acc,all_sentence_id,all_pred_labels = evaluate_test()
```

```
              precision    recall  f1-score   support

           0       0.86      0.91      0.88       402
           1       0.91      0.86      0.88       412

   micro avg       0.88      0.88      0.88       814
   macro avg       0.88      0.88      0.88       814
weighted avg       0.88      0.88      0.88       814

accuracy = 0.88
```

```
[ ]:
```

English_Task_B

June 13, 2022

[1]:
```
import torch

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

[2]:
```
import pandas as pd
import math
from sklearn import preprocessing

#2020 train and test files

file = "hasoc_2020_en_train_new_b.xlsx"
file_test = "english_test_1509.csv"


df_train = pd.read_excel(file,index_col=0)

df_train = df_train.dropna()

df_test = pd.read_csv(file_test)

task = 'task2'
task_2019 = 'task_2'

#2019 datasets also

file_2019_1 = pd.read_csv("2019/english_dataset/english_dataset/
 ↪hasoc2019_en_test-2919.tsv",sep='\t')
file_2019_2 = pd.read_csv("2019/english_dataset/english_dataset/english_dataset.
 ↪tsv",sep="\t")


sentences_2019_1 = list(file_2019_1['text'].values)
sentences_2019_2 = list(file_2019_2['text'].values)
```

```python
labels_2019_1 = list(file_2019_1[task_2019].values)
labels_2019_2 = list(file_2019_2[task_2019].values)

print(len(df_train))
print(df_train.head())

total_sentences = list(df_train['text'].values)
total_labels = list(df_train[task].values)

total_sentences.extend(sentences_2019_1)
total_sentences.extend(sentences_2019_2)

total_labels.extend(labels_2019_1)
total_labels.extend(labels_2019_2)

test_sentences = list(df_test['text'].values)
test_labels = list(df_test[task].values)

def clean_text(sentences):
    for index,line in enumerate(sentences):
        if "\n" in line:
            sentences[index] = line.replace("\n","")
    return sentences

total_sentences = clean_text(total_sentences)
test_sentences = clean_text(test_sentences)

def clean_labels(labels):
    new_list= []
    for value in labels:
        new_list.append(value.strip())
    return new_list

total_labels = clean_labels(total_labels)
test_labels = clean_labels(test_labels)

le = preprocessing.LabelEncoder()
le.fit(total_labels)
encoded_labels = le.transform(total_labels)
encoded_test_labels = le.transform(test_labels)
print(set(encoded_labels))

print(len(total_sentences),len(encoded_labels),len(test_sentences),len(encoded_test_labels))

print(df_test)
```

```
3708
```

```
                                                                     text task1  \
tweet_id
1123757263427186690  hate wen females hit ah nigga with tht bro   ,…   HOF
1123733301397733380  RT @airjunebug: When you're from the Bay but y…   HOF
1123734094108659712  RT @DonaldJTrumpJr: Dear Democrats: The Americ…   NOT
1126951188170199049  RT @SheLoveTimothy: He ain't on drugs he just …   HOF
1126863510447710208  RT @TavianJordan: Summer '19 I'm coming for yo…   NOT

                    task2                 ID
tweet_id
1123757263427186690  PRFN  hasoc_2020_en_2574
1123733301397733380  PRFN  hasoc_2020_en_3627
1123734094108659712  NONE  hasoc_2020_en_3108
1126951188170199049  PRFN  hasoc_2020_en_3986
1126863510447710208  NONE  hasoc_2020_en_5152
{0, 1, 2, 3}
10713 10713 814 814
                tweet_id                                               text  \
0    1130081762154090497  RT @delmiyaa: Samini resetting the show and mo…
1    1130048316807491584           @Swxnsea how do you know that he's left?
2    1123657766143504386  Tried to get Divock Origi on a free seeing as …
3    1126782963042013186  RT @nutclusteruwu: that…is yalls stupid whi…
4    1130159113529434113  &amp; IT DID. But a bitch got big girls things…
..                   …                                                …
809  1127061607433900032  RT @nytmike: At least twice in the past month,…
810  1123685826074951681  @ThreeDailey Omg he's so gross! This just turn…
811  1126882552587927552  RT @RFERL: Two alleged #GRU agents and 12 othe…
812  1130294488859996160  RT @ShefVaidya: In Modi 2.0, I do hope the two…
813  1130111650780991493  RT @SSTweeps: #UAE BO #May16-18 wknd:\n\n#DeDe…

    task1 task2                  ID
0     NOT  NONE  hasoc_2020_en_2713
1     HOF  NONE  hasoc_2020_en_3874
2     NOT  NONE   hasoc_2020_en_281
3     HOF  PRFN  hasoc_2020_en_2026
4     HOF  PRFN  hasoc_2020_en_4023
..    …    …                   …
809   NOT  NONE  hasoc_2020_en_1212
810   HOF  OFFN  hasoc_2020_en_3435
811   NOT  NONE  hasoc_2020_en_3987
812   NOT  NONE  hasoc_2020_en_1176
813   HOF  NONE  hasoc_2020_en_1937

[814 rows x 5 columns]
```

[3]:
```python
from transformers import BertTokenizer
```

```

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

max_length = 0
for sentence in total_sentences:
    #print(sentence)
    length = len(tokenizer.tokenize(sentence))
    if length > max_length:
        max_length  = length
print("max token length is: ",max_length)
# max token length obtained is 50
# bert tokens are limited to 514 bytes.
```

```
max token length is:  399
```

[4]:
```
def encoder_generator(sentences,labels):

    sent_index = []
    input_ids = []
    attention_masks =[]

    for index,sent in enumerate(sentences):

        sent_index.append(index)

        encoded_dict = tokenizer.encode_plus(sent,
                                             add_special_tokens=True,
                                             max_length=128,
                                             pad_to_max_length=True,
                                             truncation = True,
                                             return_attention_mask=True,
                                             return_tensors='pt')
        input_ids.append(encoded_dict['input_ids'])

        attention_masks.append(encoded_dict['attention_mask'])

    input_ids = torch.cat(input_ids,dim=0)
    attention_masks = torch.cat(attention_masks,dim=0)
    labels = torch.tensor(labels)
    sent_index = torch.tensor(sent_index)

    return sent_index,input_ids,attention_masks,labels

sent_index,input_ids,attention_masks,encoded_label_tensors =␣
 ↪encoder_generator(total_sentences,encoded_labels)
test_sent_index,test_input_ids,test_attention_masks,encoded_test_label_tensors␣
 ↪= encoder_generator(test_sentences,encoded_test_labels)
```

```
print('Original: ', total_sentences[0])
print('Token IDs:', input_ids[0])
#print(encoded_label_tensors)
#print(encoded_test_label_tensors)
```

```
Original:  hate wen females hit ah nigga with tht bro  , I’m tryna make u my la
sweety , fuck ah bro
Token IDs: tensor([  101,  5223, 19181,  3801,  2718,  6289,  9152, 23033,
2007, 16215,
         2102, 22953,   100,  1010,  1045,  1521,  1049,  3046,  2532,  2191,
         1057,  2026,  2474,  4086,  2100,  1010,  6616,  6289, 22953,   102,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0])
```

[5]:
```
from torch.utils.data import TensorDataset,random_split

dataset = TensorDataset(input_ids,attention_masks,encoded_label_tensors)
test_dataset =␣
 ↪TensorDataset(test_sent_index,test_input_ids,test_attention_masks,encoded_test_label_tensor

train_size = int(0.75*len(dataset))

val_size = len(dataset)-train_size

train_dataset,val_dataset = random_split(dataset,[train_size,val_size])

print('train data samples is {}'.format(len(train_dataset)))
print("valid data samples is {}".format(len(val_dataset)))
print("test data samples is {}".format(len(test_dataset)))
```

```
train data samples is 8034
valid data samples is 2679
test data samples is 814
```

[6]:
```
from torch.utils.data import DataLoader,RandomSampler,SequentialSampler

bs=8

train_data_loader = DataLoader(train_dataset,
                              sampler=RandomSampler(train_dataset),
                              batch_size=bs)
valid_data_loader = DataLoader(val_dataset,
                              sampler=SequentialSampler(val_dataset),
                              batch_size=bs)
test_data_loader = DataLoader(test_dataset,
                            sampler=SequentialSampler(test_dataset),
                            batch_size=bs)
```

[7]:
```
from transformers import BertForSequenceClassification, AdamW

model = BertForSequenceClassification.from_pretrained('bert-base-uncased',
                                                     num_labels=len(le.
 ↪classes_),
                                                     output_attentions=False,
                                                     output_hidden_states=False,
                                                     )
model.cuda()
```

```
Some weights of the model checkpoint at bert-base-uncased were not used when
initializing BertForSequenceClassification: ['cls.predictions.bias',
'cls.predictions.transform.dense.weight',
'cls.predictions.transform.dense.bias', 'cls.predictions.decoder.weight',
'cls.seq_relationship.weight', 'cls.seq_relationship.bias',
'cls.predictions.transform.LayerNorm.weight',
'cls.predictions.transform.LayerNorm.bias']
- This IS expected if you are initializing BertForSequenceClassification from
the checkpoint of a model trained on another task or with another architecture
(e.g. initializing a BertForSequenceClassification model from a
BertForPretraining model).
- This IS NOT expected if you are initializing BertForSequenceClassification
from the checkpoint of a model that you expect to be exactly identical
(initializing a BertForSequenceClassification model from a
BertForSequenceClassification model).
Some weights of BertForSequenceClassification were not initialized from the
model checkpoint at bert-base-uncased and are newly initialized:
['classifier.weight', 'classifier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
```

[7]:
```
BertForSequenceClassification(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0): BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (1): BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (2): BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (3): BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (4): BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (5): BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
```

```
    )
    (6): BertLayer(
      (attention): BertAttention(
        (self): BertSelfAttention(
          (query): Linear(in_features=768, out_features=768, bias=True)
          (key): Linear(in_features=768, out_features=768, bias=True)
          (value): Linear(in_features=768, out_features=768, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (output): BertSelfOutput(
          (dense): Linear(in_features=768, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
      (intermediate): BertIntermediate(
        (dense): Linear(in_features=768, out_features=3072, bias=True)
      )
      (output): BertOutput(
        (dense): Linear(in_features=3072, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (7): BertLayer(
      (attention): BertAttention(
        (self): BertSelfAttention(
          (query): Linear(in_features=768, out_features=768, bias=True)
          (key): Linear(in_features=768, out_features=768, bias=True)
          (value): Linear(in_features=768, out_features=768, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (output): BertSelfOutput(
          (dense): Linear(in_features=768, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
      (intermediate): BertIntermediate(
        (dense): Linear(in_features=768, out_features=3072, bias=True)
      )
      (output): BertOutput(
        (dense): Linear(in_features=3072, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
```

```
(8): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(9): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(10): BertLayer(
        (attention): BertAttention(
          (self): BertSelfAttention(
            (query): Linear(in_features=768, out_features=768, bias=True)
            (key): Linear(in_features=768, out_features=768, bias=True)
            (value): Linear(in_features=768, out_features=768, bias=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (output): BertSelfOutput(
            (dense): Linear(in_features=768, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (intermediate): BertIntermediate(
          (dense): Linear(in_features=768, out_features=3072, bias=True)
        )
        (output): BertOutput(
          (dense): Linear(in_features=3072, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
      (11): BertLayer(
        (attention): BertAttention(
          (self): BertSelfAttention(
            (query): Linear(in_features=768, out_features=768, bias=True)
            (key): Linear(in_features=768, out_features=768, bias=True)
            (value): Linear(in_features=768, out_features=768, bias=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (output): BertSelfOutput(
            (dense): Linear(in_features=768, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (intermediate): BertIntermediate(
          (dense): Linear(in_features=768, out_features=3072, bias=True)
        )
        (output): BertOutput(
          (dense): Linear(in_features=3072, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
    )
  )
```

```
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=768, out_features=4, bias=True)
)
```

[8]:
```
optimizer = AdamW(model.parameters(),lr=2e-5,eps=1e-8)

from transformers import get_linear_schedule_with_warmup

epochs=10
total_steps = len(train_data_loader) * epochs

scheduler = get_linear_schedule_with_warmup(optimizer,
                                            num_warmup_steps=0,
                                            num_training_steps=total_steps)
```

[9]:
```
import numpy as np

def predictions_labels(preds,labels):
    pred = np.argmax(preds,axis=1).flatten()
    label = labels.flatten()
    return pred,label
```

[10]:
```
import random
import numpy as np
import time
from sklearn.metrics import classification_report,accuracy_score,f1_score

total_t0 = time.time()

seed_val = 42

random.seed(seed_val)
np.random.seed(seed_val)
torch.manual_seed(seed_val)
torch.cuda.manual_seed_all(seed_val)
```

[11]:
```
def categorical_accuracy(preds, y):
    """
    Returns accuracy per batch, i.e. if you get 8/10 right, this returns 0.8,␣
 ↪NOT 8
    """
    max_preds = preds.argmax(dim = 1, keepdim = True) # get the index of the␣
↪max probability
    correct = max_preds.squeeze(1).eq(y)
    return correct.sum() / torch.FloatTensor([y.shape[0]])

def predictions_labels(preds,labels):
    pred = np.argmax(preds,axis=1).flatten()
    label = labels.flatten()
    return pred,label
```

[12]:
```
def train():
    total_train_loss = 0
    total_train_acc = 0

    model.train() # set model in train mode for batchnorm and dropout layers in␣
↪bert model

    for step,batch in enumerate(train_data_loader):
        b_input_ids = batch[0].to(device)
        b_input_mask = batch[1].to(device)
        b_labels = batch[2].to(device)
        model.zero_grad()

        loss,logits = model(b_input_ids,
                            attention_mask=b_input_mask,
                            labels=b_labels.long())

        total_train_loss+=loss.item()
        total_train_acc+=categorical_accuracy(logits,b_labels).item()

        loss.backward()

        torch.nn.utils.clip_grad_norm_(model.parameters(),1.0)

        optimizer.step()

        scheduler.step() #go ahead and update the learning rate

    avg_train_loss = total_train_loss/len(train_data_loader)
    avg_train_acc = total_train_acc/len(train_data_loader)

    return avg_train_loss,avg_train_acc
```

[13]:
```
def evaluate():
    model.eval()

    total_eval_accuracy = 0
    total_eval_loss = 0
    number_of_eval_steps= 0

    all_true_labels = []
    all_pred_labels = []

    for batch in valid_data_loader:
        b_input_ids = batch[0].to(device)
        b_input_mask = batch[1].to(device)
        b_labels = batch[2].to(device)

        with torch.no_grad():

            loss, logits = model(b_input_ids,
                                attention_mask= b_input_mask,
                                labels = b_labels.long())
        total_eval_loss+=loss.item()

        logits = logits.detach().cpu().numpy()

        label_ids = b_labels.to('cpu').numpy()

        pred,true = predictions_labels(logits,label_ids)

        all_pred_labels.extend(pred)
        all_true_labels.extend(true)

    print(classification_report(all_pred_labels,all_true_labels))
    avg_val_accuracy = accuracy_score(all_pred_labels,all_true_labels)
    macro_f1_score = f1_score(all_pred_labels,all_true_labels,average='macro')

    avg_val_loss = total_eval_loss/len(valid_data_loader)

    print("accuracy = {0:.2f}".format(avg_val_accuracy))

    return avg_val_loss,avg_val_accuracy,macro_f1_score
```

[14]:
```
import time
def epoch_time(start_time, end_time):
    elapsed_time = end_time - start_time
    elapsed_mins = int(elapsed_time / 60)
    elapsed_secs = int(elapsed_time - (elapsed_mins * 60))
    return elapsed_mins, elapsed_secs
```

[15]:
```
epochs = 10

best_macro_f1 = float('0')
```

```
for epoch in range(epochs):

    start_time = time.time()
    train_loss,train_acc = train()
    valid_loss,valid_acc,macro_f1 = evaluate()

    end_time = time.time()

    epoch_mins, epoch_secs = epoch_time(start_time, end_time)

    if macro_f1 > best_macro_f1:
        best_macro_f1 = macro_f1
        torch.save(model,'model_english_task_b.pt')

    print(f'Epoch: {epoch+1:02} | Epoch Time: {epoch_mins}m {epoch_secs}s')
    print(f'\tTrain Loss: {train_loss:.3f} | Train Acc: {train_acc*100:.2f}%')
    print(f'\t Val. Loss: {valid_loss:.3f} |  Val. Acc: {valid_acc*100:.2f}%')
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         2
           1       0.95      0.73      0.83      2009
           2       0.13      0.34      0.19        77
           3       0.81      0.77      0.79       591

   micro avg       0.73      0.73      0.73      2679
   macro avg       0.47      0.46      0.45      2679
weighted avg       0.90      0.73      0.80      2679

accuracy = 0.73
Epoch: 01 | Epoch Time: 3m 47s
	Train Loss: 0.822 | Train Acc: 70.22%
	 Val. Loss: 0.790 |  Val. Acc: 72.60%
              precision    recall  f1-score   support

           0       0.04      0.33      0.07        42
           1       0.93      0.74      0.83      1934
           2       0.22      0.33      0.27       132
           3       0.77      0.77      0.77       571

   micro avg       0.72      0.72      0.72      2679
   macro avg       0.49      0.54      0.48      2679
weighted avg       0.85      0.72      0.78      2679

accuracy = 0.72
Epoch: 02 | Epoch Time: 3m 47s
```

```
        Train Loss: 0.678 | Train Acc: 74.93%
         Val. Loss: 0.784 |  Val. Acc: 72.08%
              precision    recall  f1-score   support

           0       0.32      0.36      0.34       332
           1       0.84      0.77      0.81      1684
           2       0.17      0.33      0.22       104
           3       0.76      0.77      0.76       559

   micro avg       0.70      0.70      0.70      2679
   macro avg       0.52      0.56      0.53      2679
weighted avg       0.74      0.70      0.72      2679

accuracy = 0.70
Epoch: 03 | Epoch Time: 3m 51s
        Train Loss: 0.531 | Train Acc: 80.49%
         Val. Loss: 0.840 |  Val. Acc: 70.29%
              precision    recall  f1-score   support

           0       0.29      0.32      0.30       348
           1       0.83      0.78      0.80      1653
           2       0.21      0.32      0.25       133
           3       0.74      0.77      0.76       545

   micro avg       0.69      0.69      0.69      2679
   macro avg       0.52      0.54      0.53      2679
weighted avg       0.71      0.69      0.70      2679

accuracy = 0.69
Epoch: 04 | Epoch Time: 3m 46s
        Train Loss: 0.372 | Train Acc: 87.39%
         Val. Loss: 1.141 |  Val. Acc: 69.20%
              precision    recall  f1-score   support

           0       0.30      0.30      0.30       371
           1       0.81      0.79      0.80      1581
           2       0.21      0.28      0.24       148
           3       0.75      0.74      0.74       579

   micro avg       0.68      0.68      0.68      2679
   macro avg       0.52      0.53      0.52      2679
weighted avg       0.69      0.68      0.69      2679

accuracy = 0.68
Epoch: 05 | Epoch Time: 3m 41s
        Train Loss: 0.246 | Train Acc: 92.72%
         Val. Loss: 1.502 |  Val. Acc: 68.01%
              precision    recall  f1-score   support
           0       0.40      0.29      0.34       513
           1       0.75      0.81      0.78      1432
           2       0.20      0.30      0.24       128
           3       0.76      0.71      0.74       606

   micro avg       0.66      0.66      0.66      2679
   macro avg       0.53      0.53      0.52      2679
weighted avg       0.66      0.66      0.66      2679

accuracy = 0.66
Epoch: 06 | Epoch Time: 3m 34s
        Train Loss: 0.165 | Train Acc: 95.58%
         Val. Loss: 1.897 |  Val. Acc: 66.33%
              precision    recall  f1-score   support

           0       0.29      0.30      0.29       364
           1       0.82      0.79      0.80      1587
           2       0.23      0.31      0.27       151
           3       0.75      0.73      0.74       577

   micro avg       0.69      0.69      0.69      2679
   macro avg       0.52      0.53      0.53      2679
weighted avg       0.70      0.69      0.69      2679

accuracy = 0.69
Epoch: 07 | Epoch Time: 3m 35s
        Train Loss: 0.109 | Train Acc: 97.30%
         Val. Loss: 2.116 |  Val. Acc: 68.53%
              precision    recall  f1-score   support

           0       0.27      0.31      0.29       327
           1       0.83      0.78      0.81      1637
           2       0.21      0.28      0.24       149
           3       0.74      0.74      0.74       566

   micro avg       0.69      0.69      0.69      2679
   macro avg       0.51      0.53      0.52      2679
weighted avg       0.71      0.69      0.70      2679

accuracy = 0.69
Epoch: 08 | Epoch Time: 3m 34s
        Train Loss: 0.071 | Train Acc: 98.25%
         Val. Loss: 2.181 |  Val. Acc: 68.83%
              precision    recall  f1-score   support

           0       0.30      0.28      0.29       402
           1       0.79      0.79      0.79      1545
           2       0.22      0.28      0.25       158
           3       0.73      0.72      0.73       574

   micro avg       0.67      0.67      0.67      2679
   macro avg       0.51      0.52      0.51      2679
weighted avg       0.67      0.67      0.67      2679

accuracy = 0.67
Epoch: 09 | Epoch Time: 3m 33s
        Train Loss: 0.054 | Train Acc: 98.79%
         Val. Loss: 2.374 |  Val. Acc: 66.89%
              precision    recall  f1-score   support

           0       0.30      0.29      0.29       395
           1       0.81      0.79      0.80      1566
           2       0.24      0.29      0.26       165
           3       0.73      0.74      0.73       553

   micro avg       0.68      0.68      0.68      2679
   macro avg       0.52      0.53      0.52      2679
weighted avg       0.68      0.68      0.68      2679

accuracy = 0.68
Epoch: 10 | Epoch Time: 3m 34s
        Train Loss: 0.034 | Train Acc: 99.18%
         Val. Loss: 2.369 |  Val. Acc: 67.56%
```

[15]:
```
del model
import gc
gc.collect()

model = torch.load('model_english_task_b.pt')
model = model.to(device)
```

[16]:
```
def evaluate_test():
    model.eval()

    total_eval_accuracy = 0
    total_eval_loss = 0
    number_of_eval_steps= 0

    all_true_labels = []
    all_pred_labels = []

    all_sentence_id=[]

    for batch in test_data_loader:
        b_sentence_id = batch[0].to(device)
        b_input_ids = batch[1].to(device)
        b_input_mask = batch[2].to(device)
        b_labels = batch[3].to(device)

        sent_ids = b_sentence_id.to('cpu').numpy()
        all_sentence_id.extend(sent_ids)

        with torch.no_grad():

            loss, logits = model(b_input_ids,
                                  attention_mask= b_input_mask,
                                  labels = b_labels.long())
        total_eval_loss+=loss.item()

        logits = logits.detach().cpu().numpy()

        label_ids = b_labels.to('cpu').numpy()


        pred,true = predictions_labels(logits,label_ids)

        all_pred_labels.extend(pred)

        all_true_labels.extend(true)

    print(classification_report(all_pred_labels,all_true_labels))
    avg_val_accuracy = accuracy_score(all_pred_labels,all_true_labels)

    avg_val_loss = total_eval_loss/len(valid_data_loader)

    print("accuracy = {0:.2f}".format(avg_val_accuracy))

    return avg_val_loss,avg_val_accuracy,all_sentence_id,all_pred_labels

valid_loss,valid_acc,all_sentence_id,all_pred_labels = evaluate_test()
```

```
              precision    recall  f1-score   support

           0       0.12      0.15      0.13        20
           1       0.93      0.88      0.90       439
           2       0.21      0.57      0.30        30
           3       0.88      0.80      0.84       325

   micro avg       0.82      0.82      0.82       814
   macro avg       0.54      0.60      0.54       814
weighted avg       0.87      0.82      0.84       814
```

```
accuracy = 0.82
```

```
[ ]:
```

German_Task_A

June 13, 2022

[1]:
```
import torch

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

[2]:
```
import pandas as pd
import math
from sklearn import preprocessing

file = "hasoc_2020_de_train_new_a.xlsx"
file_test = "german_test_1509.csv"


df_train = pd.read_excel(file,index_col=0)

df_train = df_train.dropna()

df_test = pd.read_csv(file_test)

task = 'task1'
task_2019 = 'task_1'

#2019 datasets also

file_2019_1 = pd.read_csv("2019/german_dataset/german_dataset/
 ↪hasoc_de_test_gold.tsv",sep='\t')
file_2019_2 = pd.read_csv("2019/german_dataset/german_dataset/german_dataset.
 ↪tsv",sep="\t")


sentences_2019_1 = list(file_2019_1['text'].values)
sentences_2019_2 = list(file_2019_2['text'].values)

labels_2019_1 = list(file_2019_1[task_2019].values)
labels_2019_2 = list(file_2019_2[task_2019].values)
```

```
print(len(df_train))
print(df_train.head())

total_sentences = list(df_train['text'].values)
total_labels = list(df_train[task].values)

total_sentences.extend(sentences_2019_1)
total_sentences.extend(sentences_2019_2)

total_labels.extend(labels_2019_1)
total_labels.extend(labels_2019_2)

test_sentences = list(df_test['text'].values)
test_labels = list(df_test[task].values)

def clean_text(sentences):
    for index,line in enumerate(sentences):
        if "\n" in line:
            sentences[index] = line.replace("\n","")
    return sentences

total_sentences = clean_text(total_sentences)
test_sentences = clean_text(test_sentences)

def clean_labels(labels):
    new_list= []
    for value in labels:
        new_list.append(value.strip())
    return new_list

total_labels = clean_labels(total_labels)
test_labels = clean_labels(test_labels)

le = preprocessing.LabelEncoder()
le.fit(total_labels)
encoded_labels = le.transform(total_labels)
encoded_test_labels = le.transform(test_labels)
print(set(encoded_labels))

print(len(total_sentences),len(encoded_labels),len(test_sentences),len(encoded_test_labels))

print(df_test)
```

```
2373
                                                                   text task1  \
tweet_id
```

```
1133388798925189122  Deutsche rothaarige porno reife deutsche fraue…   NOT
1127134592517980161  RT @NDRinfo: Die deutsche Klimaaktivistin Luis…   NOT
1128897106171842560  @ruhrbahn jeden Morgen eine neue „Fahrzeugstör…   NOT
1123576753199484928  @Junge_Freiheit Die Inkas hatten sich schon dä…   NOT
1128743783393312768  RT @technosteron: leute die 'boar' schreiben l…   HOF

                    task2                ID
tweet_id
1133388798925189122  NONE  hasoc_2020_de_2684
1127134592517980161  NONE  hasoc_2020_de_1042
1128897106171842560  NONE   hasoc_2020_de_774
1123576753199484928  NONE   hasoc_2020_de_559
1128743783393312768  PRFN  hasoc_2020_de_1969
{0, 1}
7042 7042 526 526
                tweet_id                                              text  \
0    1129095874242650112    @derCarsti Boykottieren hört sich besser an.
1    1129004308396236800  RT @ibikus31: Es wird spekuliert, ob Merkel ei…
2    1130896929355907080  Hat #Hitler wirklich den Krieg in der Wüste ve…
3    1132251534329307136  RT @Beatrix_vStorch: #May tritt in UK unter Tr…
4    1124941869115498496  @justmeDoro Eher nicht. Das Gänse hauen wieder…
..                   …                                               …
521  1124809878546128897  RT @ChanMachtSo: SCHMERZEN!!!! Au!!! Mein Gehi…
522  1132433240000798720  Die Zerstörung der Grünen. https://t.co/SIYDJj…
523  1127366294255357958  RT @PParzival: "Antideutsche" Pseudo-linke Ide…
524  1124362090460975105                        Klug reden und dumm leben.
525  1131487097293103104  Wissen wir schon lange…hat das die Merkel e…

    task1 task2                ID
0     NOT  NONE  hasoc_2020_de_1053
1     NOT  NONE   hasoc_2020_de_671
2     NOT  NONE  hasoc_2020_de_2977
3     NOT  NONE  hasoc_2020_de_1746
4     NOT  NONE  hasoc_2020_de_2416
..    …    …                  …
521   NOT  NONE   hasoc_2020_de_486
522   NOT  NONE  hasoc_2020_de_3388
523   NOT  NONE  hasoc_2020_de_2745
524   NOT  NONE   hasoc_2020_de_236
525   NOT  NONE  hasoc_2020_de_2850

[526 rows x 5 columns]
```

[3]:
```
from transformers import BertTokenizer


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

```
max_length = 0
for sentence in total_sentences:
    #print(sentence)
    length = len(tokenizer.tokenize(sentence))
    if length > max_length:
        max_length  = length
print("max token length is: ",max_length)
# max token length obtained is 50
# bert tokens are limited to 514 bytes.
```

```
max token length is:  175
```

[4]:
```
def encoder_generator(sentences,labels):

    sent_index = []
    input_ids = []
    attention_masks =[]

    for index,sent in enumerate(sentences):

        sent_index.append(index)

        encoded_dict = tokenizer.encode_plus(sent,
                                             add_special_tokens=True,
                                             max_length=max_length,
                                             pad_to_max_length=True,
                                             truncation = True,
                                             return_attention_mask=True,
                                             return_tensors='pt')
        input_ids.append(encoded_dict['input_ids'])

        attention_masks.append(encoded_dict['attention_mask'])

    input_ids = torch.cat(input_ids,dim=0)
    attention_masks = torch.cat(attention_masks,dim=0)
    labels = torch.tensor(labels)
    sent_index = torch.tensor(sent_index)

    return sent_index,input_ids,attention_masks,labels

sent_index,input_ids,attention_masks,encoded_label_tensors =␣
 ↪encoder_generator(total_sentences,encoded_labels)
test_sent_index,test_input_ids,test_attention_masks,encoded_test_label_tensors␣
 ↪= encoder_generator(test_sentences,encoded_test_labels)
print('Original: ', total_sentences[0])
print('Token IDs:', input_ids[0])
```

```
#print(encoded_label_tensors)
#print(encoded_test_label_tensors)
```

```
Original:  Deutsche rothaarige porno reife deutsche frauen porno. Deutsche
politessen pornos porno deutsch inzets. https://t.co/xAag87Y0Jd
Token IDs: tensor([   101,  15389,  64354,  55200,  45854,  10183,  10343,
14243,  14601,
         17486,  10628,  25733,  10183,  10343,    119,  15389,  91929, 100319,
         10115,  10183,  14386,  10183,  10343,  24722,  10106,  17931,  10107,
           119,  14120,    131,    120,    120,    188,    119,  11170,    120,
           192,  10738,  14520,  11396,  11305,  14703,  10929,  15417,  10162,
           102,      0,      0,      0,      0,      0,      0,      0,      0,
             0,      0,      0,      0,      0,      0,      0,      0,      0,
             0,      0,      0,      0,      0,      0,      0,      0,      0,
             0,      0,      0,      0,      0,      0,      0,      0,      0,
             0,      0,      0,      0,      0,      0,      0,      0,      0,
             0,      0,      0,      0,      0,      0,      0,      0,      0,
             0,      0,      0,      0,      0,      0,      0,      0,      0,
             0,      0,      0,      0,      0,      0,      0,      0,      0,
             0,      0,      0,      0,      0,      0,      0,      0,      0,
             0,      0,      0,      0,      0,      0,      0,      0,      0,
             0,      0,      0,      0,      0,      0,      0,      0,      0,
             0,      0,      0,      0,      0,      0,      0,      0,      0,
             0,      0,      0,      0,      0,      0,      0,      0,      0,
             0,      0,      0,      0,      0,      0,      0,      0,      0,
             0,      0,      0,      0])
```

[5]:
```
from torch.utils.data import TensorDataset,random_split

dataset = TensorDataset(input_ids,attention_masks,encoded_label_tensors)
test_dataset =␣
 ↪TensorDataset(test_sent_index,test_input_ids,test_attention_masks,encoded_test_label_tensor

train_size = int(0.75*len(dataset))

val_size = len(dataset)-train_size

train_dataset,val_dataset = random_split(dataset,[train_size,val_size])

print('train data samples is {}'.format(len(train_dataset)))
print("valid data samples is {}".format(len(val_dataset)))
print("test data samples is {}".format(len(test_dataset)))
```

```
train data samples is 5281
valid data samples is 1761
test data samples is 526
```

[6]:
```
from torch.utils.data import DataLoader,RandomSampler,SequentialSampler

bs=8

train_data_loader = DataLoader(train_dataset,
                              sampler=RandomSampler(train_dataset),
                              batch_size=bs)
valid_data_loader = DataLoader(val_dataset,
                              sampler=SequentialSampler(val_dataset),
                              batch_size=bs)
test_data_loader = DataLoader(test_dataset,
                            sampler=SequentialSampler(test_dataset),
                            batch_size=bs)
```

[7]:
```
from transformers import BertForSequenceClassification, AdamW

model = BertForSequenceClassification.
 ↪from_pretrained('bert-base-multilingual-cased',
                                                     num_labels=len(le.
 ↪classes_),
                                                     output_attentions=False,
                                                     output_hidden_states=False,
                                                     )
model.cuda()
```

```
Some weights of the model checkpoint at bert-base-multilingual-cased were not
used when initializing BertForSequenceClassification: ['cls.predictions.bias',
'cls.predictions.transform.dense.weight',
'cls.predictions.transform.dense.bias', 'cls.predictions.decoder.weight',
'cls.seq_relationship.weight', 'cls.seq_relationship.bias',
'cls.predictions.transform.LayerNorm.weight',
'cls.predictions.transform.LayerNorm.bias']
- This IS expected if you are initializing BertForSequenceClassification from
the checkpoint of a model trained on another task or with another architecture
(e.g. initializing a BertForSequenceClassification model from a
BertForPretraining model).
- This IS NOT expected if you are initializing BertForSequenceClassification
from the checkpoint of a model that you expect to be exactly identical
(initializing a BertForSequenceClassification model from a
BertForSequenceClassification model).
Some weights of BertForSequenceClassification were not initialized from the
model checkpoint at bert-base-multilingual-cased and are newly initialized:
['classifier.weight', 'classifier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
```

```
[7]: BertForSequenceClassification(
       (bert): BertModel(
         (embeddings): BertEmbeddings(
           (word_embeddings): Embedding(119547, 768, padding_idx=0)
           (position_embeddings): Embedding(512, 768)
           (token_type_embeddings): Embedding(2, 768)
           (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
           (dropout): Dropout(p=0.1, inplace=False)
         )
         (encoder): BertEncoder(
           (layer): ModuleList(
             (0): BertLayer(
               (attention): BertAttention(
                 (self): BertSelfAttention(
                   (query): Linear(in_features=768, out_features=768, bias=True)
                   (key): Linear(in_features=768, out_features=768, bias=True)
                   (value): Linear(in_features=768, out_features=768, bias=True)
                   (dropout): Dropout(p=0.1, inplace=False)
                 )
                 (output): BertSelfOutput(
                   (dense): Linear(in_features=768, out_features=768, bias=True)
                   (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
                   (dropout): Dropout(p=0.1, inplace=False)
                 )
               )
               (intermediate): BertIntermediate(
                 (dense): Linear(in_features=768, out_features=3072, bias=True)
               )
               (output): BertOutput(
                 (dense): Linear(in_features=3072, out_features=768, bias=True)
                 (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
                 (dropout): Dropout(p=0.1, inplace=False)
               )
             )
             (1): BertLayer(
               (attention): BertAttention(
                 (self): BertSelfAttention(
                   (query): Linear(in_features=768, out_features=768, bias=True)
                   (key): Linear(in_features=768, out_features=768, bias=True)
                   (value): Linear(in_features=768, out_features=768, bias=True)
                   (dropout): Dropout(p=0.1, inplace=False)
                 )
                 (output): BertSelfOutput(
                   (dense): Linear(in_features=768, out_features=768, bias=True)
                   (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
                   (dropout): Dropout(p=0.1, inplace=False)
                 )
```

```
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(2): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(3): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
```

```
    (intermediate): BertIntermediate(
      (dense): Linear(in_features=768, out_features=3072, bias=True)
    )
    (output): BertOutput(
      (dense): Linear(in_features=3072, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (4): BertLayer(
    (attention): BertAttention(
      (self): BertSelfAttention(
        (query): Linear(in_features=768, out_features=768, bias=True)
        (key): Linear(in_features=768, out_features=768, bias=True)
        (value): Linear(in_features=768, out_features=768, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (output): BertSelfOutput(
        (dense): Linear(in_features=768, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (intermediate): BertIntermediate(
      (dense): Linear(in_features=768, out_features=3072, bias=True)
    )
    (output): BertOutput(
      (dense): Linear(in_features=3072, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (5): BertLayer(
    (attention): BertAttention(
      (self): BertSelfAttention(
        (query): Linear(in_features=768, out_features=768, bias=True)
        (key): Linear(in_features=768, out_features=768, bias=True)
        (value): Linear(in_features=768, out_features=768, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (output): BertSelfOutput(
        (dense): Linear(in_features=768, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (intermediate): BertIntermediate(
      (dense): Linear(in_features=768, out_features=3072, bias=True)
    )
    (output): BertOutput(
      (dense): Linear(in_features=3072, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (6): BertLayer(
    (attention): BertAttention(
      (self): BertSelfAttention(
        (query): Linear(in_features=768, out_features=768, bias=True)
        (key): Linear(in_features=768, out_features=768, bias=True)
        (value): Linear(in_features=768, out_features=768, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (output): BertSelfOutput(
        (dense): Linear(in_features=768, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (intermediate): BertIntermediate(
      (dense): Linear(in_features=768, out_features=3072, bias=True)
    )
    (output): BertOutput(
      (dense): Linear(in_features=3072, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (7): BertLayer(
    (attention): BertAttention(
      (self): BertSelfAttention(
        (query): Linear(in_features=768, out_features=768, bias=True)
        (key): Linear(in_features=768, out_features=768, bias=True)
        (value): Linear(in_features=768, out_features=768, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (output): BertSelfOutput(
        (dense): Linear(in_features=768, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (intermediate): BertIntermediate(
      (dense): Linear(in_features=768, out_features=3072, bias=True)
    )
    (output): BertOutput(
      (dense): Linear(in_features=3072, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (8): BertLayer(
    (attention): BertAttention(
      (self): BertSelfAttention(
        (query): Linear(in_features=768, out_features=768, bias=True)
        (key): Linear(in_features=768, out_features=768, bias=True)
        (value): Linear(in_features=768, out_features=768, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (output): BertSelfOutput(
        (dense): Linear(in_features=768, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (intermediate): BertIntermediate(
      (dense): Linear(in_features=768, out_features=3072, bias=True)
    )
    (output): BertOutput(
      (dense): Linear(in_features=3072, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (9): BertLayer(
    (attention): BertAttention(
      (self): BertSelfAttention(
        (query): Linear(in_features=768, out_features=768, bias=True)
        (key): Linear(in_features=768, out_features=768, bias=True)
        (value): Linear(in_features=768, out_features=768, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (output): BertSelfOutput(
        (dense): Linear(in_features=768, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (intermediate): BertIntermediate(
      (dense): Linear(in_features=768, out_features=3072, bias=True)
    )
```

```
    (output): BertOutput(
      (dense): Linear(in_features=3072, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (10): BertLayer(
    (attention): BertAttention(
      (self): BertSelfAttention(
        (query): Linear(in_features=768, out_features=768, bias=True)
        (key): Linear(in_features=768, out_features=768, bias=True)
        (value): Linear(in_features=768, out_features=768, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (output): BertSelfOutput(
        (dense): Linear(in_features=768, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (intermediate): BertIntermediate(
      (dense): Linear(in_features=768, out_features=3072, bias=True)
    )
    (output): BertOutput(
      (dense): Linear(in_features=3072, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (11): BertLayer(
    (attention): BertAttention(
      (self): BertSelfAttention(
        (query): Linear(in_features=768, out_features=768, bias=True)
        (key): Linear(in_features=768, out_features=768, bias=True)
        (value): Linear(in_features=768, out_features=768, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (output): BertSelfOutput(
        (dense): Linear(in_features=768, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (intermediate): BertIntermediate(
      (dense): Linear(in_features=768, out_features=3072, bias=True)
    )
    (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=768, out_features=2, bias=True)
)
```

[8]:
```
optimizer = AdamW(model.parameters(),lr=2e-5,eps=1e-8)

from transformers import get_linear_schedule_with_warmup

epochs=10
total_steps = len(train_data_loader) * epochs

scheduler = get_linear_schedule_with_warmup(optimizer,
                                            num_warmup_steps=0,
                                            num_training_steps=total_steps)
```

[9]:
```
import numpy as np

def predictions_labels(preds,labels):
    pred = np.argmax(preds,axis=1).flatten()
    label = labels.flatten()
    return pred,label
```

[10]:
```
import random
import numpy as np
import time
from sklearn.metrics import classification_report,accuracy_score,f1_score

total_t0 = time.time()

seed_val = 42

random.seed(seed_val)
np.random.seed(seed_val)
torch.manual_seed(seed_val)
torch.cuda.manual_seed_all(seed_val)
```

[11]:
```python
def categorical_accuracy(preds, y):
    """
    Returns accuracy per batch, i.e. if you get 8/10 right, this returns 0.8,␣
 ↪NOT 8
    """
    max_preds = preds.argmax(dim = 1, keepdim = True) # get the index of the␣
 ↪max probability
    correct = max_preds.squeeze(1).eq(y)
    return correct.sum() / torch.FloatTensor([y.shape[0]])

def predictions_labels(preds,labels):
    pred = np.argmax(preds,axis=1).flatten()
    label = labels.flatten()
    return pred,label
```

[12]:
```python
def train():
    total_train_loss = 0
    total_train_acc = 0

    model.train() # set model in train mode for batchnorm and dropout layers in␣
 ↪bert model

    for step,batch in enumerate(train_data_loader):
        b_input_ids = batch[0].to(device)
        b_input_mask = batch[1].to(device)
        b_labels = batch[2].to(device)
        model.zero_grad()

        loss,logits = model(b_input_ids,
                            attention_mask=b_input_mask,
                            labels=b_labels.long())

        total_train_loss+=loss.item()
        total_train_acc+=categorical_accuracy(logits,b_labels).item()

        loss.backward()

        torch.nn.utils.clip_grad_norm_(model.parameters(),1.0)

        optimizer.step()

        scheduler.step() #go ahead and update the learning rate

    avg_train_loss = total_train_loss/len(train_data_loader)
    avg_train_acc = total_train_acc/len(train_data_loader)

    return avg_train_loss,avg_train_acc
```

[13]:
```
def evaluate():
    model.eval()

    total_eval_accuracy = 0
    total_eval_loss = 0
    number_of_eval_steps= 0

    all_true_labels = []
    all_pred_labels = []

    for batch in valid_data_loader:
        b_input_ids = batch[0].to(device)
        b_input_mask = batch[1].to(device)
        b_labels = batch[2].to(device)

        with torch.no_grad():

            loss, logits = model(b_input_ids,
                                 attention_mask= b_input_mask,
                                 labels = b_labels.long())
        total_eval_loss+=loss.item()

        logits = logits.detach().cpu().numpy()

        label_ids = b_labels.to('cpu').numpy()

        pred,true = predictions_labels(logits,label_ids)

        all_pred_labels.extend(pred)
        all_true_labels.extend(true)

    print(classification_report(all_pred_labels,all_true_labels))
    avg_val_accuracy = accuracy_score(all_pred_labels,all_true_labels)
    macro_f1_score = f1_score(all_pred_labels,all_true_labels,average='macro')

    avg_val_loss = total_eval_loss/len(valid_data_loader)

    print("accuracy = {0:.2f}".format(avg_val_accuracy))

    return avg_val_loss,avg_val_accuracy,macro_f1_score
```

[14]:
```
import time
def epoch_time(start_time, end_time):
    elapsed_time = end_time - start_time
    elapsed_mins = int(elapsed_time / 60)
    elapsed_secs = int(elapsed_time - (elapsed_mins * 60))
    return elapsed_mins, elapsed_secs
```

```
[15]: epochs = 10

      best_macro_f1 = float('0')

      for epoch in range(epochs):

          start_time = time.time()
          train_loss,train_acc = train()
          valid_loss,valid_acc,macro_f1 = evaluate()

          end_time = time.time()

          epoch_mins, epoch_secs = epoch_time(start_time, end_time)

          if macro_f1 > best_macro_f1:
              best_macro_f1 = macro_f1
              torch.save(model,'model_german_task_a.pt')

          print(f'Epoch: {epoch+1:02} | Epoch Time: {epoch_mins}m {epoch_secs}s')
          print(f'\tTrain Loss: {train_loss:.3f} | Train Acc: {train_acc*100:.2f}%')
          print(f'\t Val. Loss: {valid_loss:.3f} |  Val. Acc: {valid_acc*100:.2f}%')
```

```
              precision    recall  f1-score   support

           0       0.43      0.70      0.53       194
           1       0.96      0.89      0.92      1567

   micro avg       0.86      0.86      0.86      1761
   macro avg       0.69      0.79      0.73      1761
weighted avg       0.90      0.86      0.88      1761

accuracy = 0.86
Epoch: 01 | Epoch Time: 3m 5s
        Train Loss: 0.408 | Train Acc: 85.29%
         Val. Loss: 0.364 |  Val. Acc: 86.43%
              precision    recall  f1-score   support

           0       0.60      0.55      0.57       339
           1       0.90      0.91      0.90      1422

   micro avg       0.84      0.84      0.84      1761
   macro avg       0.75      0.73      0.74      1761
weighted avg       0.84      0.84      0.84      1761

accuracy = 0.84
Epoch: 02 | Epoch Time: 3m 6s
        Train Loss: 0.346 | Train Acc: 88.09%
```

```
         Val. Loss: 0.376 |  Val. Acc: 84.21%
              precision    recall  f1-score   support

           0       0.52      0.61      0.56       269
           1       0.93      0.90      0.91      1492

   micro avg       0.85      0.85      0.85      1761
   macro avg       0.72      0.75      0.74      1761
weighted avg       0.87      0.85      0.86      1761

accuracy = 0.85
Epoch: 03 | Epoch Time: 3m 5s
        Train Loss: 0.280 | Train Acc: 91.75%
         Val. Loss: 0.463 |  Val. Acc: 85.46%
              precision    recall  f1-score   support

           0       0.52      0.67      0.59       245
           1       0.94      0.90      0.92      1516

   micro avg       0.87      0.87      0.87      1761
   macro avg       0.73      0.78      0.75      1761
weighted avg       0.89      0.87      0.87      1761

accuracy = 0.87
Epoch: 04 | Epoch Time: 3m 4s
        Train Loss: 0.204 | Train Acc: 94.78%
         Val. Loss: 0.663 |  Val. Acc: 86.83%
              precision    recall  f1-score   support

           0       0.50      0.62      0.55       258
           1       0.93      0.90      0.91      1503

   micro avg       0.86      0.86      0.86      1761
   macro avg       0.72      0.76      0.73      1761
weighted avg       0.87      0.86      0.86      1761

accuracy = 0.86
Epoch: 05 | Epoch Time: 3m 3s
        Train Loss: 0.116 | Train Acc: 97.35%
         Val. Loss: 0.741 |  Val. Acc: 85.52%
              precision    recall  f1-score   support

           0       0.57      0.58      0.57       309
           1       0.91      0.91      0.91      1452

   micro avg       0.85      0.85      0.85      1761
   macro avg       0.74      0.74      0.74      1761
weighted avg       0.85      0.85      0.85      1761
```

```
accuracy = 0.85
Epoch: 06 | Epoch Time: 3m 3s
        Train Loss: 0.078 | Train Acc: 98.37%
         Val. Loss: 0.966 |  Val. Acc: 84.89%
              precision    recall  f1-score   support

           0       0.52      0.64      0.57       257
           1       0.94      0.90      0.92      1504

   micro avg       0.86      0.86      0.86      1761
   macro avg       0.73      0.77      0.75      1761
weighted avg       0.88      0.86      0.87      1761

accuracy = 0.86
Epoch: 07 | Epoch Time: 3m 2s
        Train Loss: 0.044 | Train Acc: 99.17%
         Val. Loss: 0.960 |  Val. Acc: 86.14%
              precision    recall  f1-score   support

           0       0.52      0.64      0.57       257
           1       0.94      0.90      0.92      1504

   micro avg       0.86      0.86      0.86      1761
   macro avg       0.73      0.77      0.75      1761
weighted avg       0.88      0.86      0.87      1761

accuracy = 0.86
Epoch: 08 | Epoch Time: 3m 1s
        Train Loss: 0.028 | Train Acc: 99.49%
         Val. Loss: 1.064 |  Val. Acc: 86.14%
              precision    recall  f1-score   support

           0       0.52      0.64      0.57       256
           1       0.94      0.90      0.92      1505

   micro avg       0.86      0.86      0.86      1761
   macro avg       0.73      0.77      0.75      1761
weighted avg       0.88      0.86      0.87      1761

accuracy = 0.86
Epoch: 09 | Epoch Time: 3m 1s
        Train Loss: 0.012 | Train Acc: 99.72%
         Val. Loss: 1.152 |  Val. Acc: 86.20%
              precision    recall  f1-score   support

           0       0.50      0.67      0.57       233
           1       0.95      0.90      0.92      1528
   micro avg       0.87      0.87      0.87      1761
   macro avg       0.72      0.78      0.74      1761
weighted avg       0.89      0.87      0.87      1761

accuracy = 0.87
Epoch: 10 | Epoch Time: 3m 1s
        Train Loss: 0.006 | Train Acc: 99.89%
         Val. Loss: 1.159 |  Val. Acc: 86.60%
```

[16]:
```
del model
import gc
gc.collect()

model = torch.load('model_german_task_a.pt')
model = model.to(device)
```

[17]:
```
def evaluate_test():
    model.eval()

    total_eval_accuracy = 0
    total_eval_loss = 0
    number_of_eval_steps= 0

    all_true_labels = []
    all_pred_labels = []

    all_sentence_id=[]

    for batch in test_data_loader:
        b_sentence_id = batch[0].to(device)
        b_input_ids = batch[1].to(device)
        b_input_mask = batch[2].to(device)
        b_labels = batch[3].to(device)

        sent_ids = b_sentence_id.to('cpu').numpy()
        all_sentence_id.extend(sent_ids)

        with torch.no_grad():

            loss, logits = model(b_input_ids,
                                 attention_mask= b_input_mask,
                                 labels = b_labels.long())
        total_eval_loss+=loss.item()

        logits = logits.detach().cpu().numpy()
        label_ids = b_labels.to('cpu').numpy()


        pred,true = predictions_labels(logits,label_ids)

        all_pred_labels.extend(pred)

        all_true_labels.extend(true)

    print(classification_report(all_pred_labels,all_true_labels))
    avg_val_accuracy = accuracy_score(all_pred_labels,all_true_labels)

    avg_val_loss = total_eval_loss/len(valid_data_loader)

    print("accuracy = {0:.2f}".format(avg_val_accuracy))

    return avg_val_loss,avg_val_accuracy,all_sentence_id,all_pred_labels

valid_loss,valid_acc,all_sentence_id,all_pred_labels = evaluate_test()
```

```
              precision    recall  f1-score   support

           0       0.72      0.64      0.68       152
           1       0.86      0.90      0.88       374

   micro avg       0.83      0.83      0.83       526
   macro avg       0.79      0.77      0.78       526
weighted avg       0.82      0.83      0.82       526

accuracy = 0.83
```

```
[ ]:
```

German_Task_B

June 13, 2022

[1]:
```
import torch

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

[2]:
```
import pandas as pd
import math
from sklearn import preprocessing

file = "hasoc_2020_de_train_new_b.xlsx"
file_test = "german_test_1509.csv"


df_train = pd.read_excel(file,index_col=0)

df_train = df_train.dropna()

df_test = pd.read_csv(file_test)

task = 'task2'
task_2019 = 'task_2'

#2019 datasets also

file_2019_1 = pd.read_csv("2019/german_dataset/german_dataset/
 ↪hasoc_de_test_gold.tsv",sep='\t')
file_2019_2 = pd.read_csv("2019/german_dataset/german_dataset/german_dataset.
 ↪tsv",sep="\t")


sentences_2019_1 = list(file_2019_1['text'].values)
sentences_2019_2 = list(file_2019_2['text'].values)

labels_2019_1 = list(file_2019_1[task_2019].values)
labels_2019_2 = list(file_2019_2[task_2019].values)
```

```python
print(len(df_train))
print(df_train.head())

total_sentences = list(df_train['text'].values)
total_labels = list(df_train[task].values)

total_sentences.extend(sentences_2019_1)
total_sentences.extend(sentences_2019_2)

total_labels.extend(labels_2019_1)
total_labels.extend(labels_2019_2)

test_sentences = list(df_test['text'].values)
test_labels = list(df_test[task].values)

def clean_text(sentences):
    for index,line in enumerate(sentences):
        if "\n" in line:
            sentences[index] = line.replace("\n","")
    return sentences

total_sentences = clean_text(total_sentences)
test_sentences = clean_text(test_sentences)

def clean_labels(labels):
    new_list= []
    for value in labels:
        new_list.append(value.strip())
    return new_list

total_labels = clean_labels(total_labels)
test_labels = clean_labels(test_labels)

le = preprocessing.LabelEncoder()
le.fit(total_labels)
encoded_labels = le.transform(total_labels)
encoded_test_labels = le.transform(test_labels)
print(set(encoded_labels))

print(len(total_sentences),len(encoded_labels),len(test_sentences),len(encoded_test_labels))

print(df_test)
```

```
2373
                                                                    text task1  \
tweet_id
```

```
1133388798925189122  Deutsche rothaarige porno reife deutsche fraue…   NOT
1127134592517980161  RT @NDRinfo: Die deutsche Klimaaktivistin Luis…   NOT
1128897106171842560  @ruhrbahn jeden Morgen eine neue „Fahrzeugstör…   NOT
1123576753199484928  @Junge_Freiheit Die Inkas hatten sich schon dä…   NOT
1128743783393312768  RT @technosteron: leute die 'boar' schreiben l…   HOF

                    task2                 ID
tweet_id
1133388798925189122  NONE  hasoc_2020_de_2684
1127134592517980161  NONE  hasoc_2020_de_1042
1128897106171842560  NONE   hasoc_2020_de_774
1123576753199484928  NONE   hasoc_2020_de_559
1128743783393312768  PRFN  hasoc_2020_de_1969
{0, 1, 2, 3}
7042 7042 526 526
                tweet_id                                              text  \
0    1129095874242650112    @derCarsti Boykottieren hört sich besser an.
1    1129004308396236800  RT @ibikus31: Es wird spekuliert, ob Merkel ei…
2    1130896929355907080  Hat #Hitler wirklich den Krieg in der Wüste ve…
3    1132251534329307136  RT @Beatrix_vStorch: #May tritt in UK unter Tr…
4    1124941869115498496  @justmeDoro Eher nicht. Das Gänse hauen wieder…
..                   …                                              …
521  1124809878546128897  RT @ChanMachtSo: SCHMERZEN!!!! Au!!! Mein Gehi…
522  1132433240000798720  Die Zerstörung der Grünen. https://t.co/SIYDJj…
523  1127366294255357958  RT @PParzival: "Antideutsche" Pseudo-linke Ide…
524  1124362090460975105                        Klug reden und dumm leben.
525  1131487097293103104  Wissen wir schon lange…hat das die Merkel e…

    task1 task2                  ID
0     NOT  NONE  hasoc_2020_de_1053
1     NOT  NONE   hasoc_2020_de_671
2     NOT  NONE  hasoc_2020_de_2977
3     NOT  NONE  hasoc_2020_de_1746
4     NOT  NONE  hasoc_2020_de_2416
..    …    …                  …
521   NOT  NONE   hasoc_2020_de_486
522   NOT  NONE  hasoc_2020_de_3388
523   NOT  NONE  hasoc_2020_de_2745
524   NOT  NONE   hasoc_2020_de_236
525   NOT  NONE  hasoc_2020_de_2850

[526 rows x 5 columns]
```

[3]:
```
from transformers import BertTokenizer


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

```
max_length = 0
for sentence in total_sentences:
    #print(sentence)
    length = len(tokenizer.tokenize(sentence))
    if length > max_length:
        max_length  = length
print("max token length is: ",max_length)
# max token length obtained is 50
# bert tokens are limited to 514 bytes.
```

```
max token length is:  175
```

[4]:
```
def encoder_generator(sentences,labels):

    sent_index = []
    input_ids = []
    attention_masks =[]

    for index,sent in enumerate(sentences):

        sent_index.append(index)

        encoded_dict = tokenizer.encode_plus(sent,
                                             add_special_tokens=True,
                                             max_length=max_length,
                                             pad_to_max_length=True,
                                             truncation = True,
                                             return_attention_mask=True,
                                             return_tensors='pt')
        input_ids.append(encoded_dict['input_ids'])

        attention_masks.append(encoded_dict['attention_mask'])

    input_ids = torch.cat(input_ids,dim=0)
    attention_masks = torch.cat(attention_masks,dim=0)
    labels = torch.tensor(labels)
    sent_index = torch.tensor(sent_index)

    return sent_index,input_ids,attention_masks,labels

sent_index,input_ids,attention_masks,encoded_label_tensors =␣
 ↪encoder_generator(total_sentences,encoded_labels)
test_sent_index,test_input_ids,test_attention_masks,encoded_test_label_tensors␣
 ↪= encoder_generator(test_sentences,encoded_test_labels)
print('Original: ', total_sentences[0])
print('Token IDs:', input_ids[0])
```

```
#print(encoded_label_tensors)
#print(encoded_test_label_tensors)
```

```
Original:  Deutsche rothaarige porno reife deutsche frauen porno. Deutsche
politessen pornos porno deutsch inzets. https://t.co/xAag87Y0Jd
Token IDs: tensor([   101,  15389,  64354,  55200,  45854,  10183,  10343,
14243,  14601,
         17486,  10628,  25733,  10183,  10343,    119,  15389,  91929, 100319,
         10115,  10183,  14386,  10183,  10343,  24722,  10106,  17931,  10107,
           119,  14120,    131,    120,    120,    188,    119,  11170,    120,
           192,  10738,  14520,  11396,  11305,  14703,  10929,  15417,  10162,
           102,      0,      0,      0,      0,      0,      0,      0,      0,
             0,      0,      0,      0,      0,      0,      0,      0,      0,
             0,      0,      0,      0,      0,      0,      0,      0,      0,
             0,      0,      0,      0,      0,      0,      0,      0,      0,
             0,      0,      0,      0,      0,      0,      0,      0,      0,
             0,      0,      0,      0,      0,      0,      0,      0,      0,
             0,      0,      0,      0,      0,      0,      0,      0,      0,
             0,      0,      0,      0,      0,      0,      0,      0,      0,
             0,      0,      0,      0,      0,      0,      0,      0,      0,
             0,      0,      0,      0,      0,      0,      0,      0,      0,
             0,      0,      0,      0,      0,      0,      0,      0,      0,
             0,      0,      0,      0,      0,      0,      0,      0,      0,
             0,      0,      0,      0,      0,      0,      0,      0,      0,
             0,      0,      0,      0,      0,      0,      0,      0,      0,
             0,      0,      0,      0])
```

[5]:
```
from torch.utils.data import TensorDataset,random_split

dataset = TensorDataset(input_ids,attention_masks,encoded_label_tensors)
test_dataset =␣
 ↪TensorDataset(test_sent_index,test_input_ids,test_attention_masks,encoded_test_label_tensor

train_size = int(0.75*len(dataset))

val_size = len(dataset)-train_size

train_dataset,val_dataset = random_split(dataset,[train_size,val_size])

print('train data samples is {}'.format(len(train_dataset)))
print("valid data samples is {}".format(len(val_dataset)))
print("test data samples is {}".format(len(test_dataset)))
```

```
train data samples is 5281
valid data samples is 1761
test data samples is 526
```

[6]:
```
from torch.utils.data import DataLoader,RandomSampler,SequentialSampler

bs=8

train_data_loader = DataLoader(train_dataset,
                              sampler=RandomSampler(train_dataset),
                              batch_size=bs)
valid_data_loader = DataLoader(val_dataset,
                              sampler=SequentialSampler(val_dataset),
                              batch_size=bs)
test_data_loader = DataLoader(test_dataset,
                            sampler=SequentialSampler(test_dataset),
                            batch_size=bs)
```

[7]:
```
from transformers import BertForSequenceClassification, AdamW

model = BertForSequenceClassification.
 ↪from_pretrained('bert-base-multilingual-cased',
                                                      num_labels=len(le.
 ↪classes_),
                                                      output_attentions=False,
                                                      output_hidden_states=False,
                                                      )
model.cuda()
```

```
Some weights of the model checkpoint at bert-base-multilingual-cased were not
used when initializing BertForSequenceClassification: ['cls.predictions.bias',
'cls.predictions.transform.dense.weight',
'cls.predictions.transform.dense.bias', 'cls.predictions.decoder.weight',
'cls.seq_relationship.weight', 'cls.seq_relationship.bias',
'cls.predictions.transform.LayerNorm.weight',
'cls.predictions.transform.LayerNorm.bias']
- This IS expected if you are initializing BertForSequenceClassification from
the checkpoint of a model trained on another task or with another architecture
(e.g. initializing a BertForSequenceClassification model from a
BertForPretraining model).
- This IS NOT expected if you are initializing BertForSequenceClassification
from the checkpoint of a model that you expect to be exactly identical
(initializing a BertForSequenceClassification model from a
BertForSequenceClassification model).
Some weights of BertForSequenceClassification were not initialized from the
model checkpoint at bert-base-multilingual-cased and are newly initialized:
['classifier.weight', 'classifier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
```

```
[7]: BertForSequenceClassification(
       (bert): BertModel(
         (embeddings): BertEmbeddings(
           (word_embeddings): Embedding(119547, 768, padding_idx=0)
           (position_embeddings): Embedding(512, 768)
           (token_type_embeddings): Embedding(2, 768)
           (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
           (dropout): Dropout(p=0.1, inplace=False)
         )
         (encoder): BertEncoder(
           (layer): ModuleList(
             (0): BertLayer(
               (attention): BertAttention(
                 (self): BertSelfAttention(
                   (query): Linear(in_features=768, out_features=768, bias=True)
                   (key): Linear(in_features=768, out_features=768, bias=True)
                   (value): Linear(in_features=768, out_features=768, bias=True)
                   (dropout): Dropout(p=0.1, inplace=False)
                 )
                 (output): BertSelfOutput(
                   (dense): Linear(in_features=768, out_features=768, bias=True)
                   (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
                   (dropout): Dropout(p=0.1, inplace=False)
                 )
               )
               (intermediate): BertIntermediate(
                 (dense): Linear(in_features=768, out_features=3072, bias=True)
               )
               (output): BertOutput(
                 (dense): Linear(in_features=3072, out_features=768, bias=True)
                 (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
                 (dropout): Dropout(p=0.1, inplace=False)
               )
             )
             (1): BertLayer(
               (attention): BertAttention(
                 (self): BertSelfAttention(
                   (query): Linear(in_features=768, out_features=768, bias=True)
                   (key): Linear(in_features=768, out_features=768, bias=True)
                   (value): Linear(in_features=768, out_features=768, bias=True)
                   (dropout): Dropout(p=0.1, inplace=False)
                 )
                 (output): BertSelfOutput(
                   (dense): Linear(in_features=768, out_features=768, bias=True)
                   (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
                   (dropout): Dropout(p=0.1, inplace=False)
                 )
```

```
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(2): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(3): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(4): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(5): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(6): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(7): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
```

```
      )
      (output): BertOutput(
        (dense): Linear(in_features=3072, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (8): BertLayer(
      (attention): BertAttention(
        (self): BertSelfAttention(
          (query): Linear(in_features=768, out_features=768, bias=True)
          (key): Linear(in_features=768, out_features=768, bias=True)
          (value): Linear(in_features=768, out_features=768, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (output): BertSelfOutput(
          (dense): Linear(in_features=768, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
      (intermediate): BertIntermediate(
        (dense): Linear(in_features=768, out_features=3072, bias=True)
      )
      (output): BertOutput(
        (dense): Linear(in_features=3072, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (9): BertLayer(
      (attention): BertAttention(
        (self): BertSelfAttention(
          (query): Linear(in_features=768, out_features=768, bias=True)
          (key): Linear(in_features=768, out_features=768, bias=True)
          (value): Linear(in_features=768, out_features=768, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (output): BertSelfOutput(
          (dense): Linear(in_features=768, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
      (intermediate): BertIntermediate(
        (dense): Linear(in_features=768, out_features=3072, bias=True)
      )
```

```
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(10): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(11): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=768, out_features=4, bias=True)
)
```

[8]:
```
optimizer = AdamW(model.parameters(),lr=2e-5,eps=1e-8)

from transformers import get_linear_schedule_with_warmup

epochs=10
total_steps = len(train_data_loader) * epochs

scheduler = get_linear_schedule_with_warmup(optimizer,
                                            num_warmup_steps=0,
                                            num_training_steps=total_steps)
```

[9]:
```
import numpy as np

def predictions_labels(preds,labels):
    pred = np.argmax(preds,axis=1).flatten()
    label = labels.flatten()
    return pred,label
```

[10]:
```
import random
import numpy as np
import time
from sklearn.metrics import classification_report,accuracy_score,f1_score

total_t0 = time.time()

seed_val = 42

random.seed(seed_val)
np.random.seed(seed_val)
torch.manual_seed(seed_val)
torch.cuda.manual_seed_all(seed_val)
```

[11]:
```python
def categorical_accuracy(preds, y):
    """
    Returns accuracy per batch, i.e. if you get 8/10 right, this returns 0.8,␣
↪NOT 8
    """
    max_preds = preds.argmax(dim = 1, keepdim = True) # get the index of the␣
↪max probability
    correct = max_preds.squeeze(1).eq(y)
    return correct.sum() / torch.FloatTensor([y.shape[0]])

def predictions_labels(preds,labels):
    pred = np.argmax(preds,axis=1).flatten()
    label = labels.flatten()
    return pred,label
```

[12]:
```python
def train():
    total_train_loss = 0
    total_train_acc = 0

    model.train() # set model in train mode for batchnorm and dropout layers in␣
↪bert model

    for step,batch in enumerate(train_data_loader):
        b_input_ids = batch[0].to(device)
        b_input_mask = batch[1].to(device)
        b_labels = batch[2].to(device)
        model.zero_grad()

        loss,logits = model(b_input_ids,
                            attention_mask=b_input_mask,
                            labels=b_labels.long())

        total_train_loss+=loss.item()
        total_train_acc+=categorical_accuracy(logits,b_labels).item()

        loss.backward()

        torch.nn.utils.clip_grad_norm_(model.parameters(),1.0)

        optimizer.step()

        scheduler.step() #go ahead and update the learning rate

    avg_train_loss = total_train_loss/len(train_data_loader)
    avg_train_acc = total_train_acc/len(train_data_loader)

    return avg_train_loss,avg_train_acc
```

[13]:
```
def evaluate():
    model.eval()

    total_eval_accuracy = 0
    total_eval_loss = 0
    number_of_eval_steps= 0

    all_true_labels = []
    all_pred_labels = []

    for batch in valid_data_loader:
        b_input_ids = batch[0].to(device)
        b_input_mask = batch[1].to(device)
        b_labels = batch[2].to(device)

        with torch.no_grad():

            loss, logits = model(b_input_ids,
                                attention_mask= b_input_mask,
                                labels = b_labels.long())
        total_eval_loss+=loss.item()

        logits = logits.detach().cpu().numpy()

        label_ids = b_labels.to('cpu').numpy()

        pred,true = predictions_labels(logits,label_ids)

        all_pred_labels.extend(pred)
        all_true_labels.extend(true)

    print(classification_report(all_pred_labels,all_true_labels))
    avg_val_accuracy = accuracy_score(all_pred_labels,all_true_labels)
    macro_f1_score = f1_score(all_pred_labels,all_true_labels,average='macro')

    avg_val_loss = total_eval_loss/len(valid_data_loader)

    print("accuracy = {0:.2f}".format(avg_val_accuracy))

    return avg_val_loss,avg_val_accuracy,macro_f1_score
```

[14]:
```
import time
def epoch_time(start_time, end_time):
    elapsed_time = end_time - start_time
    elapsed_mins = int(elapsed_time / 60)
    elapsed_secs = int(elapsed_time - (elapsed_mins * 60))
    return elapsed_mins, elapsed_secs
```

[15]:
```
epochs = 10

best_macro_f1 = float('0')

for epoch in range(epochs):

    start_time = time.time()
    train_loss,train_acc = train()
    valid_loss,valid_acc,macro_f1 = evaluate()

    end_time = time.time()

    epoch_mins, epoch_secs = epoch_time(start_time, end_time)

    if macro_f1 > best_macro_f1:
        best_macro_f1 = macro_f1
        torch.save(model,'model_german_task_b.pt')

    print(f'Epoch: {epoch+1:02} | Epoch Time: {epoch_mins}m {epoch_secs}s')
    print(f'\tTrain Loss: {train_loss:.3f} | Train Acc: {train_acc*100:.2f}%')
    print(f'\t Val. Loss: {valid_loss:.3f} |  Val. Acc: {valid_acc*100:.2f}%')
```

```
C:\Users\suman\Miniconda3\envs\py3_env\lib\site-
packages\sklearn\metrics\classification.py:1145: UndefinedMetricWarning: Recall
and F-score are ill-defined and being set to 0.0 in labels with no true samples.
  'recall', 'true', average, warn_for)
C:\Users\suman\Miniconda3\envs\py3_env\lib\site-
packages\sklearn\metrics\classification.py:1145: UndefinedMetricWarning: F-score
is ill-defined and being set to 0.0 in labels with no true samples.
  'recall', 'true', average, warn_for)
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         0
           1       0.97      0.89      0.93      1602
           2       0.10      0.61      0.18        18
           3       0.71      0.54      0.61       141

   micro avg       0.86      0.86      0.86      1761
   macro avg       0.45      0.51      0.43      1761
weighted avg       0.94      0.86      0.89      1761

accuracy = 0.86
Epoch: 01 | Epoch Time: 3m 20s
        Train Loss: 0.606 | Train Acc: 83.23%
         Val. Loss: 0.538 |  Val. Acc: 85.69%
              precision    recall  f1-score   support
           0       0.01      0.50      0.03         2
           1       0.98      0.88      0.93      1637
           2       0.09      1.00      0.16         9
           3       0.64      0.60      0.62       113

   micro avg       0.86      0.86      0.86      1761
   macro avg       0.43      0.75      0.43      1761
weighted avg       0.95      0.86      0.90      1761

accuracy = 0.86
Epoch: 02 | Epoch Time: 3m 9s
        Train Loss: 0.493 | Train Acc: 86.14%
         Val. Loss: 0.518 |  Val. Acc: 86.26%
              precision    recall  f1-score   support

           0       0.05      0.36      0.09        11
           1       0.97      0.89      0.93      1613
           2       0.18      0.49      0.26        39
           3       0.63      0.68      0.65        98

   micro avg       0.86      0.86      0.86      1761
   macro avg       0.46      0.61      0.48      1761
weighted avg       0.93      0.86      0.89      1761

accuracy = 0.86
Epoch: 03 | Epoch Time: 3m 10s
        Train Loss: 0.399 | Train Acc: 88.62%
         Val. Loss: 0.615 |  Val. Acc: 86.48%
              precision    recall  f1-score   support

           0       0.15      0.33      0.21        36
           1       0.97      0.89      0.93      1603
           2       0.16      0.68      0.26        25
           3       0.64      0.70      0.67        97

   micro avg       0.87      0.87      0.87      1761
   macro avg       0.48      0.65      0.52      1761
weighted avg       0.92      0.87      0.89      1761

accuracy = 0.87
Epoch: 04 | Epoch Time: 3m 11s
        Train Loss: 0.301 | Train Acc: 91.70%
         Val. Loss: 0.669 |  Val. Acc: 86.60%
              precision    recall  f1-score   support

           0       0.10      0.24      0.14        34
           1       0.96      0.89      0.92      1576
           2       0.27      0.42      0.33        66
           3       0.53      0.67      0.59        85

   micro avg       0.85      0.85      0.85      1761
   macro avg       0.46      0.56      0.50      1761
weighted avg       0.89      0.85      0.87      1761

accuracy = 0.85
Epoch: 05 | Epoch Time: 3m 10s
        Train Loss: 0.201 | Train Acc: 95.12%
         Val. Loss: 0.832 |  Val. Acc: 85.18%
              precision    recall  f1-score   support

           0       0.21      0.24      0.22        68
           1       0.93      0.90      0.92      1526
           2       0.16      0.57      0.25        30
           3       0.69      0.54      0.61       137

   micro avg       0.84      0.84      0.84      1761
   macro avg       0.50      0.56      0.50      1761
weighted avg       0.87      0.84      0.85      1761

accuracy = 0.84
Epoch: 06 | Epoch Time: 3m 11s
        Train Loss: 0.116 | Train Acc: 97.37%
         Val. Loss: 0.952 |  Val. Acc: 84.04%
              precision    recall  f1-score   support

           0       0.09      0.35      0.14        20
           1       0.96      0.89      0.93      1591
           2       0.23      0.38      0.29        63
           3       0.57      0.70      0.63        87

   micro avg       0.86      0.86      0.86      1761
   macro avg       0.46      0.58      0.50      1761
weighted avg       0.91      0.86      0.88      1761

accuracy = 0.86
Epoch: 07 | Epoch Time: 3m 12s
        Train Loss: 0.075 | Train Acc: 98.41%
         Val. Loss: 1.043 |  Val. Acc: 85.69%
              precision    recall  f1-score   support

           0       0.15      0.26      0.19        47
           1       0.95      0.89      0.92      1567
           2       0.18      0.40      0.25        47
           3       0.63      0.67      0.65       100

   micro avg       0.85      0.85      0.85      1761
   macro avg       0.48      0.56      0.50      1761
weighted avg       0.89      0.85      0.87      1761

accuracy = 0.85
Epoch: 08 | Epoch Time: 3m 12s
        Train Loss: 0.037 | Train Acc: 99.17%
         Val. Loss: 1.133 |  Val. Acc: 85.12%
              precision    recall  f1-score   support

           0       0.21      0.25      0.22        65
           1       0.94      0.90      0.92      1549
           2       0.19      0.48      0.27        42
           3       0.64      0.66      0.65       105

   micro avg       0.85      0.85      0.85      1761
   macro avg       0.50      0.57      0.52      1761
weighted avg       0.88      0.85      0.86      1761

accuracy = 0.85
Epoch: 09 | Epoch Time: 3m 11s
        Train Loss: 0.015 | Train Acc: 99.68%
         Val. Loss: 1.196 |  Val. Acc: 84.84%
              precision    recall  f1-score   support

           0       0.17      0.27      0.20        49
           1       0.95      0.89      0.92      1570
           2       0.19      0.42      0.26        48
           3       0.63      0.71      0.67        94

   micro avg       0.85      0.85      0.85      1761
   macro avg       0.48      0.57      0.51      1761
weighted avg       0.89      0.85      0.87      1761

accuracy = 0.85
Epoch: 10 | Epoch Time: 3m 9s
        Train Loss: 0.009 | Train Acc: 99.75%
         Val. Loss: 1.218 |  Val. Acc: 85.41%
```

[15]:
```
del model
import gc
gc.collect()

model = torch.load('model_german_task_b.pt')
model = model.to(device)
```

[16]:
```
def evaluate_test():
    model.eval()
    total_eval_accuracy = 0
    total_eval_loss = 0
    number_of_eval_steps= 0

    all_true_labels = []
    all_pred_labels = []

    all_sentence_id=[]

    for batch in test_data_loader:
        b_sentence_id = batch[0].to(device)
        b_input_ids = batch[1].to(device)
        b_input_mask = batch[2].to(device)
        b_labels = batch[3].to(device)

        sent_ids = b_sentence_id.to('cpu').numpy()
        all_sentence_id.extend(sent_ids)

        with torch.no_grad():

            loss, logits = model(b_input_ids,
                                 attention_mask= b_input_mask,
                                 labels = b_labels.long())
        total_eval_loss+=loss.item()

        logits = logits.detach().cpu().numpy()

        label_ids = b_labels.to('cpu').numpy()


        pred,true = predictions_labels(logits,label_ids)

        all_pred_labels.extend(pred)

        all_true_labels.extend(true)

    print(classification_report(all_pred_labels,all_true_labels))
    avg_val_accuracy = accuracy_score(all_pred_labels,all_true_labels)

    avg_val_loss = total_eval_loss/len(valid_data_loader)

    print("accuracy = {0:.2f}".format(avg_val_accuracy))

    return avg_val_loss,avg_val_accuracy,all_sentence_id,all_pred_labels

valid_loss,valid_acc,all_sentence_id,all_pred_labels = evaluate_test()
```

```
              precision    recall  f1-score   support

           0       0.17      0.36      0.23        11
           1       0.93      0.86      0.89       407
           2       0.06      0.40      0.10         5
           3       0.75      0.64      0.69       103

   micro avg       0.80      0.80      0.80       526
   macro avg       0.48      0.57      0.48       526
weighted avg       0.87      0.80      0.83       526

accuracy = 0.80
```

```
[ ]:
```

Bibliography

[1] Flor Miriam Plaza-del-Arco,Sercan Halat,Sebastian Padó, Roman Klinger(2021),Multi-Task Learning with Sentiment, Emotion, and Target Detection to Recognize Hate Speech and Offensive Language.

[2]] European Commission against Racism and Intolerance (ECRI), ECRI General Policy Recommendation No. 15 on Combating Hate Speech, Online, 2015.

[3] F.-M. Plaza-Del-Arco, M. D. Molina-González, L. A. Ureña López, M. T. Martín-Valdivia, Detecting Misogyny and Xenophobia in Spanish Tweets Using Language Technologies, ACM Trans. Internet Technol. 20 (2020).

[4]] A. Schmidt, M. Wiegand, A Survey on Hate Speech Detection using Natural Language Processing, in: Proceedings of the Fifth International Workshop on Natural Language Processing for Social Media, Association for Computational Linguistics, Valencia, Spain, 2017, pp. 1–10.

[5] M. Sap, D. Card, S. Gabriel, Y. Choi, N. A. Smith, The Risk of Racial Bias in Hate Speech Detection, in: Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Florence, Italy, 2019, pp. 1668–1678

[6] A. Rodríguez, C. Argueta, Y.-L. Chen, Automatic Detection of Hate Speech on Facebook Using Sentiment and Emotion Analysis, in: 2019 International Conference on Artificial Intelligence in Information and Communication (ICAIIC), 2019, pp. 169–174.

[7] R. Plutchik, The nature of emotions: Human emotions have deep evolutionary roots, a fact that may explain their complexity and provide tools for clinical practice, American scientist 89 (2001) 344–350.

[8] B. Liu, Sentiment Analysis – Mining Opinions, Sentiments, and Emotions, 2nd edition ed., Cambridge University Press, 2012

[9] S. Mohammad, F. Bravo-Marquez, Emotion Intensities in Tweets, in: Proceedings of the 6th Joint Conference on Lexical and Computational Semantics (*SEM 2017), Association for Computational Linguistics, Vancouver, Canada, 2017, pp. 65–77

[10] L. A. M. Oberländer, R. Klinger, Token Sequence Labeling vs. Clause Classification for English Emotion Stimulus Detection, in: Proceedings of the Ninth Joint Conference on Lexical and Computational Semantics, Association for Computational Linguistics,Barcelona, Spain (Online), 2020, pp. 58–70

[11] B. M. Doan Dang, L. Oberländer, R. Klinger, Emotion Stimulus Detection in German News Headlines, in: Proceedings of the 17th Conference on Natural Language Processing (KONVENS 2021), German Society for Computational Linguistics Language Technology, Düsseldorf, Germany, 2021.

[12] P. Fortuna, S. Nunes, A Survey on Automatic Detection of Hate Speech in Text, ACM Comput. Surv. 51 (2018). doi:https://doi.org/10.1145/3232676.

[13] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, L. Zettlemoyer, Deep contextualized word representations, arXiv preprint arXiv:1802.05365 (2018).

[14] T. Kudo, J. Richardson, Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing, arXiv preprint arXiv:1808.06226 (2018).

[15] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, et al., Huggingface's transformers: State-of-the-art natural language processing, ArXiv (2019) arXiv–1910.

[16]] T. Mandl, S. Modha, P. Majumder, D. Patel, M. Dave, C. Mandlia, A. Patel, Overview of the hasoc track at fire 2019: Hate speech and offensive content identification in indo-european languages, in: Proceedings of the 11th Forum for Information Retrieval Evaluation, 2019,pp. 14–17.

[17] T. Mandl, S. Modha, G. K. Shahi, A. K. Jaiswal, D. Nandini, D. Patel, P. Majumder, J. Schäfer, Overview of the HASOC track at FIRE 2020: Hate Speech and Offensive Content Identification in Indo-European Languages), in: Working Notes of FIRE 2020 - Forum for Information Retrieval Evaluation, CEUR, 2020.

[18] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, in: Advances in neural information processing systems, 2017, pp.5998–6008.

[19] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding, arXiv preprint arXiv:1810.04805 (2018).

[20] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural computation 9 (1997) 1735–1780.

[21] Y. Kim, Convolutional neural networks for sentence classification, arXiv preprint arXiv:1408.5882 (2014).

[22] P. Badjatiya, S. Gupta, M. Gupta, V. Varma, Deep learning for hate speech detection in tweets, in: Proceedings of the 26th International Conference on World Wide Web Companion, 2017, pp.759–760.

[23] B. Gambäck, U. K. Sikdar, Using convolutional neural networks to classify hate-speech, in: Proceedings of the first workshop on abusive language online, 2017, pp. 85–90.

[24] A. Gaydhani, V. Doma, S. Kendre, L. Bhagwat, Detecting hate speech and offensive language on twitter using machine learning: An n-gram and tfidf based approach, arXiv preprint arXiv:1809.08651 (2018).

[25] T. Davidson, D. Warmsley, M. Macy, I. Weber, Automated hate speech detection and the problem of offensive language, arXiv preprint arXiv:1703.04009 (2017).

[26] Awantee Deshpande, Dana Ruiter, Marius Mosbach, Dietrich Klakow(2022), StereoKG: Data-Driven Knowledge Graph Construction for Cultural Knowledge and Stereotypes

[27] Raviraj Joshi(2022),L3Cube-MahaNLP: Marathi Natural Language Processing Datasets, Models, and Library.

[28] Andrei Paraschiv, Mihai Dascalu, Dumitru-Clementin Cerce(2021),UPB at SemEval-2022 Task 5: Enhancing UNITER with Image Sentiment and Graph Convolutional Networks for Multimedia Automatic Misogyny Identification.

[29] Khanh Q. Tran, An T. Nguyen, Phu Gia Hoang, Canh Duc Luu, Trong-Hop Do, Kiet Van Nguyen(2022), Hate and Offensive Detection using PhoBERT-CNN and Social Media Streaming Data.

[30] Xin Lian(2022), Speech Detection Task Against Asian Hate: BERT the Central, While Data-Centric Studies the Crucial.

[31] Georgios K. Pitsilis(2022), Improved two-stage hate speech classification for twitter based on Deep Neural Networks.

[32] Suman Dowlagar,Radhika Mamid (2020) , Using BERT and Multilingual BERT models for Hate Speech Detection.

[33] Aurelion Geron (2015), Hands on machine learning with SciKIT learn,keras and tensorflow.

[34] Y. Li, H. Su, X. Shen, W. Li, Z. Cao, S. Niu, DailyDialog: A Manually Labelled Multi-turn Dialogue Dataset, in: Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers), Asian Federation of Natural Language Processing, Taipei, Taiwan, 2017, pp. 986–995.

[35] K. R. Scherer, H. G. Wallbott, The ISEAR Questionnaire and Codebook, Geneva Emotion Research Group, 1997.

[36] A. Abbasi, H. Chen, A. Salem, Sentiment Analysis in Multiple Languages: Feature Selection for Opinion Classification in Web Forums, ACM Trans. Inf. Syst. 26 (2008). doi:10.1145/

[37] A. Elmadany, C. Zhang, M. Abdul-Mageed, A. Hashemi, Leveraging Affective Bidirectional Transformers for Offensive Language Detection, in: Proceedings of the 4th Workshop on Open-Source Arabic Corpora and Processing Tools, with a Shared Task on Offensive Language Detection, European Language Resource Association, Marseille, France, 2020, pp. 102–108.

[38] H. Schuff, J. Barnes, J. Mohme, S. Padó, R. Klinger, Annotation, Modelling and Analysis of Fine-Grained Emotions on a Stance and Sentiment Detection Corpus, in: Proceedings of the 8th Workshop on Computational Approaches

to Subjectivity, Sentiment and Social Media Analysis, Association for Computational Linguistics, Copenhagen, Denmark, 2017, pp. 13–23.

[39] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Ng, C. Potts, Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank, in: Proceedings of thec2013 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Seattle, Washington, USA, 2013, pp. 1631–1642.

[40] P. Nakov, A. Ritter, S. Rosenthal, F. Sebastiani, V. Stoyanov, SemEval-2016 task 4: Sentiment analysis in Twitter, in: Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016), Association for Computational Linguistics, San Diego, California, 2016, pp. 1–18.