DEPARTMENT OF COMPUTER SCIENCE

BERT-IE: A New Method for Improving Performance of Text Classifiers

Ambika Agarwal

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Master of Engineering in the Faculty of Engineering.

Thursday 4th May, 2023

Abstract

Over the last decade, larger and more complex Machine Learning (ML) models have been developed, as ML becomes more widely adopted. These require large amounts of training data which can be computationally expensive to process. This renders these models inaccessible to individuals and small organisations who do not have the data and computational resources for this.

To tackle this problem, previous researchers proposed a method, ExpBERT, which used transfer learning and Human-In-The-Loop (HITL) explanations. Including explanations using ExpBERT resulted in an increase in performance over not using explanations, NoExp. However, large representations of the input data were created, thus not solving the problem of computational expense.

In this project, I designed a new method, BERT-IE, that reduces the computational expense whilst retaining the performance for the chosen dataset comprised of tweets published during natural disasters. This scenario is a prime use case for a lightweight ML model. This dataset was also used to test the performance of ExpBERT in another researcher's implementation that I have corrected in this project.

Impressively, BERT-IE retained 98.5% of the performance reached by ExpBERT, whilst creating representations 31 times smaller and taking just a sixth of the time to pass the representations through a classifier. Therefore, this method allows individuals and small organisations to use ML and widens the scenarios in which ML can be used due to it now being quicker and computationally cheaper.

Summary of contributions and achievements:

- I designed, implemented and evaluated my new method, BERT-IE, which preserved 98.5% of the performance reached using ExpBERT, and classified tweets in a sixth of the time.
- I corrected a previous implementation of ExpBERT which resulted in a 57% speed up. The resulting version was used to obtain a new baseline set of results.
- I created a clear, easily configurable code base for BERT-IE, allowing researchers to reproduce results or develop BERT-IE further.

Acknowledgements

I would like to offer my thanks to my supervisor, Dr. Edwin Simpson, for his support, guidance and expertise throughout this project.

Declaration

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Taught Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, this work is my own work. Work done in collaboration with, or with the assistance of others, is indicated as such. I have identified all material in this dissertation which is not my own work through appropriate referencing and acknowledgement. Where I have quoted or otherwise incorporated material which is the work of others, I have included the source in the references. Any views expressed in the dissertation, other than referenced material, are those of the author.

Ambika Agarwal, Thursday 4th May, 2023

Contents

List of Figures

List of Tables

Ethics Statement

This project did not require ethical review, as determined by my supervisor, Dr. Edwin Simpson.

Supporting Technologies

During this project, the following third-party resources were used:

- Hugging Face Model Hub [33] to find pre-trained models to use in my experiments.
- Multiple common Python libraries in my code base to handle, augment and manipulate the data. These were PyTorch [68], Pandas [59], NumPy [57] and scikit-learn [77].
- TensorBoard [83] to create all of the graphs and confusion matrices.
- GitHub [21] for version control and to make my code publicly available.
- Computational facilities of the Advanced Computing Research Centre, University of Bristol [89].

Notation and Acronyms

ML	:	Machine Learning
SL	:	Supervised Learning
NLP	:	Natural Language Processing
GCP	:	Google Cloud Platform
CPU	:	Central Processing Unit
GPU	:	Graphics Processing Unit
BERT	:	Bidirectional Encoder Representations from Transformers
HITL	:	Human-In-The-Loop
AI	:	Artificial Intelligence
NLI	:	Natural Language Inference
GPT	:	Generative Pre-trained Transformer
MLM	:	Masked Language Modelling
NSP	:	Next Sentence Prediction
MNLI	:	Multi-Genre Natural Language Inference
BERT-IE	:	BERT with Inference and Explanations
CEL	:	Cross-Entropy Loss
WCEL	:	Weighted Cross-Entropy Loss
MB	:	MegaBytes
GB	:	GigaBytes

Chapter 1

Contextual Background

1.1 Overview

From the first model of a neural network in 1943 [44] to the release of an artificial intelligent, highly used chatbot [12] in 2022, the Machine Learning (ML) domain has grown vastly. With that growth has come increased complexity, for example, ML models being produced with an increasing number of layers. As the number of layers in a model increases, so does the number of parameters, and in turn (a) the amount of data needed to train these parameters and (b) the computational expense.

But keeping up with the increasing amount of data required is a challenge, in particular for the Supervised Learning (SL) form of ML that requires labelled data, which is often unobtainable. Labelled data is extensively used in Natural Language Processing (NLP) tasks, such as text classification [34], which focus on understanding spoken and written words.

My project investigates if rather than providing a text classifier with lots of labelled data, can explanations as to how a class label was identified be provided instead to improve the performance, and in turn decrease the required computational effort.

My project builds on an existing method [55] to include explanations that matched baseline results with "3–20x less labeled data", however took substantially longer to train due to an increased amount of text needing to be classified. An improvement to this method, BERT-IE, is proposed which incorporates the explanations in such a way that the performance boost is maintained, whilst the time taken to train the model is significantly reduced.

1.2 Motivation

ML has now become an ubiquitous part of daily lives, from facial recognition for unlocking phones to 'suggested posts' on social media, where an ML algorithm suggests posts a user is likely to enjoy, based on their previous interactions on the platform. Research by Pugliese et al. [65] shows the popularity of ML has doubled between 2016 and 2021 in a number of countries, such as UK, Italy and USA, with some countries seeing a six-fold increase, where popularity is measured by public interest in the topic, collected from Google Trends. This increase in popularity can also be seen in academia. The number of ML-related publications has grown from 3,886 in 2016, when ML first peaked due to the rise of deep learning, to 16,339 by the end of 2020 [65]. As a result, ML now has a global market value of $21.17 billion in 2022, which is predicted to rise to $209.91 billion by 2029 [37].

However, as with all trends there are benefits and challenges, some of which are outlined below.

1.2.1 Benefits

Undoubtedly, the biggest benefit of Machine Learning is the speed at which it can process data. With almost 328.77 million terabytes of data being produced each day [17], it would not be possible for humans to analyse trends and patterns and make full use of this at the same rate as ML models [92]. This is what

makes ML beneficial in a number of industries, such as in retail to analyse all the transactions occurring in a day.

Additionally, the accuracy and reliability of ML classifiers can outperform that of humans. For example, a recent study evaluating human vs ML performance for classifying research abstracts [23], found that the ML classifier, which used NLP, was "2-15 standard errors" more accurate than human classifiers and was consistently more reliable. Another area in which ML has been proven to outperform humans is healthcare. A study done in 2019 [87], found that ML classifiers "outperformed human experts" in diagnosing "pigmented skin lesions" and therefore "should have a more important role in clinical practice". Javaid et al. [40] notes that the "average life expectancy has increased substantially" due to technological developments, and by using ML to process and analyse data faster and with more accuracy than humans, ML can support physicians to make "timely decisions".

1.2.2 Challenges

However, to train ML models to process large amounts of data and outperform humans, a large amount of training data is needed. As found by Amershi et al. [6], "data availability, collection, cleaning, and management" is ranked as the top challenge in ML "regardless of experience level". This is particularly prevalent in SL, the form of ML that requires the training data to be labelled, as the majority of data being produced everyday is unstructured and not labelled. This means individuals or organisations wanting to use SL must first collect and label the training data, either in-house or through outsourcing. Both of these options are time consuming and expensive.

It is important to note that the training data used also needs to be specific to the task at hand, further limiting the amount of data suitable to use. For example, for an ML model to be used for classifying data gathered from social media, it first needs to be trained on social media data. This is important because of the difference in the standard of language written on social media in comparison to say academic work. Academic work must meet rigorous and formal writing requirements, whereas social media messages often include abbreviations or incorrect grammar. This is particularly prevalent in tweets, due to the limit of 140 characters. As noted by Imran et al. [36], this can "degrade the performance" of NLP models. Collecting data from numerous sources is also important to prevent biases being baked into the model [20]. For example, social media data only collected from Reddit is not likely to represent the opinions of many demographics.

Furthermore, due to the amount of data being passed through the ML models, the required computational expense is growing. As researched by Sevilla et al., [78] computational requirements were "doubling roughly every 20 months" (Moore's law [10]) before 2010. However, since 2015, ML models requiring "10 to 100-fold" more computational effort have been developed. This can also be very expensive for individuals or organisations, in particular if computational effort is outsourced to the cloud. In cases where high computational power is not available, training the ML model and then using it to make predictions can take numerous days. This is not suitable for data that needs to be analysed in real-time, for example in emergency situations.

1.3 Example use cases

Although there are a growing number of ML models, as mentioned above many require large amounts of training data. However, there are many scenarios in which there is limited available data, particularly labelled data, and/or limited computational resources. In these cases most current ML models cannot be utilised. In contrast, there are many experts available who could label the data, but using their resources and expertise can be very expensive. This section outlines two of these scenarios which illustrate the need for lightweight ML models that can be trained with minimal data and run by individuals or small organisations.

1.3.1 Scenario One - Sentiment analysis to predict the stock market

Scenario 1a) ML has already become an integral part of the finance domain, in particular the stock market through the use of automated traders. Goldman Sachs found automated traders performed more accurately than human traders and as a result between 2017 and 2018 99% of human traders were replaced [56]. This is a similar story in other large financial organisations. This use of ML is suitable for large companies as it relies on collecting historical trading data, of which there is lots, and training the model

on this data [38]. The ML model can then be used to make predictions on how the stock market will change and what trades are best to make. However, this is only suitable for large organisations as they will have the resources to run such large ML models with all the historical data, which individuals will not.

Scenario 1b) Another use of ML, particularly NLP, in the finance domain is performing sentiment analysis on live data to predict how the stock market may be affected. For example, when layoffs occur, companies can experience a rise in their stock price, as it indicates they are focusing on profit [74]. For example, Bloomberg found big tech giants, such as Alphabet and Meta, saw an average "5.6% bump in their stock price" in January 2023, following their announcement of mass layoffs [80]. News of these layoffs may be disseminated through news articles or through social media and can entice an individual to invest in the company. ML models would be useful in this situation to extract the key data from these sources. However, to train the ML models there isn't a vast amount of data available. Furthermore, individuals wishing to use this may not have the resources to train a large model and have it predict changes to the stock market in real-time to keep up with the pace at which the stock market changes. Therefore, a lightweight ML model that can be trained on a small number of tweets or articles and run quickly and relatively inexpensively by an individual is needed.

1.3.2 Scenario Two - Gathering reviews on a product or service

ML can also be useful to small companies wanting to gain organic reviews of a product or service they produce, and analyse the reception of unique features they have implemented. But research shows small companies (less than 500 employees) are "four times less likely" to use ML than larger companies [7]. This may be due to a variety of reasons. Firstly, small companies are highly unlikely to have their own data centers with advanced computers, like larger companies such as Google do [25]. Therefore, small businesses must outsource and use Cloud resources, such as those provided by Microsoft Azure [50] or Amazon Web Services [4]. These can be complicated to set up due to the level of technical knowledge needed in-house, despite the user interface being highly simplified to use. For example, just to calculate the price of using cloud resources on Google Cloud Platform (GCP) [24], one must know the number of CPU's or GPU's they require, as well as the RAM usage and number of threads per core. It is unlikely a small business wanting to understand the reception of their product will have the technical expertise in-house to understand this.

Furthermore, to be able to make use of an ML model and get real-time results, a GPU will need to be used. For the simplest type of "accelerator-optimized" GPU available on GCP, this will cost approximately £600 per month to run on weekdays for seven hours [26]. This may not be affordable for many small businesses and thus highlights the need to have ML models that do not need large amounts of training time.

Additionally, for a small business to be able to analyse the reviews of their product or service, the SL model must first be trained on labelled data, as mentioned above. To label the data a large workforce may be needed and in some cases, expertise. Once again, this is feasible for large companies in-house but smaller companies will likely have to outsource. Currently, there are crowdsourcing platforms that can label data, such as FigureEight (formerly known as CrowdFlower) [5], Amazon Mechanical Turks (MTurk) [3] and Toloka [86]. On MTurk, for example, companies can choose how much they pay workers, and must also pay a fee to Amazon depending on the amount paid to the workers. This can quickly add up. An alternative method of labelling data is using chatbots, such as ChatGPT [12], however it is very difficult to control the biases that may come into the data, and therefore it would be preferable to perform the labelling in-house.

This highlights the need for ML models that can be trained with relatively small amounts of data that can be labelled in-house or for a small price through outsourcing.

1.4 Related work

My project makes use of pre-trained models - large ML models trained on a vast amount of data, as further explained in Section 2.5. The chosen pre-trained model for this project is Bidirectional Encoder Representations from Transformers (BERT) [15], which is trained for 11 different NLP tasks, using a corpora of 3.3 billion words.

This pre-trained model was used by Murty et al. [55] who created a novel way of providing explanations to a classifier to aid performance. This method was called ExpBERT. Their implementation was tested on a relation extraction task and an artificial dataset. Although an increase in performance was seen from the inclusion of explanations, the way in which they were included resulted in large amounts of data being passed through the pre-trained model, hence rendered the method very computationally expensive.

Pilitsis [62] then adapted ExpBERT to work for document classification, and measured the performance using CrisisNLP [36]. However, the problem of large data and computational expense still persisted.

My project builds on the work of Murty et al. [55] and Pilitsis [62] to create a new method that also includes explanations, but tackles the computational expense of ExpBERT, and therefore meets the need required by individuals, small organisations and businesses.

1.5 The dataset

The dataset chosen to use in this project is CrisisNLP [36], a corpora of annotated tweets collected during 19 crises between 2013 and 2015. 52 million messages published in locations across the world and at different times of the year, were collected from Twitter. A small subset of these messages (50,000) were then annotated by volunteers or paid workers, with the label for each annotated message needing to be agreed by three unique workers. Nine label annotations were used, derived from annotations used by the United Nations Office for Coordination of Humanitarian Affairs [88]. Three example messages, randomly sampled, from the dataset are provided in Table 1.1.

tweet_id	**tweet_text**	**label**
'501351321195536385'	#India: #Bihar floods toll rises to nine, 11 lakh affected - http://t.co/kHNRWRXdvf	injured_or_dead_people
'541226376003469312'	Keep praying guys. Stay safe #RubyPH	sympathy_and_emotional_support
'503915980624367617'	Earthquake magnitude 6 in CA, . it's usual but damage is small in japan. USA must think safety of building. #earthquake	other_useful_information

Table 1.1: Table showing three example datapoints from the dataset, randomly sampled

Out of the subset of annotated tweets, only 22,099 were annotated by paid crowd-source workers from CrowdFlower (now known as Figure Eight) [5], and were therefore suitable for use in this project. 3,132 tweets out of the 22,099 were not related to natural disasters, rather were about infectious diseases and viruses and therefore were unsuitable for this project, leaving 18,967 tweets. Finally, having removed non-English or duplicate tweets, there were 17,117 tweets suitable for use in this project, which going forward will now be referred to as 'the dataset'. Figure 1.1 visualises the distribution of the labels within the dataset.

I chose to use CrisisNLP as the task of analysing tweets published during a natural disaster is a prime use case for a lightweight ML model that runs with reduced computational resources and data. A recent study by Murthy and Longwell [54] found that Twitter was used during the 2010 floods in Pakistan as a voice of the affected residents. This use of Twitter, as a microblogging platform, has continued, and the information shared in these tweets can be helpful to disaster relief charities to allocate aid and resources, as well as prepare for future natural disasters. However, there is only limited data available on Twitter, and disaster relief charities are unlikely to have access to high performing supercomputers to process this data quick enough in line with the time pressures in a natural disaster, using classic ML models. Again, this brings about the need for a method of analysing data, with minimal labelled data and computational resources.

Finally, CrisisNLP has been used in a variety of ML research, such as testing the "effectiveness of Manifold Mixup" [72] and the "application of different neural networks" [2], suggesting it is suitable for a range of ML tasks.

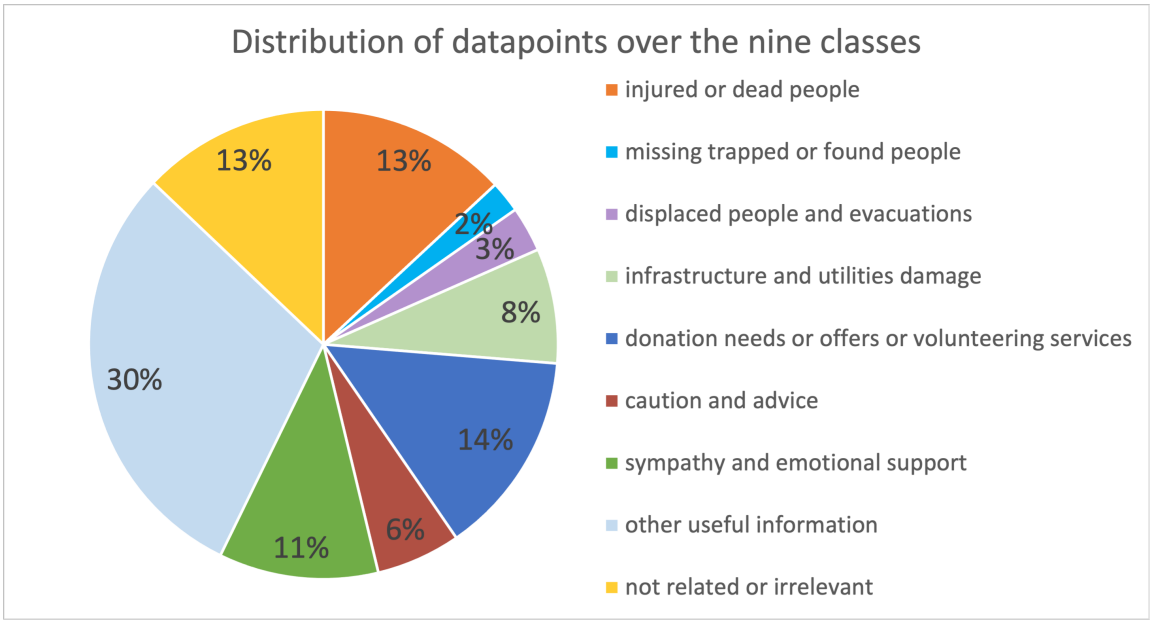


Figure 1.1: Pie chart visualising the distribution of class labels within the dataset

1.6 Aims and Objectives

The high-level aim of my project is to create a unique method that preserves the performance increase seen in ExpBERT [55], through the use of explanations, whilst being significantly less computationally expensive. More specifically, my objectives are:

1. Understand the ExpBERT method proposed by Murty et al. [55] and identify the benefits and hurdles of using it.
2. Design a new method for incorporating explanations that preserves at least 90% of the performance seen in ExpBERT, whilst overcoming the hurdle of large computational expense.
3. Implement the new method in a clear, simple way, following best practices to allow the code base to be easily configured for different experiments and used in further research.
4. Assess the performance of the proposed method on the CrisisNLP dataset [36] by running a series of experiments and tuning the hyperparameters of the models created.
5. Compare the performance and computational costs of using the proposed method to using ExpBERT and present the trade-offs of each.

1.7 Dissertation outline

Chapter 1, this section, provides a contextual background to my project. Chapter 2 provides a brief technical introduction into the concepts used in both ExpBERT and the proposed method. Chapter 3 provides details of the project execution, and Chapter 4 discusses the results obtained. Finally, Chapter 5 summarises the the work done in my project and suggestions for further work are outlined.

Chapter 2

Technical Background

As introduced in Chapter 1, there is a need for ML models that can perform well when trained on limited amounts of labelled data whilst consuming minimal computational resources. One way of reducing the amount of labelled data required is by leveraging knowledge stored in pre-trained models, through transfer learning. We can then further reduce the amount of labelled data needed by including Human-In-The-Loop (HITL) error correction, in the form of explanations. This chapter presents the fundamental concepts needed for this project. These include NLP, neural networks and transformers, as well as transfer learning, which makes use of the transformers, and HITL which incorporates explanations.

2.1 NLP

To be able to process large amounts of text in a dataset, NLP is required. NLP is a branch of Artificial Intelligence (AI) that allows computers to understand text in the same way as humans do [35], not just understanding single words but also the context surrounding them. It can be used for many tasks, including sentiment analysis, speech-to-text translation and document classification [52]. A common example of document classification is spam content detection in emails [43]. In this project I make use of document classification - taking textual inputs (tweets) and mapping them to a category (one of the dataset labels).

2.1.1 Natural Language Inference

The use of explanations in addition to the tweets, as done in ExpBERT, means the relationship between the tweet and explanation needs to be comprehended by the model. A task within NLP called Natural Language Inference (NLI) is suitable for this [49]. NLI takes two sentences: a premise and a hypothesis. The task is then to identify whether the premise and hypothesis have an entailing, neutral or contradictory relationship.

For example, a premise may be "my house has collapsed due to the earthquake and I am trapped inside" and the hypothesis may be "I am warm and safe in a shelter". It is clear to humans that in this case the premise and hypothesis have a contradictory relationship. Conversely, with a premise stating "my house has collapsed due to the earthquake *but I was able to leave on time*" and a hypothesis of "I am warm and safe in a shelter", we can understand there is an entailing relationship between the premise and the hypothesis.

This task can be applied to the chosen dataset, where the tweet is the premise and explanations provided as additional input act as the hypothesis. Further details of this application are described in Section 3.1.3.

2.2 Neural Networks

The first neural network model developed in 1943, marked the start of deep learning [44], and is now extensively used in NLP [16]. These networks take "a vector of input values" [42] and produce a single output value or a vector of probabilities. Classification tasks, such as binary classification and multi-class

classification, make use of neural networks, as they have the ability to non-linearly learn which features from the input are indicative of the output.

Figure 2.1 produced by Jurafsky and Martin [42], shows a diagram of a neural network, that produces a vector of probabilities. Neural networks are comprised of a series of layers, with a minimum of an input and output layer and a varying number of hidden layers. The more hidden layers included, the 'deeper' the network. This can be beneficial as the network is able to pick up on more nuances from the input layer, however too many hidden layers can result in overfitting and an increased amount of computational time to pass the input through.

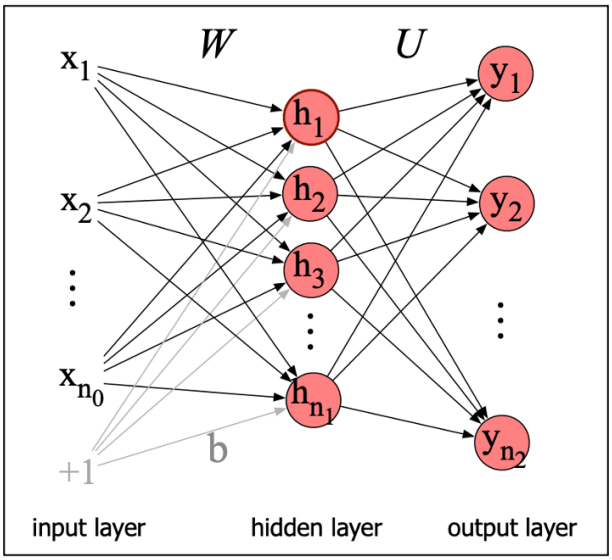


Figure 2.1: Diagram of a neural network, produced by Jurafsky and Martin [42].

Each layer in the network is made up of neurons; there can be the same or different number in each. Each neuron takes in the value from each neuron in the previous layer, weights and a bias to produce a "weighted sum" of inputs plus the bias (output). This can be represented by Equation 2.1, where z is the output of the neuron, $\mathbf{w}$ is the vector of all the weights passed in, $\mathbf{x}$ is the input vector and b is the bias term. Figure 2.2 adapted from [42], provides a visual representation of this.

$$z = \mathbf{w} \cdot \mathbf{x} + b \tag{2.1}$$

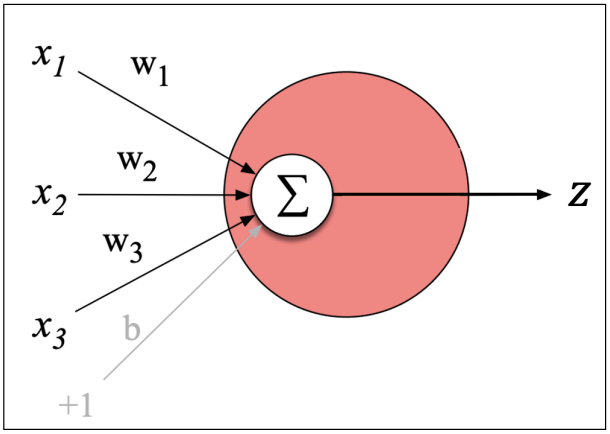


Figure 2.2: Visual representation of the weighted sum calculated for each neuron, adapted from [42].

For the neural network to perform well and produce values similar to the true labels, for example for a multi-class classification task to predict the correct class, the weights in the network, also known as parameters, need to be updated during the training phase. These weights are usually randomly initialised, and therefore the performance of the network is low at first. Using the backpropagation algorithm [98], these weights are refined with the aim of the training to adapt the weights such that the predicted values, $\hat{y}$, are as close as possible to the true values, y. This requires a loss function and optimisation algorithm. As error backpropagation is not the main focus of the project, details have not been provided, and it is sufficient to know that the weights are randomly initialised and then adapted during the training phase, with the help of a loss function and optimiser.

Having looked at the general structure of a neural network, attention can be focused to neural networks used for NLP. These networks include an embeddings layer that is used to map individual words to low dimensional vectors. This layer can be either trained like the other hidden layers in the network, or 'frozen' which means it makes use of pre-trained embedding representations, such as word2vec [51] or GloVe [60].

2.3 Transformers

Transformers are a type of neural network used in NLP, proposed in 2017 by Vaswani et al. [91]. They can be used to solve a variety of tasks, such as question answering, summarising text, generating text etc. Since 2017, numerous more transformers have been invented, such as a Generative Pre-trained Transformer (GPT) [69], which is an 'auto-regressive' model, BERT [15], which is an 'auto-encoding' model, and BART [47], which is a 'sequence-to-sequence' model. Over the years better performance has been reached by creating bigger models trained on larger amounts of data. Bigger models can mean more layers and in turn more parameters (weights), as introduced above in Section 2.2. For example, in June 2018 OpenAI created a (GPT) [69] that had 110 million parameters. By 2019, a new GPT model had been released, GPT-2 [70], which had 1500 million parameters. This is a 13-fold increase in the number of parameters, in less than a year.

However, despite the large amounts of data the models are trained on, they do not perform well on specific practical tasks, due to their self-supervised nature. Self-supervised means they do not take in human input, for example labelled data, but rather their "objective is automatically computed from the inputs of the model" [18]. For this reason, transformers need to be fine-tuned with human input to perform well on a task. Fine-tuning is part of transfer learning, a commonly used technique of transferring weights, and is discussed further in Section 2.5.

2.3.1 Architecture

A transformer can have up two parts: an encoder and a decoder. The encoder takes inputs and produces embeddings, which as mentioned are low dimensional numerical representations. Decoders can use the embeddings created by the encoder, and/or their own input to predict an output. Decoders are most useful in text generation, whereas encoders are more suitable when understanding of the full sentence is needed, for example in classification. The use of both an encoder and a decoder is mostly used for summarising tasks. Given the task at hand in this project is text classification, the remainder of this chapter will focus only on encoders.

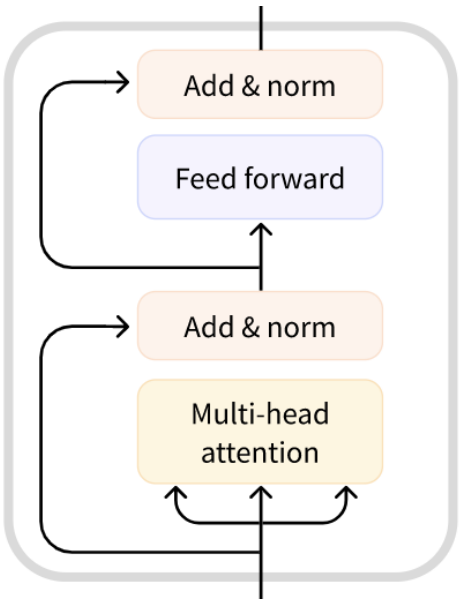


Figure 2.3: Illustration of a transformer block, created by Hugging Face [18]

Due to the bi-directional nature of encoders [91], transformers are able to understand the context of a particular word, i, from both the words preceding i and the words following i. For example, given a sentence, 'I like apples', the transformer will be able to comprehend that the sentence starts with a pronoun, 'I' and ends with a noun, 'apple'. Taking into account this information, a numerical representation in vector form of the word 'like' can be produced. Due to the context being taken into account, each word will not have identical embeddings. For example, given a second sentence 'this orange juice tastes like apple juice', the word 'like' will have a different embedding for this sentence, as it is used in a different context to show similarity. Comprehending the context is made possible by the use of self-attention.

A self-attention layer is one of the layers in a transformer block. Figure 2.3 provides an illustration of the transformer block [18], which can be stacked to further encode the input. The first layer is an attention layer which informs the model which words to pay attention to from the representation. As the layer can compare a word to other words, it can "reveal their relevance in the current context" [42]. To do this comparison, the dot product is used, with a large vector output

signalling that the compared words are similar. This comparison is done for each word and finally softmax is applied to gain a proportional relevance of each previous word to the word being compared.

Due to the complexity of natural language, it is difficult for just one self-attention layer to measure all the types of relationships. Therefore, a multi-head attention layer is used, which is made up of sets of self-attention layers [42]. This allows different word relationships, such as antonym or discourse, to be focused on. The outputs from each layer are then projected down to the size of the input, ready to be normalised.

Layer normalisation is used to normalise the values in the layers so they do not get too big or too small. This uses z-score normalisation and new vectors are produced by the following equation:

$$\hat{x} = \frac{x - \mu}{\sigma} \tag{2.2}$$

The normalised values are passed through a neural network, as described in Section 2.2 and again normalised. Finally, these layers are also connected via residual connections [42], represented in Figure 2.3 by the black arrows. These allow information to bypass an intermediate layer so higher layers can access information from lower layers.

2.3.2 Tokenization

In order to fully understand how transformers work, it is important to understand how the raw text input is converted into an input that can be processed by the transformer. This process is called tokenization.

Before any training can begin, the raw text is split into words and subwords by a tokenizer and then translated to numbers (ids) that can be recognised. There are three tokenization methods that can lead to different sized vocabularies.

- **Word-based tokenization:** in this case the raw text is split up into whole words, however it can struggle on punctuation that appears after a word, as well as contractions.
- **Character-based tokenization:** to overcome punctuation and contraction issues, the input can be split into individual characters. This forms a large vocabulary and means it is hard to obtain the context of the input. For example, the tokenizer will struggle to understand if the character "a" was it's own word or part of a larger word.
- **Sub-word tokenization:** to overcome the problems of the above two methods, the input can be split into words and subwords that appear in the vocabulary. For example, the word "interestingly" would be split into "interesting" and "ly". This is beneficial as it can decrease the size of the vocabulary.

2.4 BERT

Bidirectional Encoder Representations from Transformers (BERT) [15] is a popular encoder, developed in 2018, and the most suitable type of transformer for the text classification task in this project.

The BERT architecture contains 12 transformer blocks, giving 110 million parameters. A larger BERT model, BERT-large has also been developed that has 24 transformer blocks with 340 million parameters. BERT was trained over four days on 3.3 billion words collected from Wikipedia and Google's BooksCorpus [15]. As mentioned above in Section 2.3, due to the self-supervised fashion in which transformers automatically create their objective [18], none of the data required labelling. BERT-large also took four days to train, but required four times the number of processing units than standard BERT.

BERT makes use of the subword tokenization algorithm, WordPiece [18], which gives a vocabulary of size 30,000. The output ids are stored in a list called "input_ids". BERT tokenizer also returns "token_type_ids" which distinguishes which ids relate to which passed in input, and "attention_mask" which informs the model which ids relate to the input, and which ids are 'padding' so that the output is always the same size.

During training, the 110 million parameters were adapted for two tasks simultaneously: Masked Language Modelling (MLM) and Next Sentence Prediction (NSP). MLM, also known as the cloze task [82], leverages

BERT's bidirectional ability in order to predict a masked word in a sentence. For example, in a sentence 'I [MASK] apples, they are tasty.' the words before and after the [MASK] are needed to understand the context and predict the missing word. NSP makes use of an encoders ability to understand full sentences. The task is to predict if one sentence follows another. For example, the sentences 'I like apples.' and 'I also like oranges.' do follow each other, but 'I like apples.' and 'I fell over today.' do not.

After full training, BERT achieved state-of-the-art accuracy on benchmark tasks and even outperformed humans on some [15]. Some benchmark tasks included question answering using the SQuAD dataset [71], sentiment analysis, and paraphrasing using the GLUE dataset [93]. The code base for BERT is open-source, allowing developers to fine-tune it, saving time and computational resources.

2.4.1 Variations of BERT

Since the release of BERT in 2018, other models with similar architectures have been developed.

One notable model is distilBERT [73], which has been branded as "smaller, cheaper, faster, lighter". DistilBERT uses knowledge distillation, a method for transferring knowledge from large models to smaller models that can mimic the large models at each level of the network [64]. The smaller model only has six transformer blocks (reduced by a factor of 2) and 40% less parameters than BERT, and as a result runs 60% faster. Despite this reduction in size, distilBERT still manages to maintain "over 95% of BERT's performance" [73], measured on the GLUE dataset benchmark [93]. This model is cheaper to pre-train and to fine-tune.

Another notable model is 'Robustly optimized BERT approach' (RoBERTa) [48]. The authors found that "BERT was significantly undertrained" and changing some hyperparameter values, such as increasing the batch size and learning rate, can result in an increase in performance between two and 20%. As the research was a "replication study of BERT", the architecture of RoBERTa is the same as BERT's. Importantly, RoBERTa was not trained on the NSP task, only the MLM task, but was trained on 1000 times more data [81].

There are many more models, such as ALBERT [46] and TinyBERT [41], all of which were developed with slightly different aims, and most of which are publicly available to use from the Hugging Face Model Hub [33], allowing developers and researchers to explore the code base and use the model for transfer learning.

2.5 Transfer Learning

Transfer learning enables knowledge, such as weights stored in a neural network, to be leveraged in new models. As introduced in Section 2.4, there are models that have been trained on a large dataset on a large scale, usually using very high-specification supercomputers, unfeasible for the general public [90]. These models are known as pre-trained models and are commonly created by large companies, such as Google and Microsoft.

Some common pre-trained models are:

- BERT [15] - as introduced in Section 2.4, this model was trained on over 2.5 billion words from Wikipedia and when released obtained state-of-the-art results on benchmark NLP tasks, such as sentiment analysis and identifying contradictions, similar tasks to the task at hand in this project.
- ImageNet [14] - this model is trained on over 14 million hand-annotated images, and is prominent in the Computer Vision research area.
- GPT-3 [11] - this model has 175 billion parameters and can generate text, given an input. It is this model that is behind ChatGPT, a tool that has "everyone talking about it" [12].

The pre-trained models can be used as a "starting point" to train a new model for a different, but closely related task, known as the downstream task, where there may not be as much data available [39]. This is beneficial as rather than the weights in a neural network being randomly initialised, they can be initialised with trained values. This means the neural network is likely to need less training to perform well on the downstream task, and in turn this is less computationally expensive. Due to the size of the datasets that the pre-trained models are trained on, the parameters learnt are general enough to be "suitable to both base and target tasks" as researched by Yosinski et al [99].

Transfer learning can take two forms: fine-tuning or feature extraction [85]. In feature extraction, a new classifier, called the model head, is added on top of the layers in the pre-trained model. The model head is trained from scratch and is usually only a few layers, primarily used to map the output from the pre-trained model to the new task.

In fine-tuning, some, if not all, of the layers of the pre-trained model are re-tuned, along with the model head, so the neural network is more applicable to the task at hand. Usually, only the final layers of the pre-trained model are re-tuned as they hold most of the task specific information, which does not generalise as well to the new task [85]. Re-tuning these final layers can give better performance over feature extraction [61]. However, fine-tuning can take considerably longer, as more layers and weights need to be trained.

Therefore, there is a trade-off to be made between performance and training time, with the balance shifting depending on how closely related the pre-trained model task and the target task are. In this project, I will be using feature extraction as there are existing pre-trained models that have been trained on tasks closely related to the task of classifying tweets.

As introduced by Torrey and Shavlik [[58] as cited in [39]], three benefits of using transfer learning, over training a model from scratch are:

1. Higher start in performance (before any training)
2. The rate at which the model improves is higher (higher slope)
3. The converged performance of the new model is higher (higher asymptote)

Figure 2.4 provides a graphical representation of these benefits.

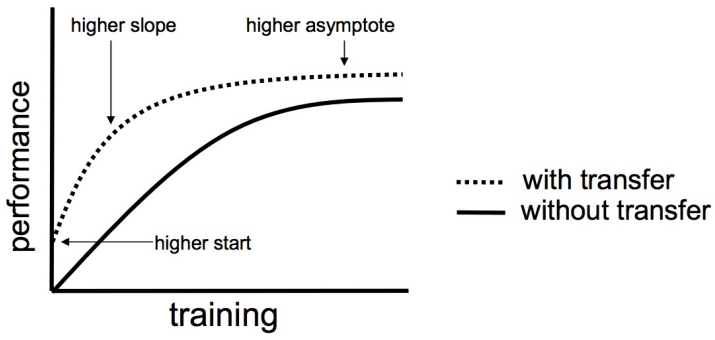


Figure 2.4: Graph illustrating the difference in performance with and without transfer learning [[58] as cited in [39]]

There is also an environmental benefit to using transfer learning. As transfer learning results in new models starting with higher performance and improving at a higher rate, less iterations are needed during training to reach or exceed previous expected performance. This means less processing time thus reducing the carbon emissions produced during training.

It is important to consider, however, the drawbacks of transfer learning. One drawback is the potential transfer of bias. If the pre-trained model is trained on a very large, but bias dataset, for example an imbalanced dataset where high credit scores are only given to men, or data is primarily collected from selected regions, as is the case in ImageNet [18], the model can quickly learn biases. These biases can then be transferred to the new model, thus resulting in bias outputs. Another drawback is the time taken to pass the new data through the pre-trained model. This is particularly prevalent where pre-trained models have multiple layers, as this can result in millions of parameters, for example Resnet-50 which has 50 layers with over 23 million parameters [29]. To overcome this, some large pre-trained models have lighter versions, such as BERT [15] has distilBERT [73].

2.5.1 Application

One application of transfer learning is sequence classification, which is used in this project.

Standard sequence classification

As described in Section 2.4, a [CLS] token is prepended to each input sequence. This is passed through the pre-trained model and by the final layer it holds a representation of the entire input sequence. This is a vector of size 768, signified by Y_{CLS}. Y_{CLS} forms the input to the model head, which has weights W_C, and can map the input to a distribution of the class labels. To convert this output to a probability

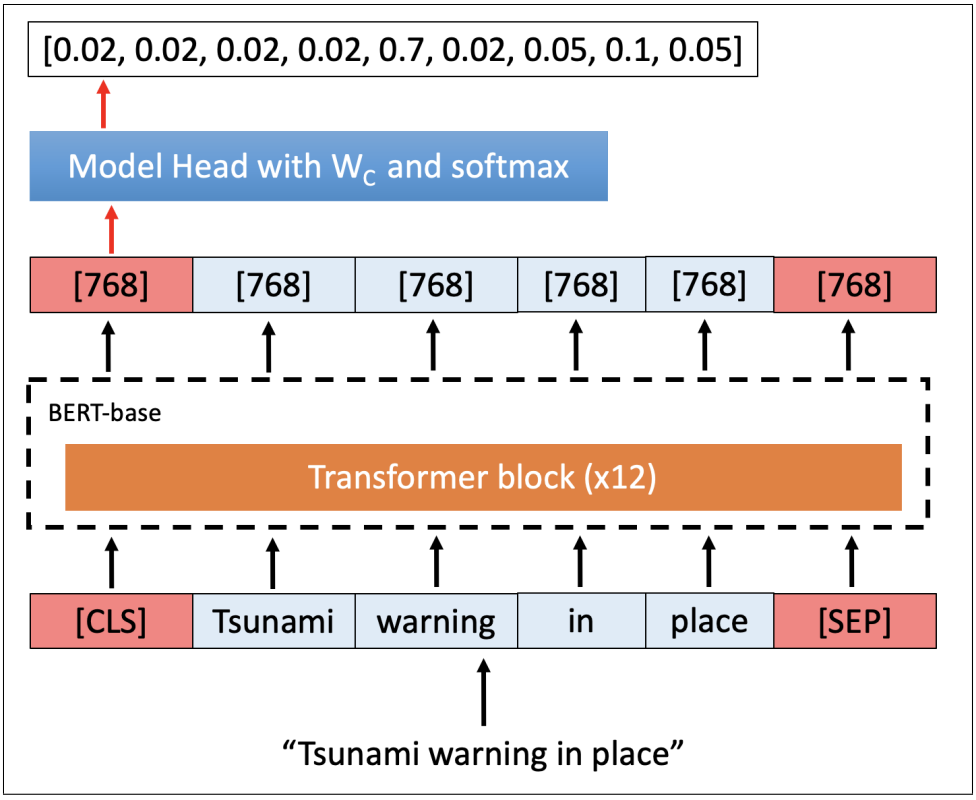


Figure 2.5: Visualisation of a input sequence being passed through a pre-trained model and a model head ('NoExp').

distribution, D, softmax is applied. This process is represented by Equation 2.3 [42] and visualised in Figure 2.5. Going forward, this process will be referred to as '**NoExp**'.

$$D = softmax(W_C * Y_{CLS}) \tag{2.3}$$

Pair-Wise sequence classification

A slight variation to the standard sequence classification is Pair-Wise sequence classification.

This is used for NLI tasks, where a premise and hypothesis are provided, and the objective is to comprehend the relationship between the two, as outlined in Section 2.1.1.

In this case the [CLS] token is passed through the pre-trained model and model head in the same way, but it now holds the "model's view of the input pair" [42]. As mentioned previously, the input pair are separated by the [SEP] token.

A common dataset used for the NLI downstream task is Multi-Genre Natural Language Inference (MNLI) [95]. This is a dataset with 433,000 annotated sentence pairs. An example model fine-tuned on this dataset is `huggingface/distilbert-base-uncased-finetuned-mnli`, which uses distilBERT [73] as the pre-trained model.

2.6 Human-in-the-loop

Having reviewed transfer learning, a way of reducing the amount of data needed and in turn computational costs, attention can be focused on further reducing the amount of required labelled data through HITL error correction.

HITL allows both the human and ML model to "interact continuously" [96] with the humans giving feedback on what the model outputs. This allows the model to learn from errors, in a similar way to how humans do. For example, feedback given to a child on where their spelling is incorrect will help them

to identify their mistakes and learn the correct spellings, rather than if they were presented with 50 new words, as is done in ML. As such, HITL is particularly useful in cases where human expertise is essential, for example in healthcare when predicting diagnoses of patients. Using human feedback provides the opportunity to check the model is performing well, and provide it with more information in cases where it is not.

HITL is of growing interest, due to the reduction it provides in the amount of labelled data needed, due to the expertise fed in. From 2016 to 2021, the amount of Google Scholar searches mentioning "human-in-the-loop" and "machine learning" grew by 7-fold [97].

Holmberg et al. [30] defines three approaches to HITL in ML, as further explained by Mosqueira-Rey et al. [53]. These are:

1. **Active Learning** - in this approach, the model can ask the humans for clarity on cases it is unsure about, forming a semi-supervised model. Using this approach means feedback provided is irregular.
2. **Interactive ML** - in this approach humans again supply information, but this time in a more structured, regular way, than in active learning.
3. **Machine Teaching** - finally, this approach gives humans the control over the learning process and allows them to "transfer their expertise" [53].

As found by Hartmann and Sonntag [28], including human explanations into ML models is "considered a promising way" of interaction and one that leads to better models. Godbole et al. [22] further found that models provided with reasoning can "often learn more and faster" than models without, hence reducing the need for annotated data. Therefore, I have chosen to provide explanations to the classifier along with the input (tweets), as a means of transferring the "expertise" [53]. This falls into the Machine Teaching approach.

2.7 Related work

Finally, having reviewed the concepts of transfer learning and HITL, it is important to discuss the research area in which this project lies to understand the impact of this project. This section will outline some existing literature that makes use of HITL, in particular passing in explanations, and transfer learning.

2.7.1 Overview

Research done by Wiegreffe and Marasovic [94], found that there are two "most prominent explanation types" for improving NLP models. The first is 'highlight'; key words from the input text are highlighted to indicate to the model which words it should pay attention to in order to make a prediction. The second type is known as 'free text' which contains reasoning about why a specific label was reached. Free text explanations do not *need* to include words from only the input data, and therefore can be more informative and suitable for complex tasks, such as NLI. Therefore, research using free text explanations only has been considered in this project.

The first notable work, by Srivastava et al. [79], uses semantic parsers to comprehend the passed in explanations, which were developed through crowdsourcing, into a logical form. They did not use a pre-trained model, rather they used a naive bayes classifier. For the document classification task, used by Srivastava, an improvement in performance was found through using explanations.

A similar work by Hancock et al. [27] also used semantic parsers to interpret the explanations given for unlabelled data. For example, if a task is to identify if two people, x_1 and x_2 are married, and an explanation returns 'True because the words "his wife" are right before person 2', the parser will only return 'True' for future inputs where "his wife" appears before x_2. Of course, this is not ideal as the sentence could be rearranged so "his wife" appears after the wife's name.

This suggests semantic parsers are most suitable for low-level explanations. Therefore, Murty et al. [55] created a method, ExpBERT, that uses "modern distributed language representations" to interpret more complex, natural language explanations, resulting in a fine-tuned model. My project builds upon this.

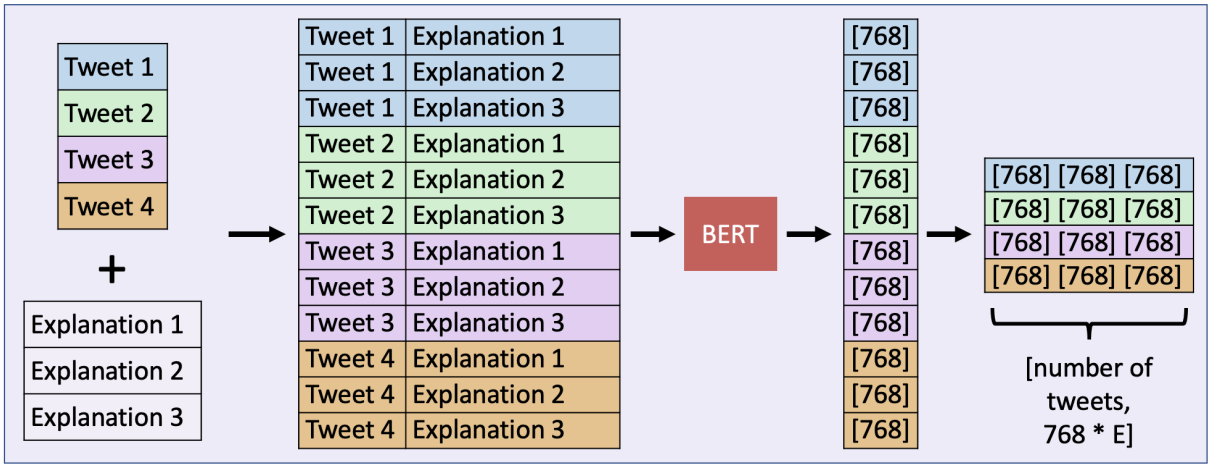


Figure 2.6: Visual representation of the data flow using ExpBERT

2.7.2 ExpBERT

ExpBERT makes use of both transfer learning and HITL (free-text explanations). The pre-trained model used is BERT, finetuned on the MNLI dataset, and the task is relationship extraction to pick out spouses in the text. This was the same task used in Hancock et al [27].

Figure 2.6 provides a visual representation of how explanations are incorporated with the input data, and passed through to the model head. In this figure, the input data is represented by tweets.

Firstly, the cartesian product of the input data and explanations are formed, giving an expanded dataset where each tweet is concatenated with each explanation. In the diagram there are four tweets, and three explanations, giving an expanded dataset of size 12. The explanations were created by the authors who were given at most one minute to create each. Each explanation was not meant to apply to one datapoint directly, rather to the dataset as a whole. To ensure, however, that each class was represented, textual descriptions of the classes were also used, which provided information about each class. These did not need to be created as they were part of the dataset. The way in which the textual descriptions are included and processed does not differ from the explanations, and therefore the term explanations will be used to refer to both together.

In the expanded dataset, the tweet (x) and the explanation (e_j) are prepended with a [CLS] token and separated with a [SEP] token as introduced in Section 2.4. This can be represented by Equation 2.4 [42], where I is the embedding created, a vector of length 768.

$$I(x, e_j) = BERT([CLS], x, [SEP], e_j) \tag{2.4}$$

As this embedding is created for each tweet and explanation combination, the vectors need to be concatenated, for each tweet, to form "the explanation representation", $v(x)$, which is of length 768 * E, where E is the number of explanations. This can be represented by Equation 2.5 [42] and visually seen in the right of Figure 2.6. As the example had three explanations, for each of the four tweets, three vectors have been concatenated.

$$v(x) = [I(x, e_1), I(x, e_2), ..., I(x, e_E)] \tag{2.5}$$

Once concatenated, these embeddings are known as "explanation guided representations of the input" [55], and are passed in to the model head for classification.

ExpBERT reached state-of-the-art results, most notably seeing a "3–10" point increase in performance, measured using the F1 score, (further explained in Section 3.3.3) over NoExp (Section 2.5.1). To ensure the increase was seen due to the "diverse set of high quality explanations" rather than due to an increased number of features, some explanations were replaced with random sentences. As expected, the model did not perform well in this case.

Whilst appreciating these state-of-the-art results, it is important to discuss the drawbacks of using this method. The first drawback is the datasets used to measure the performance. Although three datasets were chosen, these were only small datasets. Additionally, the dataset which saw the most improvement in performance was an artificial dataset. Therefore, it cannot be said with confidence that the performance boost seen would also apply to real-life datasets.

Secondly, due to the size of the expanded dataset created as a result of the Cartesian product, it can be very time consuming to pass each datapoint from the expanded dataset through the pre-trained model. This requires a lot of computational time, as well as memory to store the expanded dataset. Furthermore, the "explanation guided representations of the input" are E times bigger than in NoExp. It is trivial to understand that this also increases the time taken for each tweet representation to be passed through the model head. Hence, this renders the ExpBERT method incredibly computational expensive.

This provokes the need for a new method that can encapsulate all of the information in a smaller representation, with it's performance tested on a real-life dataset. In turn, a smaller representation would reduce the computational resources required, the objective of this project.

2.7.3 Implementation of ExpBERT

To test if ExpBERT maintained the performance boost seen with an artificial dataset, Pilitsis [62] tested ExpBERT on CrisisNLP [36] - a dataset comprised of tweets published during natural disasters, as introduced in Section 1.5. The task of classifying tweets meant that Pilitsis had to adapt ExpBERT for document classification, as Murty et al. [55] had developed ExpBERT for relation extraction.

As this implementation was adapted for document classification and Pilitsis [62] had tested this adaption on CrisisNLP, the same dataset being used in this project, this seemed to be a suitable research to build from and compare results to.

However, upon inspection of the code and methodology used by Pilitsis [62], it became apparent there were some technical errors. This included the application of softmax twice to the results, which harmed performance, as well as only partial information from the pre-trained tokenizer being passed into the pre-trained model. Therefore, this deemed the results unsuitable. It was still possible, however, to build off the code base once the technical errors were rectified. Consequently, all results for NoExp and ExpBERT used in this dissertation for comparison are produced using the corrected code.

Chapter 3

Project Execution

3.1 Design

Having explored the research area, it was clear to see that HITL input did improve performance of ML models. However, the method by which natural language explanations were included, ExpBERT, resulted in the representations consisting of many high dimensional vectors, that needed to be passed into the model head. Therefore, there is a need for a new method that includes HITL explanations in a way that results in lower dimensional outputs, thus decreasing the time taken and computational requirements to classify the tweets.

3.1.1 Criteria for a new method

The first step in designing this new method was to outline a criteria. The new method needed to:

1. comprehend the relationship between the tweet and the explanation, in order to identify which explanations were most helpful for tweets containing certain key words.
2. produce embeddings which are significantly smaller than those produced by ExpBERT. Those were (768 * E) where E was the number of explanations provided.
3. take less computational time than ExpBERT, as a result of having smaller embeddings.
4. improve performance (measured using the F1 score) over NoExp, to make it worthwhile to have explanations.
5. give comparable results to ExpBERT to make using the smaller method worthwhile.

Meeting these criteria would fulfil Objective 2, outlined in Section 1.6.

3.1.2 Use of NLI

In order to meet criteria 1 & 2, I chose to use NLI, as introduced in Section 2.1.1. NLI takes two sentences: a premise and a hypothesis and returns if the relationship between them is contradictory, entailing or neutral. When using the tweets as the premise, and the explanations as the hypothesis, NLI can be used to understand and identify the relationship between the tweet and the explanation, thus meeting criterion 1.

When the tweet and the explanation are passed through the NLI model, a representation of size three will be produced. This is the probability of the relationship being contradictory, entailing or neutral. For example, for a tweet: "Prayers for Cabo: Hurricane Odile Roars Through Cabo San Lucas" and explanation: "there is a hurricane", a distilBERT model finetuned on MNLI [32], returns a vector of [0.916, 0.071, 0.013]. This means there is a 92% probability of the explanation being an entailment of the tweet, a 7% probability of the explanation being neutral to the tweet, and a 1% probability of the explanation being a contradiction to the tweet.

An embedding of size three starkly contrasts the embeddings produced by ExpBERT, which when given the tweet and explanation would return a vector of size 768. Therefore this suggested that NLI was a suitable technique to use in the new method.

3.1.3 BERT with Inference and Explanations (BERT-IE)

Having decided upon using NLI, the next step in the design stage was deciding the best way to incorporate it. It was important that the information being captured in ExpBERT was also captured in this new method, to preserve the increase seen in performance over NoExp. However, I also had to keep in mind that the way I added NLI did not add unnecessary noise. It soon became apparent that there were not many different ways to incorporate NLI, given that it required both the tweet and the explanation. Furthermore, a big hurdle of using ExpBERT was the size of the embeddings created and therefore naturally this was the most logical part to tackle. For this reason I chose to include NLI in the embeddings stage of the method.

Given the use of inference and explanations in this new method, I chose the name of BERT-IE, which stands for BERT with Inference and Explanations.

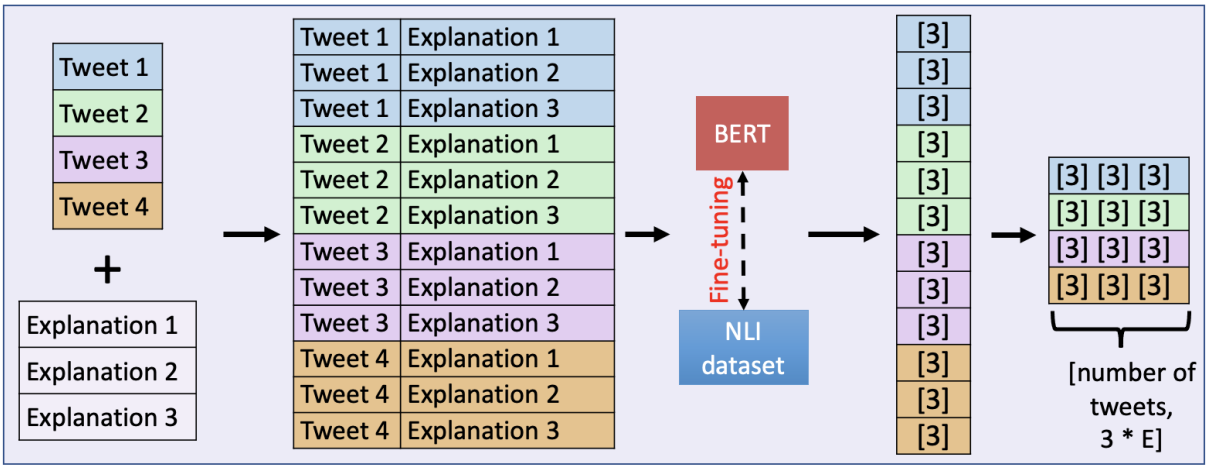


Figure 3.1: Visual representation of the data flow using BERT-IE

Figure 3.1 shows the flow of data through BERT-IE. As with ExpBERT, the first step in BERT-IE, is combining the tweets and explanations using the Cartesian product to form an expanded dataset. Whereas in ExpBERT, this expanded dataset would be passed through BERT and the [CLS] token retrieved, in BERT-IE, the expanded dataset is passed through a version of BERT fine-tuned on an NLI dataset. Using this fine-tuned model, representations of size three are created. As with ExpBERT, these representations need to be concatenated to form "the explanation representation" [55]. Equations 3.1 and 3.2, show the mathematical representation of this whole process.

$$R(x, e_j) = NLI\ BERT([CLS], x, [SEP], e_j) \tag{3.1}$$

$$z(x) = [R(x, e_1), R(x, e_2), ..., R(x, e_E)] \tag{3.2}$$

An additional step in creating the embeddings using BERT-IE, is concatenating the existing embeddings to the [CLS] token of each tweet, which was seen in Section 2.5.1. The [CLS] token is a vector of 768 and represents the information seen in the tweet. Figure 3.2 provides a visual representation of the concatenation. This was necessary in order to ensure the information stored in the tweet itself was still being passed to the model head. Without this, the model head would only be learning to classify NLI outputs, and would not have access to any information directly about the tweet.

The final size of the embedding for each tweet created using BERT-IE is $768 + (3 * E)$ where E is the number of explanations provided. For example, for a dataset with four tweets and three explanations, as seen in the diagrams, the embedding size for each tweet would be $768 + (3 * 4)$, which is 780. In contrast, if ExpBERT was used to create the embeddings, for each of the three tweets, the size would be $768 * 4$,

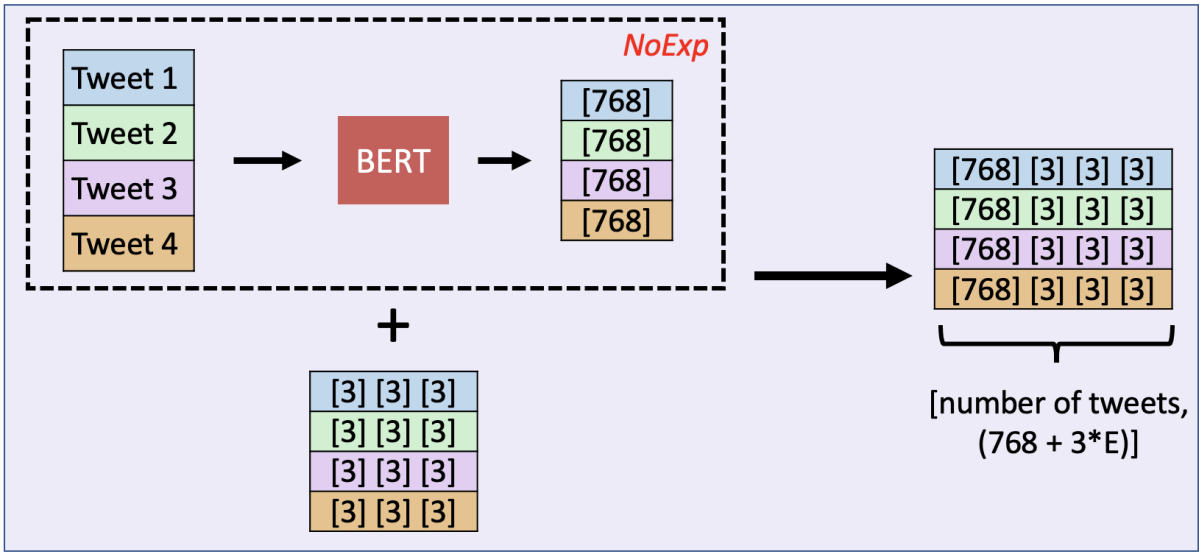


Figure 3.2: Visual representation of the embeddings being concatenated to NoExp embeddings to form new, more informative embeddings.

which is 3,072, almost four times bigger. Consequently, the use of NLI in this method meets criterion 2 of producing significantly smaller embeddings, and overcomes the hurdles of ExpBERT, thus also meeting Objective 1 of the project.

3.2 Implementation

Having designed a new method of including explanations that produced embeddings significantly smaller than those produced by ExpBERT, the next stage was implementation. The purpose of this stage was to enable me to produce results to investigate if my new method met criteria 3-5. This meant the performance of the new method, as well as computational resources and time required to run it, needed to be measurable in order to be compared to NoExp and ExpBERT. Furthermore, in line with Objective 3, the implementation needed to be clear and simple, allowing the code base to be easily configured for future work.

As explained in Section 2.7.3, Pilitsis [62] adapted ExpBERT to perform document classification and tested this on CrisisNLP [36], but had some technical errors in the implementation. Additionally, Pilitsis implemented NoExp to compare their findings to, but this implementation also included the same technical errors. As a consequence I had to modify this implementation to correct the errors and gain a new base set of results.

The correction of NoExp and ExpBERT and the implementation of BERT-IE built off the publicly available code [63] produced by Pilitsis [62].

In my implementation, the code was primarily split into three parts, as was also done by Pilitsis [62]:

1. preparing the dataset (`prepare_dataset_all.py`)
2. creating the embeddings (`prep_embeddings_all.py`)
3. classifying the tweets (`classify_tweets_all.py`)

Each file was written in Python using PyTorch [68], a framework commonly used in ML. A suitable alternative is TensorFlow [84], which provides similar features and can be used for the same ML tasks. PyTorch was chosen due to having more familiarity with it. Common Python libraries were used throughout the implementation, such as Pandas [59], NumPy [57] and scikit-learn [77], and all code formatted with Black [8] which is PEP 8 compliant.

To maintain the code base and for version control, GitHub was used. The GitHub repository for this project is publicly available here: [`https://github.com/Ambikaa1/BERT-IE`]. Using GitHub allowed me to track my progress, as well as restore code from previous iterations if needed.

The following sections outline the content of the three primary files and explain the reasoning behind implementation choices, in particular where the code was significantly changed from that of Pilitsis [62].

3.2.1 Data preprocessing

As with most ML and data science tasks, the first job was to preprocess the dataset.

As introduced in Section 1.5, the dataset has 17,117 tweets each labelled through crowdsourcing with one of nine labels. These labels were converted into numerical representations from zero to eight, so they could be understood by the model. Columns that wouldn't be used, such as tweet_id, as can be seen in Table 1.1 were then removed and some columns renamed, such as "tweet_text" to "text" for easy future reference. It was important that I deleted columns I was not going to use as the dataset was going to grow when adding in explanations and unnecessary information did not need to be duplicated or passed in to the pre-trained model. In addition, duplicate tweets were removed to prevent data leaking; this occurs when the same data is used across training, testing or validation sets.

A series of data cleaning tasks were then performed on individual tweets, as done by Pilitsis [62]. Some of these were also done by the producers of the dataset, during their investigation of classifying the tweets using "three well-known learning algorithms" [36]. These included replacing URLs with "http" and removing usernames. This was necessary as they are common features in tweets and I did not want the classifier to make associations with those features rather than the specific features that would help to classify the tweets. For example, the classifier may have started to recognise the author of the tweets and therefore when it saw a tweet from that username, it would label the tweet, regardless of the actual content in the tweet.

Additional preprocessing tasks such as splitting up hashtags, replacing emojis with textual descriptions and removing "RT" from tweets were also performed, as done by Pilitsis [62], motivated by the same reasons.

For ExpBERT and BERT-IE an additional step was needed to concatenate the explanations to the now-cleaned dataset, as described in Section 3.1.3 and Equation 2.5. 27 explanations, approximately five to 10 words long and produced by Pilitsis [62], were read in and appended to the textual descriptions (labels) for each class. This meant each class had 3 explanations and one textual description, giving a total of 36 textual values. I chose to use only 3 explanations per class based on Pilitsis' [62] findings that having more explanations per class (up to 6), did not result in a significant improvement in performance, however this did significantly increase the training time and number of embeddings produced.

For each tweet to have each explanation appended to it, forming the expanded dataset described in Section 3.1.3, first 36 copies of the tweet needed to be created. Then, each copy of the tweet had one of the explanations appended to it. This resulted in a dataset with 17,117 * 36 (616,212) datapoints.

3.2.2 Preparing the embeddings

Once I had created the expanded datasets for both ExpBERT and BERT-IE, the next stage of the process was to pass each datapoint through the pre-trained model to obtain the embeddings for both. For NoExp, the cleaned tweet dataset needed to be passed through to obtain the embeddings. As introduced in Section 2.5.1, an embedding is a vector representation of the inputs which then gets passed into the model head.

The first step in this stage was to initialise the pre-trained model and tokenizer, available from the Hugging Face model hub [33]. For ExpBERT and NoExp, the `AutoModel` class was used. This is a "generic model class", which automatically instantiates to one of the base model classes, when given the name of the pre-trained model. For example, given the pre-trained model `distilbert-base-uncased`, a `DistilBertModel` class will be instantiated. The same automatic instantiation works for the tokenizer, when the name of a pre-trained model is passed into `AutoTokenizer`. For example, when passed in `distilbert-base-uncased`, as in the example above, a tokenizer of model `DistilBertTokenizer` is created. I chose to use `AutoModel` and `AutoTokenizer` to allow me to easily change the pre-trained model being initialised. These pre-trained models used different classes, such as `DistilBertModel` and `BertModel`, and it would not have been as efficient if each time I wanted to change the pre-trained model I had to change the base model class in the code base.

For BERT-IE, `AutoModelForSequenceClassification` was needed. As with `AutoModel`, passing in the name of the pre-trained model instantiated a base class for the model. For example, for the pre-trained

model `distilbert-base-uncased`, a `DistilBertForSequenceClassification` model would be instantiated. This differs from `AutoModel` because a classification head has been appended to the previous layers. Figure 3.3, modified from Hugging Face [18], indicates this difference. Up to the pre-trained model output, both classes are the same. But, the sequence classification head in `AutoModelForSequenceClassification` takes the output of the pre-trained model, and maps it to the number of labels provided as input, using the `num_labels` parameter. In this case this is three, as the output from NLI is a vector of size three. However, for a classification task ascertaining if a sentence was positive or negative for example, the number of labels needed would be two. There are other model classes available, such as `AutoModelForQuestionAnswering` and `AutoModelForMultipleChoice`, but these were not appropriate for the task at hand. The model head was not needed for NoExp or ExpBERT as the [CLS] token needed was stored in the pre-trained model output. It is important to note here that `AutoTokenizer` is sufficient for all tasks as the process of tokenizing an input, as described in Section 2.3.2 is consistent, regardless of the desired output.

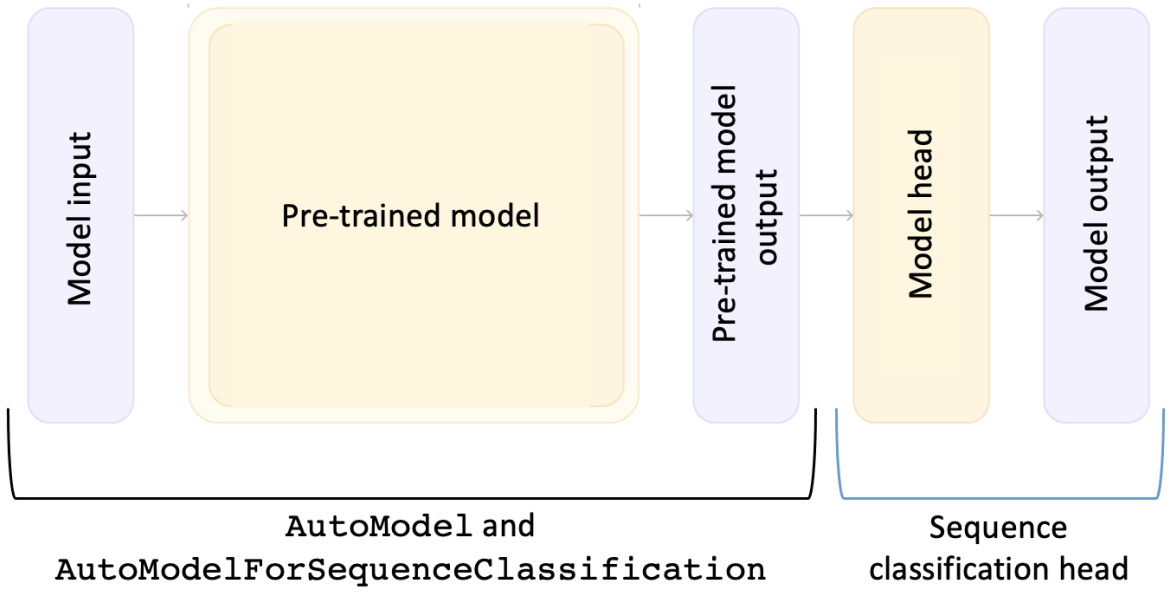


Figure 3.3: Diagram modified from Hugging Face [18] illustrating which parts of the class architecture are specifically for the `AutoModelForSequenceClassification` class.

Having initialised the model and tokenizer, the dataset was passed through both, one at a time. For BERT-IE, the `pipeline` feature available from Hugging Face was an option. This would take the input and pass it through the tokenizer and model, with no other interaction needed. For this, a task needs to be defined, which in this case would be 'sequence classification'. I chose not to use this feature, however, as I wanted to keep the method consistent with NoExp and ExpBERT, so that the results were comparable, as using the `pipeline` feature may have changed the speed at which the data was processed.

Due to the size of the data being processed (616,212 datapoints for ExpBERT and BERT-IE), it was important that memory constraints were kept in mind throughout the implementation. Passing the entire dataset into the model at once would require a large amount of memory, for both the pre-trained model and for storing the output. Storing all the embeddings in one variable would not have been possible and therefore the need to split the dataset into batches arose. In Pilitsis' [62] implementation, the expanded dataset was passed through the tokenizer altogether and the output split into 150 batches. Each batch was then passed into the model. Although this implementation worked, the process of splitting the dataset into the 150 batches was very time consuming and computationally expensive. To overcome this, I chose to use a DataLoader [67] to batch the dataset, which is standard practice when using PyTorch, due to the simplicity and convenience it provides, and more notably, the speed up. I also chose to pass the dataset through the tokenizer in these batches, rather than all at once and then batching the tokenizer output. This is because there was no benefit for the dataset to be tokenized altogether as each datapoint is independent. I created batches of size 10, as the amount of memory required to store the output of this sized batch was more aligned with the computational resources that small businesses and organisations are likely to have.

One technical error noticed in Pilitsis' [62] implementation was the transfer of data from the tokenizer to the model. As explained in Section 2.4, standard BERT tokenizers return three parts: `input_ids`, `token_type_ids` and `attention_mask`. Although the most useful part of the output is stored in `input_ids`, the other two parts are needed to help the model separate the tweet, explanation and padding. Within Pilitsis' implementation, only the `input_ids` were being passed into the model. However, passing in the `token_type_ids` is particularly important for BERT-IE, as the model needs to be able to separate the premise (tweet) and the hypothesis (explanation) for NLI. Listing 3.1 shows Pilitis' implementation, and Listing 3.2 shows my implementation with the technical error corrected.

```
tokenized_train = tokenize_noexp_function(raw_datasets['train'])
train\_ids = tokenized\_train['input\_ids']
train_ids_split = torch.split(train_ids, int(train_ids.shape[0] /
    args.split_value))
for count, train_ids in enumerate(train_ids_split):
    model_outputs = model(train_ids)
    output = model_outputs['last_hidden_state']
    embeddings = output[:,0,:]
```

Listing 3.1: Pilitsis' implementation of the creation of embeddings

```
train_dataloader = DataLoader(raw_dataset[''train''], batch_size=10)
for batch in train_dataloader:
    with torch.no_grad():
        tokenized_train = tokenize_exp_function(batch)
        model_outputs = model(**tokenized_train)
        output = model_outputs['last_hidden_state']
        embeddings = output[:,0,:]
```

Listing 3.2: Implementation of the creation of embeddings, with technical errors corrected

Once the dataset had been passed through the tokenizer and model, and the embeddings obtained, the final step was to reshape them. This step was not needed for NoExp. For each of the 17,117 tweets, the embeddings for the 36 variations of it needed to be concatenated, as shown in Equations 2.5 and 3.2. For ExpBERT this meant each tweet was now represented by an embedding of size 27,648 $(36 * 768)$, and for BERT-IE the embedding was of size 876 $(768 + (36 * 3))$.

It is important to highlight the speed up in my implementation from correcting the technical error and making use of a DataLoader [67]. Creating the embeddings for ExpBERT with an NLI distilRoBERTa pre-trained model [31] took 28 hours with Piltisis' implementation. With my implementation, this same process took just 12 hours. This is almost **60% faster**, and undoubtedly **more suitable for our use case**. Despite this substantial improvement in runtime, due to the size of the dataset being processed and the computational facilities available, it was necessary to split the expanded dataset into nine subsets. Embeddings were created for each subset and concatenated before classification and therefore this did not have an impact on the results. More detail on this process can be found in Appendix A.

3.2.3 Classifying the tweets

The final part of the implementation was classifying the tweets. This part involved taking the embeddings, a form of the dataset understood by the model, and turning this into predictions, a format understood by humans. As with the previous two parts, the implementation of this final part built off code implemented by Pilitsis [62], but made changes where technical errors had been made, or better practice could have been used.

The first step was taking the 17,117 embeddings (one for each tweet), shuffling and splitting them into a training, validation and test dataset. The ratio of split was 70:15:15 respectively, giving 11,981 training datapoints, 2,568 validation datapoints and 2,568 test datapoints. I chose to include both a validation and test dataset, as opposed to just a validation set as was done by Pilitsis [62], as it allows the final model to be evaluated on data it has not previously seen. This ensures that the model is generalisable to other data, not just data it has seen during training and testing. This was important given the range of wider domains where BERT-IE could be applied, due to it being quicker and lighter and requiring less labelled data.

Once shuffled and split, the data is passed to a `Trainer` class with the model head. As described in Section 2.5, the model head is a simple neural network that maps the embeddings to the classes. As previously described, the embeddings are of size $[17117, 768]$ for NoExp, $[17117, (768 * E)]$ for ExpBERT and $[17117, 768 + (3 * E)]$ for BERT-IE, where E is the number of explanations provided. For this dataset, the number of classes being mapped to is nine. In my implementation, the model head contains one hidden layer with 100 neurons. The number of hidden layers required is an open question within the ML industry, with each network varying largely and only worked out by trial-and-error [19]. I chose to use only one hidden layer as the task was not very complex, and previous research by Pilitsis [62] had found that increasing the number of hidden layers did not improve the result, but rather stunted performance as more noise was being added to the network.

Within the `Trainer` class, 40 iterations were performed to train the parameters of this model head. These iterations are known as epochs. I chose 40 epochs for my experiments as it was the number of epochs used to train BERT [15]. However, I made it easy to alter this value from the command line, if I found that the performance had not converged within 40 epochs. This differed from Pilitsis' [62] implementation, which used 400 epochs. In each epoch, the training data is passed into the input layer of the model head, of size 768, through the one hidden layer, of size 100, and to the output layer, of size nine. To facilitate this, I again used a `DataLoader` [67], which split the training data into batches to be passed into the model head.

For each batch in the training data, the loss was calculated and propagated back through the network to update the model head parameters. I used Cross Entropy Loss (CEL) in this implementation as it is the most suitable loss function for classification. An alternative loss function is Mean Squared Error, but that is more suitable for regression tasks. The equation for multi-class CEL for a single datapoint is given in Equation 3.3, where C is the number of possible classes, in this case nine, y_i is the true label for the class i and $\hat{y}_i$ is the predicted label for the class i. This means there is a high loss when the model head returns low probabilities for the true class.

$$CEL = -\sum_{i=1}^{C} y_i \cdot \log \hat{y}_i \tag{3.3}$$

Once the loss was calculated, an optimiser updated the model head parameters accordingly. I used AdamW instead of using Stochastic Gradient Descent because the latter can be slower to converge than the former. The amount by which the optimiser updates the parameters is influenced by the learning rate and weight decay. These are two hyperparameters that can be tuned as further discussed in Section 3.3.2.

Following the training, the standard practice for evaluating a neural network is followed. Firstly, the performance can be tested on the validation dataset. The frequency of this can be altered although for each experiment I tested every epoch. The validation dataset is split into batches again using a `DataLoader`, and the loss calculated using CEL. As the validation dataset is not used for training the model, the optimiser is not needed. Rather, the purpose of the validation dataset is to provide a way of evaluating the performance of the model as it learns, which can be useful for hyperparameter tuning. Finally, after all 40 iterations, the tuned model is evaluated on the test dataset. The results from this evaluation, such as F1 score and loss, are used for comparison in the experiments.

As outlined in Section 2.7.3, due to technical errors in Pilitsis' [62] work, their results were unsuitable. One significant technical error was the application of `Softmax`. `Softmax` is a function used to normalise the outputs of a model, so that for each datapoint the probabilities of each class (logits) add up to one. This allows the class with the maximum probability to be taken as the predicted value. In the PyTorch implementation of CEL [66], `Softmax` is applied. Therefore, I did not apply it again. However, in Pilitisis' implementation this was not the case, meaning `Softmax` was wrongly applied to the logits twice.

Table 3.1 shows the effect of applying `Softmax` twice. Although the class with the maximum probability is still class three after applying `Softmax` twice, the range of probabilities is considerably reduced (0.0609 to 0.0058). Therefore the calculated loss for each class will be very similar, meaning the model is not able to learn as much in each iteration and as a result is likely to learn slower. Figure 3.4 illustrates the impact applying `Softmax` twice has on the performance. It can be seen there is a difference in score of 12.7 (measured using F1 score further explained in Section 3.3.3). For this reason, it was unsuitable to

	0	1	2	3	4	5	6	7	8
logits	0.2000	0.0400	0.1000	0.5000	0.1500	0.2000	0.0100	0.0300	0.0500
softmax applied once	0.1164	0.0992	0.1053	0.1572	0.1107	0.1164	0.0963	0.0982	0.1002
softmax applied twice	0.1117	0.1098	0.1105	0.1163	0.1111	0.1117	0.1095	0.1097	0.1099

Table 3.1: Table showing the effect of applying `Softmax` once and then twice to logits obtained from a model.

use results produced by Pilitsis [62] as baseline results, and therefore experiments needed to be run for both NoExp and ExpBERT to produce new baselines.

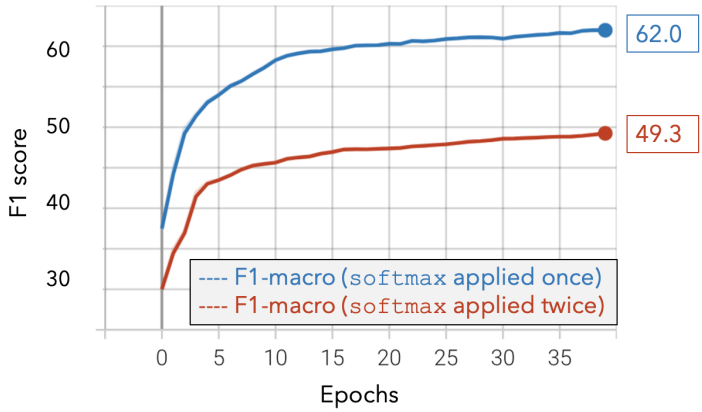


Figure 3.4: Performance of a standard NoExp classifier with softmax applied once and twice.

Seed	F1 score
2	62.3
10	62.1
37	61.9
42	61.4
57	62.1

Table 3.2: Table showing the change in performance of the model due to the use of different seeds.

An important implementation choice to highlight is the use of a seed. By default, shuffling in Python produces different results each time, as would be intuitive from a random shuffle in the real-world. However, in order to compare experiments, it was necessary for the same shuffle to be performed each time to remove the possible effect of any extenuating factors. For example, there could be a shuffle that by chance meant all the tweets in the test dataset were from a flooding disaster, and none of these had appeared in the training or validation dataset. This could result in poor test performance, due to key words being seen in the tweets that wouldn't have been seen during training. In contrast, a well balanced dataset of tweets from different types of natural disaster, is likely to perform better. To obtain the same shuffle for each experiment, a seed can be used. This initialises the random number generator in Python. Table 3.2 contains the F1 score, the chosen performance metric (Section 3.3.3), for five different seeds. Here the impact on performance from using different seeds can be seen, thus highlighting the need for one seed to be used for all experiments. Furthermore, the use of a seed promotes the reproducibility of the results. For all experiments ran in this project the seed has been set to 37 which was chosen at random.

3.3 Experiments

Having implemented NoExp, ExpBERT and BERT-IE, the next step in the project was to conduct experiments. These were needed to understand how the way in which explanations were included affected performance, memory usage and training time.

During this stage, as well as comparing the methods against each other, I wanted to identify the best pre-trained model for each method. This is because some pre-trained models are smaller and lighter than others, as introduced in Section 2.4.1, and so I wanted to identify if there was a tradeoff to be made between the pre-trained model used, the time taken to run the classification and the resulting performance.

Each pre-trained model required slightly different values for the learning rate, which affects how quickly the model learns, as well as the weight decay, which is used to prevent overfitting. These values and

others are known as hyperparameters and changing these to get the best performance is known as hyperparameter tuning.

As is the case with most ML models, the computational resources required meant the experiments were not suitable to be run on a local computer. Therefore, all experiments were run on a supercomputer with an Nvidia P100 GPU, which was given at most 48GB RAM per experiment. This GPU can be ran on Google Cloud Platform [24] for $0.43 or £0.34 per hour.

The following sections outline how I went about efficiently running the experiments, hyperparameter tuning the models and evaluating the results.

3.3.1 Pre-trained models

The way in which pre-trained models can be used from the Hugging Face Model Hub [33] is consistent. As outlined in Section 3.2.2, the `AutoModel` class was used for NoExp and ExpBERT, and the `AutoModelForSequenceClassification` class for BERT-IE. Both classes only required the name of the pre-trained model, passed in as a string. As I frequently changed the pre-trained model used to create the embeddings, I chose to use argument parsing as that was a more efficient way of changing the string, than going into the code base for each experiment and making the change there. To do this, I made use of the `argparse` module from Python, which allows a user to pass in strings on the command line. For example to run the `prepare_embeddings_all.py` file with the `bert-base-uncased` model, the command line argument was:

```
python prepare_embeddings_all.py --pretrained bert-base-uncased
```

Listing 3.3 shows the `pretrained` argument being added to the `argparse` module, and then the value being passed into `AutoModel` and `AutoTokenizer`.

```
import argparse
parser.add_argument("--pretrained", type=str, required=True,
    help="the pre-trained model to run")
model = AutoModel.from_pretrained(args.pretrained)
tokenizer = AutoTokenizer.from_pretrained(args.pretrained)
```

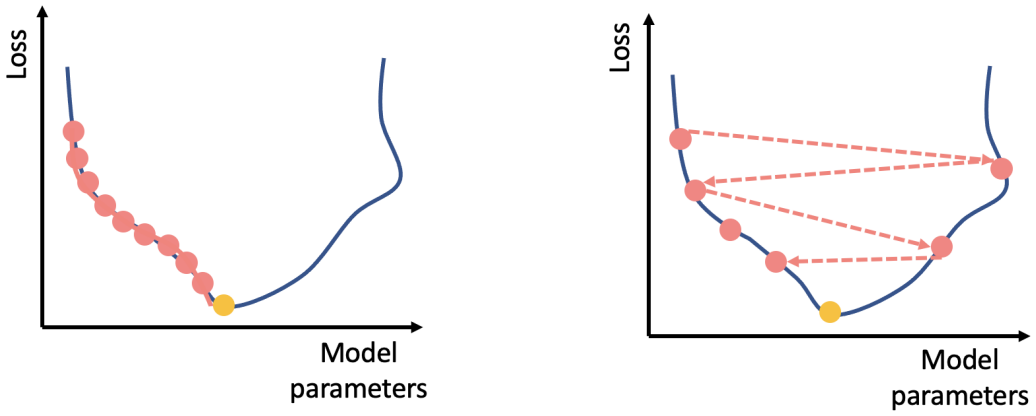
Listing 3.3: Parsing the name of the pre-trained model using argparse

3.3.2 Hyperparameter tuning

As previously mentioned, the required hyperparameter values for each pre-trained model can differ. This is because the behaviour of each model can be different. Hyperparameter tuning is an important aspect of training a model and as found by the authors of RoBERTa [48], it can have a "significant impact on the final results". As was done when changing the pre-trained model used, I used argument parsing to modify the hyperparameter values. This again meant I did not have to go into the code base to change the values and could easily run multiple experiments efficiently.

The first hyperparameter I tuned was the learning rate. This controls by how much the model parameters are updated. If the learning rate is too low, the model parameters are updated very minimally and a large number of epochs are needed for the model to reach it's optimal performance (Figure 3.5a). This can result in high computation times. Conversely, if the learning rate is too high, the model parameters are updated by a large amount which can lead to divergent behaviours and the model not reaching optimal performance (Figure 3.5b). A model with a good learning rate will converge near optimal performance after a small number of epochs.

The second hyperparameter I tuned for each experiment was the inclusion of weights in the loss function. As mentioned in Section 3.2.3, CEL was used. By default, the PyTorch implementation [66] gives each class an equal weighting and therefore it cannot take into account the class imbalance of a dataset. Thus, CEL did not seem suitable, given the class imbalance present in the CrisisNLP dataset [36], as seen in Section 1.5. Instead, the distribution of class labels were passed in to the `CrossEntropyLoss` function to form the Weighted Cross Entropy Loss (WCEL) function. Listing 3.4 shows the distribution of the labels being passed in as the weights, and then made inversely proportional to the class size and scaled. Using this helps the smaller classes, such as classes two and three, to be classified.



(a) A low learning rate requires numerous iterations (b) A high learning rate leads to divergent behaviours

Figure 3.5: Diagrams showing the impact of the learning rate on reaching optimal performance which is indicated in yellow.

The final hyperparameter I tuned was the weight decay. This is a regularisation technique used to prevent overfitting, and therefore it does not have a substantial impact on performance. It works by reducing the weights over the iterations, so they do not become too large or pick up too much noise as they get updated.

```
#Use distribution of class labels for weights
weights = torch.tensor([13.0864, 2.1791, 3.1197, 7.9103, 14.1146,
    5.8246, 11.0066, 29.8417, 12.917], dtype=torch.float32)
#Convert percentages to decimal (e.g. 13.08\% to 0.13)
weights = weights / weights.sum()
#Make weights inversely proportional to class size
weights = 1.0/weights
#Scale weights so they sum to 1
weights = weights / weights.sum()
return weights

criterion = nn.CrossEntropyLoss(weight=weights)
```

Listing 3.4: Passing in weights to the CrossEntropyLoss function

Another hyperparameter that could have been changed was the batch size. As described in the previous section, Section 3.2.3, the data is passed into the model head in batches. As the loss is calculated for each batch and the weights updated accordingly, a large batch size generally leads to the model not being able to generalise well, as found by Keskar et al. [45]. Having a very small batch size, however, can cause a large increase in the computational time required as the model needs to calculate the loss and refine the model parameters more frequently. Therefore, I chose to use a batch size of eight, a common value used in ML.

An alternative solution to changing the hyperparameter values manually, via the command line, was to use a module to automate this, such as `GridSearchCV` [76] from scikit-learn [77]. In `GridSearchCV` potential hyperparameter values are passed in and each combination of hyperparameters is ran. For example, the potential learning rate values may be: 5e-4, 5e-5 and 5e-6 and the potential weight decay values may be: 1e-1, 1e-2 and 1e-3. `GridSearchCV` would try each combination of these values and return the performance of the model. I chose not to use this module, as I wanted to understand the effect that each hyperparameter had on the model, not just in terms of performance, but also in terms of training time. Additionally, manually doing the hyperparameter tuning meant that not all combinations needed to be tested if they were not having an impact.

3.3.3 Performance metrics

To be able to assess and compare and contrast the different methods of including explanations, as per Objectives 4 and 5 of this project and criteria 4 and 5 for the design, it was important that a robust performance metric was used. For this investigation I chose to use the F1 score. This is a common metric used in the existing literature and was used to measure performance by Murty et al. [55], Hancock et al. [27] and Srivastava et al. [79], as well as in other research that used the CrisisNLP dataset [72], [2].

The calculation of the F1 score relies on two other metrics: precision and recall. Both of these metrics rely on the understanding of True Positives (TP), True Negatives (TN), False Positives (FP) and False Negatives (FN). Figure 3.6 shows a visual representation of these definitions.

		Predicted	
		Class A	Not Class A
Expected	Class A	True Positive	False Negative
	Not Class A	False Positive	True Negative

Figure 3.6: Visual representation of the definition of True Positive, True Negative, False Positive and False Negative.

A TP (top left) is where the classifier has predicted class A, and the expected label is also class A, whereas an FP (bottom left) is when the classifier has again predicted class A, but the expected label is not class A, rather one of the other classes. Similarly, a TN (bottom right) is the case where the classifier predicts a class that is not class A, and the expected label is also not class A. Whereas an FN (top right) is when the classifier predicts a class that is not class A, but the expected label is class A. Both FP and FN, highlighted in red in Figure 3.6, can have a large impact. For example, if published tweets are used to allocate emergency resources and a tweet from the label "injured or dead people" is classified into the "caution and advice" category, allocation of emergency resources to the area where the tweet was published will be missed which can have fatal consequences. Therefore, it is important that a metric that captures this information is used, to enable the performance of the classifier on real-world applications to be assessed.

Precision is used to analyse all the *predicted* positive cases, and measure how many were TP. Recall, on the other hand, is used to analyse the *expected* positive cases, and understand how many were falsely predicted as negative, FN. The F1 score is defined as the "harmonic mean of precision and recall" [75]. The classifier can only obtain a high F1 score when both precision and recall are high. Equations 3.4, 3.5 and 3.6 show the equations for precision, recall and F1 score respectively. In this project, all values have been scaled from 0-1 to 0-100 to allow for easy comparison of results, as done in related works [48], [55].

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive} \tag{3.4}$$

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative} \tag{3.5}$$

$$F1score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{3.6}$$

In this implementation the `f1_score` module [75] from `scikit-learn` [77] was used as it provides a simple mechanism for calculation, only needing to pass in the true labels and the predicted labels. Within the module, there is an option (parameter) to set how the average is calculated, and is "required for multiclass/multilabel targets" as is the case in this project. One way of calculating the average is using a 'weighted' method which takes into account the label imbalance. For example, given a class that only represents 13% of the dataset, as is the case with the "injured or dead people" class and produces an F1 score of 60.4 and another class represents the other 87%, and produces an F1 score of 64.2, the average would be calculated as follows:

$$\begin{aligned}(60.4 * 0.13) + (64.2 * 0.87)) &= 7.852 + 55.854 \\ &= 63.706 \\ &= 63.7\end{aligned}$$

More generally, this is:

$$\sum_{i=1}^{N}(F1score[i] * percentage\_of\_dataset[i]) \tag{3.7}$$

where N is the number of classes in the dataset.

This method of averaging is suitable where the class imbalance should be reflected in the performance. However, in this case an average is needed that gives each class an equal weighting, regardless of how big it is. This is important as in an ideal situation we would have an equal weighting of each class in the dataset. Therefore, setting the average parameter to 'macro' is more suitable. Following the same example above, with a class representing 13% of the dataset and giving a result of 60.4 and the remaining 87% giving a result of 64.2, the F1-macro average would be calculated as follows:

$$((60.4 + 64.2)/2) = 62.3$$

This gives a different result to using the 'weighted' average as it is giving each of the imbalanced classes more of a chance to be represented. Furthermore, to ensure that the F1 score of each class was fully taken into account, the individual scores for each class were also output for analysis purposes. This is done by setting the average parameter to 'None'.

An alternative performance metric is accuracy. This simply calculates the number of predictions that were correct, but does not take into account cases of FP and FN. This means that it is difficult to understand where the classifier is going wrong. Additionally, although a classifier may have a high accuracy, the FN produced can have a big impact, as discussed above, but may get overlooked. Therefore, although accuracy gives a good overall picture of the classifier performance, it does not provide the rigorous detail needed.

3.3.4 Visualising results

To obtain a visual representation of the results, in particular during the hyperparameter tuning, three graphs were produced: accuracy, F1 score and loss. These were essential as it allowed easy comparison of results, in contrast to trying to analyse the results just by looking at the numbers.

Although it was not the main performance metric, I chose to produce an accuracy graph for each experiment as it was a simple way to compare overall performance of a model and ascertain if the F1 score being produced was likely to be correct. Similarly, although loss was also not the main performance metric it provided a good understanding of the effect of the hyperparameters, in particular the learning rate, as can be seen further on in Figures 4.1 and 4.2, in Section 4.1.

These graphs were produced using TensorBoard [83], which uses the values written to a separate file as the training progresses. Being able to see the results during the experiment, rather than having to wait till the end, was particularly helpful as experiments producing erroneous or poor results could be stopped mid-way, thus reducing the computational resources consumed.

Finally, a confusion matrix, of size nine by nine, was produced to visualise which classes were often predicted correctly (TP, TN), and which classes were often misclassified (FP, FN). For a class with a low TP value, it was simple to see what the datapoints from that class were often predicted as. This provides a more detailed breakdown of the precision and recall. For each row (predicted class) in the confusion matrix the values were normalised meaning the total values in the row added up to one. This was done as using the raw numbers can make it hard to compare the classes due to the different class sizes. I chose to use a confusion matrix as it was important that I could identify poorly performing classes, and assess if the explanations provided using ExpBERT and BERT-IE improved their performance. As an extension to this project, using this data, more explanations could be provided to the poorly performing classes, with an emphasis on distinguishing between commonly confused classes.

Chapter 4

Results & Critical Evaluation

This section outlines the results obtained for each of the three methods: NoExp, ExpBERT and BERT-IE, and discusses the benefits and challenges of each method, as well as why certain results may have been produced. Table 4.1 provides an overview of the experiments that I have run, as well as the hyperparameters that I changed and the metrics I used for measuring performance. It is important to highlight that the results for ExpBERT and BERT-IE are those obtained using the corrected code, rather than the results obtained by Pilitsis [62]. A detailed comparison of the methods is then performed. This meets Objectives 4 and 5 of this project and criteria 3, 4 and 5 for the design.

Overview of the experiments run

Method	Pre-trained model	Hyperparameters tuned	Performance metrics
NoExp	`bert-base-uncased`	LR, CEL vs WCEL, WD	Precision, Recall, F1 score
	`bert-base-uncased-MNLI`	LR, CEL vs WCEL, WD	Precision, Recall, F1 score
ExpBERT	`bert-base-uncased-MNLI`	LR, CEL vs WCEL, WD	Precision, Recall, F1 score
BERT-IE	`bert-base-uncased-MNLI`	LR, CEL vs WCEL, WD	Precision, Recall, F1 score
	`distilbert-base-uncased-finetuned-mnli`	LR, CEL vs WCEL, WD	Precision, Recall, F1 score
	`nli-distilroberta-base`	LR, CEL vs WCEL, WD	Precision, Recall, F1 score

Table 4.1: Table providing an overview of the experiments ran, the hyperparameters tuned in each and the metrics used for measuring performance.

4.1 NoExp

To ascertain if using BERT-IE gave better performance over using NoExp, as was outlined in criteria 4, a base set of results were needed for NoExp. As discussed in Section 2.7.3, the results obtained by Pilitsis [62] could not be used as baseline results due to the technical errors.

I hypothesised that BERT-IE would outperform NoExp in terms of performance, but would require a slight increase in computational resources and classification time due to the slight increase in the size of the embeddings (768 to 876).

4.1.1 Pre-trained model: `bert-base-uncased`

For the first batch of experiments, I ran NoExp with the `bert-base-uncased` pre-trained model. This was the original BERT model described in Section 2.5.1. To create the embeddings for this model, it took 40 minutes and the file size created was 51MB. To then classify these embeddings, as described in Section 3.2.3 I calculated the loss with the CEL function and used the AdamW optimiser to update the model parameters. I first ran NoExp with the default PyTorch values for the learning rate hyperparameter (1e-3) and weight decay (1e-2) hyperparameter.

Figure 4.1 shows the loss of the classifier with this configuration, running over 40 epochs, which took five minutes to run. In the graph, it can be seen that the model starts to learn (indicated by the pink line) between epochs zero and seven, but converges too quickly. The model then starts to overfit on the

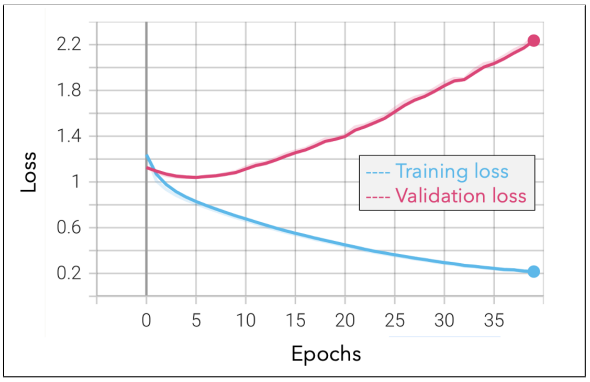


Figure 4.1: Loss curve for NoExp with a learning rate of 1e-3, weight decay of 1e-2 and CEL.

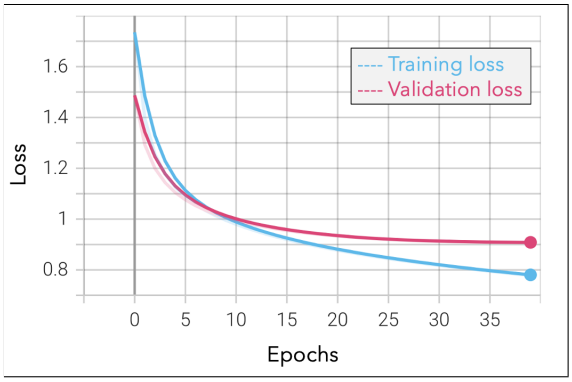


Figure 4.2: Loss curve for NoExp with a learning rate **changed to 5e-5** with an unchanged weight decay of 1e-2 and CEL.

training data, as can be seen by the blue line tending towards zero, which results in poor performance on the validation dataset, causing the loss to increase exponentially. This indicates that the learning rate is too high.

A learning rate of 5e-4, half of the previous learning rate, also produces similar results to those seen in Figure 4.1. Therefore, I chose to test a learning rate of 5e-5, 10 times smaller than 5e-4, and 20 times smaller than 1e-3. Figure 4.2 shows the loss curve for this learning rate, with all other hyperparameters unchanged. Here it can be seen that both the training and validation loss decrease over the 40 epochs, with the validation loss converging after 30 epochs, at 0.9.

Having first decreased the model loss, as high loss can hinder performance, my focus shifted to looking at the performance of the model. Figure 4.3 shows how the F1-score increases over the 40 epochs. As the model learns, the F1-score increases sharply, which corresponds with the loss seen in Figure 4.2 decreasing sharply between epochs zero and 10. After 20 epochs, this increase in performance slows down and the model starts to converge, as seen after epoch 27. This gives an overall F1-score of 61 on the validation dataset. On the test set, the F1 score is 63.5, however this is reached by some high performing classes and some low performing classes, as seen in Table 4.2. The range in performance per class is a significant 44.9 points. This suggests that some of the larger classes dominate the F1 score and the smaller classes aren't able to be classified well. For example, classes one, two, three and five, which each form less than 10% of the dataset, all perform under the average. This highlights the need for a loss function that takes into account the percentage each class forms of the dataset, namely WCEL as introduced in Section 3.3.2.

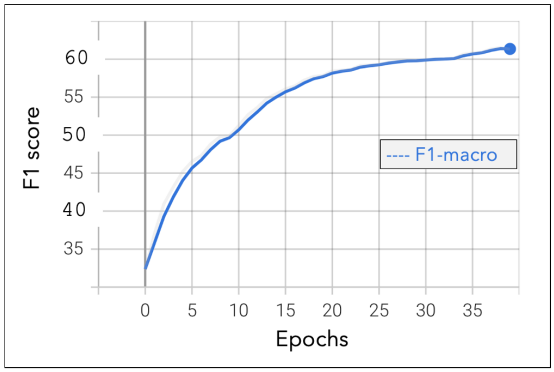


Figure 4.3: F1-macro curve for NoExp with a tuned learning rate of 5e-5, weight decay of 1e-2 and CEL.

The fourth column ('F1 score WCEL') in Table 4.2 shows the per class breakdown of the F1 score when using WCEL. Decreases in performance are coloured red and increases in performance over CEL are coloured green. It is surprising to see that the macro average has decreased to 61.9 from 63.5. Moreover, the range in performance has widened marginally from 44.9 points to 47.1 points, with five classes now performing under average. Notably, the largest class, class 7, now gives a result below the average, suggesting that the inclusion of weights is not beneficial to the model. This is a surprising result as I was expecting the performance to remain the same or increase when using WCEL, due to accounting for the class imbalance in the dataset. However, this suggests that the need for human input is required, potentially in the form of explanations. It is also important to highlight here, that due to the use of a seed, discussed in Section 3.2.3, it is possible to compare the results of CEL and WCEL without having to take into account the possibility of a poor shuffle for WCEL.

Class No.	% of dataset	F1 score (CEL)	F1 score (WCEL)
0	13.1	83.9	82.6
1	2.2	47.9	35.5
2	3.1	51.1	52.3
3	7.9	56.6	54.8
4	14.1	76.3	75.5
5	5.8	39.0	49.5
6	11.0	76.1	74.7
7	29.8	69.1	59.9
8	12.9	71.2	72.6
Macro average		**63.5**	**61.9**

Table 4.2: Table showing per-class breakdown of % of dataset and F1 score with CEL and WCEL for No-Exp.

Class No.	% of dataset	F1 score (CEL)	F1 score (WCEL)
0	13.1	87.2	87.5
1	2.2	35.4	35.2
2	3.1	56.9	55.6
3	7.9	62.2	57.3
4	14.1	75.1	74.8
5	5.8	41.7	42.0
6	11.0	77.1	75.6
7	29.8	68.3	55.6
8	12.9	68.2	68.6
Macro average		**63.6**	**61.4**

Table 4.3: Table showing per-class breakdown of % of dataset and F1 score with CEL and WCEL for the NLI pre-trained model with NoExp

The final hyperparameter to tune was the weight decay. Changing this value from the default of 1e-2 to 1e-1, a factor of 10, made negligible difference in the results. This was expected as weight decay is often more useful in cases where overfitting occurs, however this was not seen in Figures 4.2 and 4.3.

4.1.2 Pre-trained model: `bert-base-uncased-MNLI`

Having obtained results using the `bert-base-uncased` pre-trained model, I obtained results for NoExp when using a pre-trained model fine-tuned on NLI, as described in Section 2.4.1. The pre-trained model used was `textattack/bert-base-uncased-MNLI`. This was necessary as an NLI model is required for BERT-IE in order to comprehend the relationship between the tweet and the explanation, and therefore a baseline result was needed with this pre-trained model.

The process of creating the embeddings was the same as with `bert-base-uncased`, and also took 40 minutes, with the embeddings being stored in a file of size 51MB. Figure 4.4 shows the loss curve and Figure 4.3 shows the F1 curve for the classification of the tweets with a learning rate of 5e-5, weight decay of 1e-2 and CEL; the same configuration as used for the `bert-base-uncased` pre-trained model.

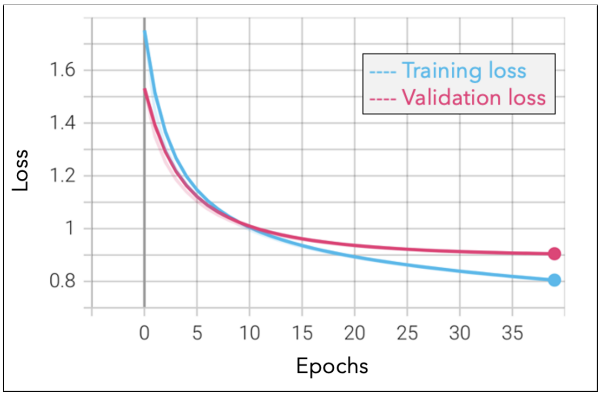


Figure 4.4: Loss curve for NoExp with an **NLI** pre-trained model, learning rate of 5e-5, weight decay of 1e-2 and CEL.

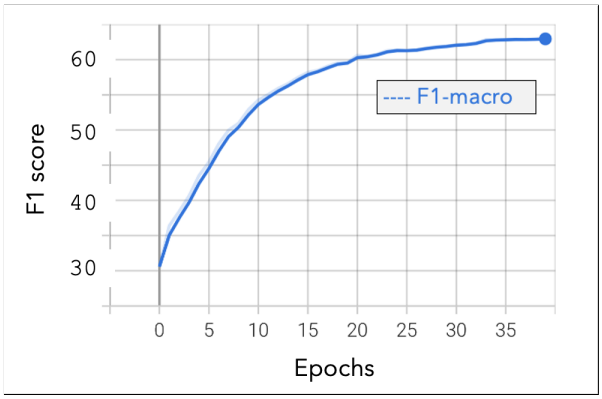


Figure 4.5: F1-macro curve for NoExp with an NLI pre-trained model with a learning rate of 5e-5, weight decay of 1e-2 and CEL.

As was seen in Figure 4.2, Figure 4.4 shows the validation loss rapidly decreasing between epochs zero and 10 as the model learns, before slowly converging to a loss of 0.9. Similarly, the F1 score increases by the same trend for both `bert-base-uncased` and `textattack/bert-base-uncased-MNLI` which reaches a score of 63.6, a marginal 0.1 points higher than `bert-base-uncased`. In addition, a similar F1 score of 63.6 is produced with CEL and a decrease in score to 61.4 with WCEL, as seen in Table 4.3. WCEL again does not improve performance with five out of nine classes now performing under average.

4.1.3 Summary

These results suggest that using different pre-trained models does not have much of an impact on performance when using the NoExp method, however, that may not be the case when explanations are included.

As I wanted to use these results as a baseline to compare my results from ExpBERT and BERT-IE to, it was important that I researched the performance obtained by others using the same dataset to ensure my baseline results were reliable. As expected, my performance of NoExp was greater than obtained by Pilitsis [62]. Furthermore, ALRashdi and O'Keefe [2] found that using a convolutional neural network for classifying CrisisNLP [36] gave an F1 score of 61.4, and when using a variant of a recurrent neural network (Bi-LSTM), obtained an F1 score of 62.0. This gave me confidence that my baseline results (63.6) were reliable and could be used for comparison.

4.2 ExpBERT

The aim of the next set of experiments was to explore the impact on performance of adding explanations, in particular on the class imbalance, using the ExpBERT method, and to obtain a baseline set of results that BERT-IE could be compared to, in line with Objectives 2 and 5 and criteria 3 and 5. My hypothesis was that ExpBERT would give an increase in performance over NoExp, however, would be a lot more computationally expensive, in terms of runtime and memory usage.

4.2.1 Pre-trained model: `bert-base-uncased-MNLI`

The first step for this batch of experiments was to create the expanded dataset as explained in Section 3.2.1. This meant combining the 17,117 tweets with the 36 explanations, to form an expanded dataset of size 616,212. As the expanded dataset was not affected by the pre-trained model or hyperparameters, this only needed to be done once.

Having created this expanded dataset, the `textattack/bert-base-uncased-MNLI` pre-trained model was used to embed each datapoint and the [CLS] token then retrieved for each row. As hypothesised, this took considerably longer than NoExp, as the dataset was now 36 times greater. The total runtime was 22.75 hours (34 times longer than NoExp), and once created, the embeddings were stored in a file of size 1.8GB. Appendix A details how the expanded dataset was split into subsets to accommodate this large runtime.

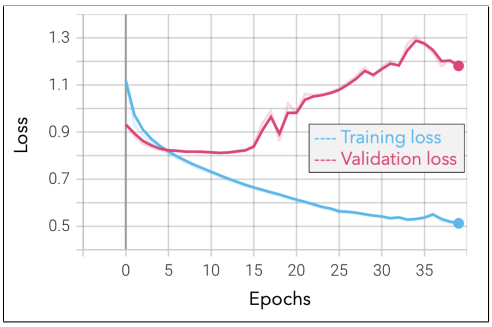


Figure 4.6: Loss curve for ExpBERT with a learning rate of 1e-3, weight decay of 1e-2 and CEL.

Next I classified the tweets using the default hyperparameters for the learning rate, weight decay and loss function, as was done with NoExp. Figure 4.6 shows the loss over 40 epochs, which took 50 minutes to run. It can be seen that from epochs zero to four the model does learn, however results in only a marginal decrease in validation loss. The model then appears to converge from epochs five to 12, before signs of overfitting can be seen from epoch 13 and onwards, as the validation loss increases rapidly and the training loss decreases rapidly. This suggests that the model has found a local minima by epoch five, causing the model to converge. However, because of the high learning rate, the model then jumps to a different minima, as was visualised in Figure 3.5 and then starts to overfit. Therefore, it was clear a lower learning rate was required.

I chose to try a learning rate of 5e-5 next as that was the optimal learning rate found for NoExp. I kept the weight decay at 1e-2, and first tried with the unweighted loss function. Again, Figure 4.7 shows the model learning, and by epoch seven it has converged. From epoch 15, the model then starts to overfit on the training data, causing the validation loss to go up. This suggests that the model does not need more than 15 epochs, and therefore is bound to overfit when run for 40 epochs. This possibility is supported by the F1 curve, shown in Figure 4.8. Here it can be seen that the model's performance peaks by epoch 15, but starts to decrease after. This is a plausible result, however was surprising as I expected the model performance to remain constant after convergence, as was seen in Figure 4.2 for NoExp.

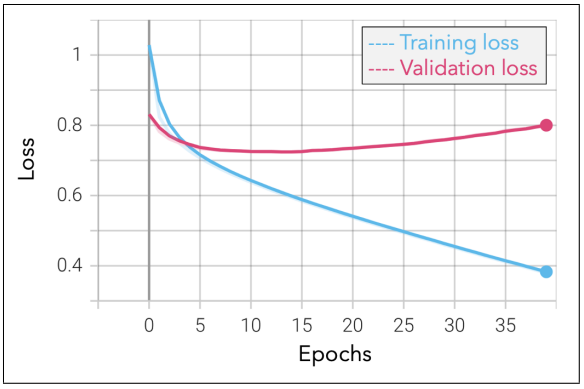


Figure 4.7: Loss curve for ExpBERT with a **changed learning rate of 5e-5**, weight decay of 1e-2 and CEL.

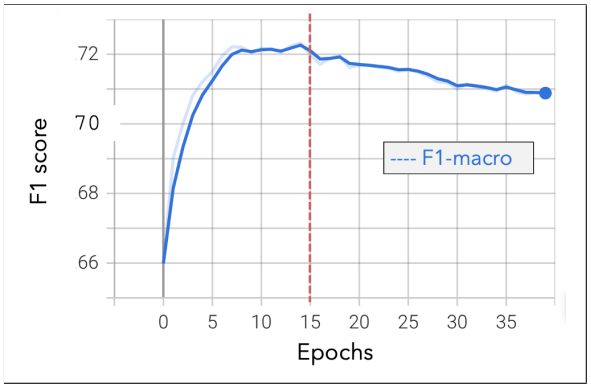


Figure 4.8: F1-macro curve for ExpBERT with a learning rate of 5e-5, weight decay of 1e-2 and CEL.

As a result, I experimented with a learning rate of 5e-6 to explore if a similar trend was seen, or if once the model had converged the performance and loss remained stable. As expected, the model learnt quickly during epochs zero to seven, as indicated by the steep loss curves and F1 curve, in Figure 4.9 and 4.10 respectively. After this the model learnt slowly, gently increasing in performance.

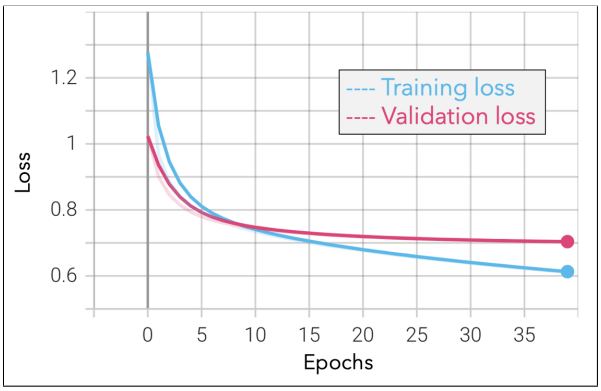


Figure 4.9: Loss curve for ExpBERT with a **changed learning rate of 5e-6**, weight decay of 1e-2 and CEL.

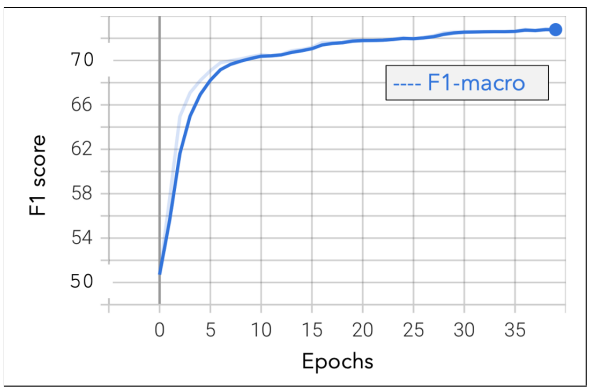


Figure 4.10: F1-macro curve for ExpBERT with a **changed learning rate of 5e-6**, weight decay of 1e-2 and CEL.

It was interesting to see that with a learning rate of 5e-5 and a learning rate of 5e-6, the peak performance reached on the validation dataset was 72.3 and 72.8 respectively, a difference of only of 0.5 points. However, 5e-5 only required 15 epochs, which took 20 minutes to run, in contrast to a learning rate of 5e-6 which required 40 epochs, and 50 minutes to run. Table 4.4 shows the per-class breakdown in performance on the test dataset with these two learning rates. It can be seen that a learning rate of 5e-6 produces better results (indicated in green) for almost all classes, with the exception of class one and four (indicated in red). However, running the classifier for 15 epochs only takes 20 minutes, 2.5 times less time than running for 40 epochs which takes 50 minutes. This suggests that although there is a slight loss in performance (1.1 points on the test dataset), 5e-5 is the more suitable learning rate, when also taking into account the main focus of this project of making

Class No.	**% of dataset**	**F1 score (5e-5, 15 epochs)**	**F1 score (5e-6, 40 epochs)**
0	13.1	90.8	91.9
1	2.2	55.7	51.7
2	3.1	66.7	70.4
3	7.9	69.0	71.5
4	14.1	81.2	80.7
5	5.8	48.7	49.6
6	11.0	77.9	80.5
7	29.8	70.4	72.3
8	12.9	75.0	76.4
Macro average		**70.6**	**71.7**

Table 4.4: Table showing per-class breakdown of % of dataset and F1 score with a learning rate of 5e-5, with 15 epochs and a learning rate of 5e-6, with 40 epochs.

ML less computationally expensive.

Finally, having determined the optimal learning rate, I wanted to understand whether including weights in the loss function improved performance. This did not result in an increase in NoExp, however this could be different with the inclusion of explanations. Table 4.5 shows the results for running ExpBERT without weights (CEL) and with weights (WCEL). As was seen in NoExp, the inclusion of weights had a slight negative impact on performance, with six out of nine classes performing worse with WCEL. In both models, classes one, two, three and five performed below average. This was expected as these classes only make up a minority of the dataset and therefore can be harder to classify. It is surprising to see however, class 7 perform below average for both models, given that it is the largest class, and by including weights, the performance for this class reduces by 6.7 points. To understand this effect further, I analysed the precision and recall values for all classes. The two right-most columns of Table 4.5 show how these changed with the introduction of weights. The first notable class is class two, which has a decrease in precision of 10.5 following the inclusion of weights, but an increase in recall of 11.6. Because all classes are being made inversely likely, the classifier may be predicting more datapoints as class two. This means that the recall goes up as more of the expected class two datapoints are correctly labelled, but many non class two datapoints are also being labelled as class two, resulting in a decrease in precision. A similar pattern is seen for class five, also one of the smallest classes in the dataset. The opposite however, is seen for class seven, the largest class. As the classifier now thinks class seven is less likely to be the true value, it predicts less datapoints as class seven, including ones that are actually class seven. This results in a decrease in recall of 13.3. Therefore, this suggests WCEL is not a suitable way of taking into account the class imbalance.

Class No.	**% of dataset**	**F1 score (CEL)**	**F1 score (WCEL)**	**Change in F1 score**	**Change in precision**	**Change in recall**
0	13.1	90.8	89.8	-1	-0.6	-1.4
1	2.2	55.7	51.6	-4.1	-10.5	+11.6
2	3.1	66.7	68.0	+1.3	+3.2	0
3	7.9	69.0	66.8	-2.2	-5.2	+2.8
4	14.1	81.2	79.8	-1.4	-0.2	-2.8
5	5.8	48.7	52.4	+3.7	-9.4	+11.5
6	11.0	77.9	78.9	+1	+2.5	-0.3
7	29.8	70.4	63.7	-6.7	+2.9	-13.3
8	12.9	75.0	72.0	-3	-8.6	+5.3
Macro average		**70.6**	**69.2**	**-1.4**	**-1.1**	**-2.8**

Table 4.5: Table showing per-class breakdown of % of dataset and F1 score with CEL and WCEL for ExpBERT, as well as the change in F1 score, precision and recall as a result of using WCEL.

4.2.2 Summary

As before, as these results were being used as a baseline, it was important to ensure they were reliable. To do this I compared my results to the findings published by Murty et al [55] who found that including explanations improved the model performance over NoExp by "3–10 F1 points" depending on the dataset used. A similar increase can be seen in my results, with ExpBERT improving performance over NoExp by 7 points ($70.6 - 63.6$). This shows these results are reliable and can be used as a baseline when comparing the impact of BERT-IE to ExpBERT, in line with Objectives 2 and 5.

4.3 BERT-IE

Having reproduced baseline results for NoExp and ExpBERT, the last set of experiments was to finally test the performance of my new proposed method, BERT-IE, as described in Section 3.1.3. I also ran experiments with different pre-trained models in order to further explore whether lighter pre-trained models (Section 2.4.1) still gave good performance when used in BERT-IE.

I hypothesised that using BERT-IE would give an increased performance over using NoExp and would take approximately the same amount of time to classify, given the size of the embeddings in BERT-IE were only slightly bigger (876 compared to 768). I also hypothesised that using BERT-IE would result in

no change or a slight decrease in performance compared to ExpBERT, but would be significantly faster to classify, due to the size of embeddings in BERT-IE being 31 times smaller (876 compared to 27,648). Therefore, BERT-IE would be less computationally expensive and require less memory usage.

4.3.1 Pre-trained model: `bert-base-uncased-MNLI`

As was the case with NoExp and ExpBERT, the first step was to create the embeddings by passing the expanded dataset through the first pre-trained model, `textattack/bert-base-uncased-MNLI`. I chose to test this model first as it was the most similar to the "BERT fine-tuned on MultiNLI" model used by Murty et al. [55], in their implementation of ExpBERT. This process created embeddings of size 876 for each row, which when reshaped and appended to the original tweet embeddings, resulted in a total size of [17117, 876]. This required 58MB of space, and took 23.75 hours to create.

As before, I first classified the tweets using the default hyperparameters, which took five minutes, the same time as NoExp. Figure 4.11 shows the loss curve over the 40 epochs. As was the case with NoExp, 1e-3 was not a suitable learning rate, as the model does start to learn but within five epochs, the validation loss increases rapidly due to the model overfitting on the training data, and therefore is not able to generalise to the validation data.

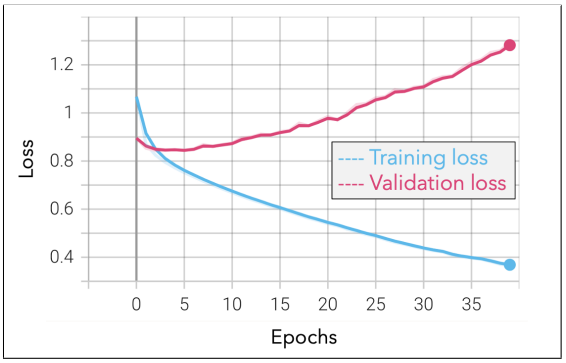


Figure 4.11: Loss curve for BERT-IE with a default learning rate of 1e-3, weight decay of 1e-2 and an unweighted loss function.

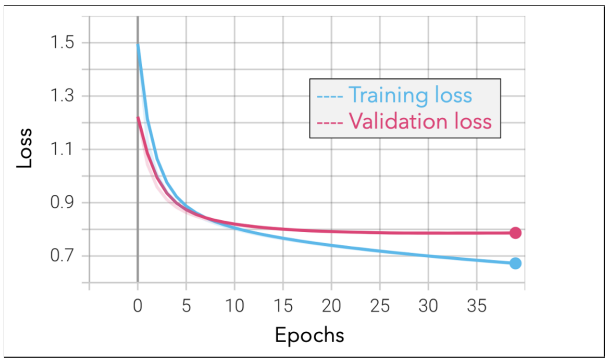


Figure 4.12: Loss curve for BERT-IE with a **changed learning rate of 5e-5**, weight decay of 1e-2 and an unweighted loss function.

From this experiment it appeared that BERT-IE would perform in a very similar way to NoExp, and for this reason the next learning rate I chose was 5e-5, the optimal learning rate for NoExp and ExpBERT. Figure 4.12 shows the loss curve with a learning rate of 5e-5. It is clear to see, that this learning rate is much more suitable, as the model is able to converge and produce a stable loss for the validation dataset. Moreover, a stable performance is seen after epoch 20 in Figure 4.13. It is interesting to note that the number of epochs required before the model converges is not reflected in both curves as the same. The loss curve suggests the model fully converges after 25 epochs, after which the loss does not change by an amount visible to the human eye. In contrast, the F1 curve suggests the model converges after 30 epochs, after which the performance starts to decrease, potentially caused by overfitting.

For this reason, I chose to change the weight decay from the default 1e-2, to 1e-1, 10 times larger. As can be seen in Figure 4.14, this produces better results, as the model performance remains stable throughout. From this, it is clear to see that BERT-IE only requires 25 epochs to converge, which takes three minutes. This is quicker than NoExp which took 40 epochs (four minutes and 30 seconds) to converge. Although the performance is not harmed by running the classifier for 40 epochs, for the remainder of the experiments I chose to use 25 epochs, as that is less computationally expensive, a key focus throughout this project.

Finally, I tested the impact of including weights in the loss function. Given that this caused a decrease in performance for NoExp and ExpBERT, I was not surprised to see the same result for BERT-IE. Therefore it can be concluded that WCEL is also not suitable for this method.

Overall, it can be seen that BERT-IE outperforms NoExp, and produces comparable results to ExpBERT, preserving 98.6% of the performance, as hypothesised.

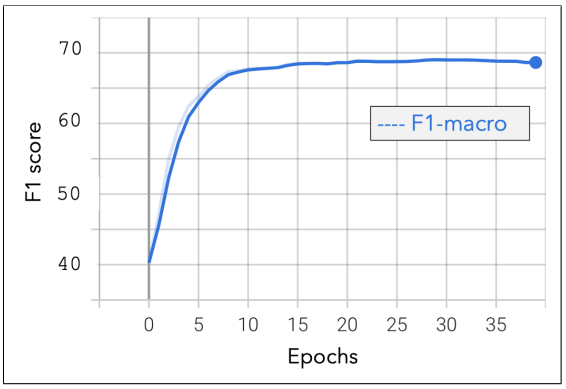


Figure 4.13: F1-macro curve for BERT-IE with an NLI pre-trained model with a learning rate of 5e-5, **weight decay of 1e-2** and CEL.

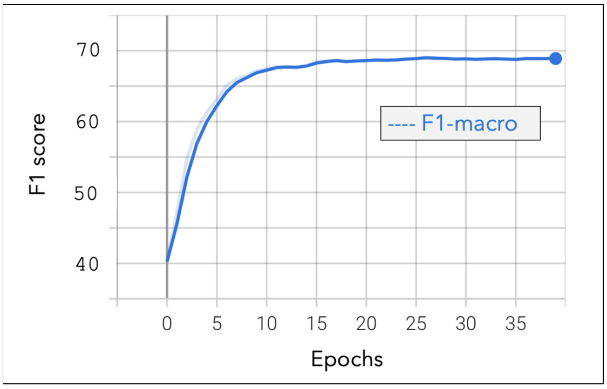


Figure 4.14: F1-macro curve for BERT-IE with an NLI pre-trained model with a learning rate of 5e-5, a **changed weight decay of 1e-1** and CEL.

4.3.2 Pre-trained model: `distilbert-base-uncased-finetuned-mnli`

The next model I chose to experiment with was `distilbert-base-uncased-finetuned-mnli`. This pre-trained model was first distilled from `bert-base-uncased` then fine-tuned on the MultiNLI dataset [95], the same dataset used when fine-tuning `textattack/bert-base-uncased-MNLI`, as used above. As described in Section 2.4.1, the distilled version of BERT is said to be "smaller, faster, cheaper and lighter" [73] as it is 40% smaller than BERT and 60% faster, ideal for reducing the computational expense. I chose to run some experiments on this pre-trained model to explore if there was a trade-off between performance and computational effort when using BERT-IE.

Passing the expanded dataset through this pre-trained model took 12.75 hours, just over half (53.7%) of the time taken to create the embeddings with `textattack/bert-base-uncased-MNLI`. I then classified the tweets using the best configuration found for `textattack/bert-base-uncased-MNLI` above. This was a learning rate of 5e-5, weight decay of 1e-1 and CEL.

Figure 4.15 shows the performance with this pre-trained model and configuration. It is clear to see that by epoch 20 the model has converged and the performance remains stable, the same trend as seen for `textattack/bert-base-uncased-MNLI`. The time taken to classify the tweets using the model head also does not change with this pre-trained model, however this was expected as the same amount of data is being passed through the classifier, regardless of the pre-trained model. This means to run the classifier for 40 epochs, it also takes four minutes and 30 seconds. For 20 epochs, the classifier takes two minutes and 15 seconds. Table 4.6 shows how the performance of this fine-tuned distilBERT pre-trained model compares with the fine-tuned BERT pre-trained model. It is clear to see that there is a 5% decrease in overall performance between distilBERT and BERT, with all nine classes seeing a decrease. This was expected as the producers of distilBERT [73] also found a slight loss in decrease, with distilBERT retaining "97% of BERT performance". However, given the significant time speed up, distilBERT appears to be a reasonable pre-trained model to use, in particular where quick classification is valued over very high performance.

4.3.3 Pre-trained model: `nli-distilroberta-base`

The final pre-trained model to run experiments with was `cross-encoder/nli-distilroberta-base`. Like the other NLI pre-trained models, this model was trained on MNLI [95], and additionally on SNLI [9]. As described in Section 2.4.1, RoBERTa was trained without NSP which was used in BERT to understand dependencies between sentences [15]. For this reason, I expected a slight drop in performance, as was seen by Devlin et al. [15] during the training of BERT, before NSP. However, Liu et al. [48], the authors of RoBERTa, measured better performance than BERT with RoBERTa, suggesting an increase in performance could be seen in this experiment instead.

For this pre-trained model, it took 12.5 hours to create the embeddings, due to the small architecture of this pre-trained model. This is 52.6% of the time required for `textattack/bert-base-uncased-MNLI`. As with distilBERT, I ran the experiment with the best configuration found for `textattack/bert-base-uncased-MNLI`. However, I found that a learning rate of 5e-5 was too low for

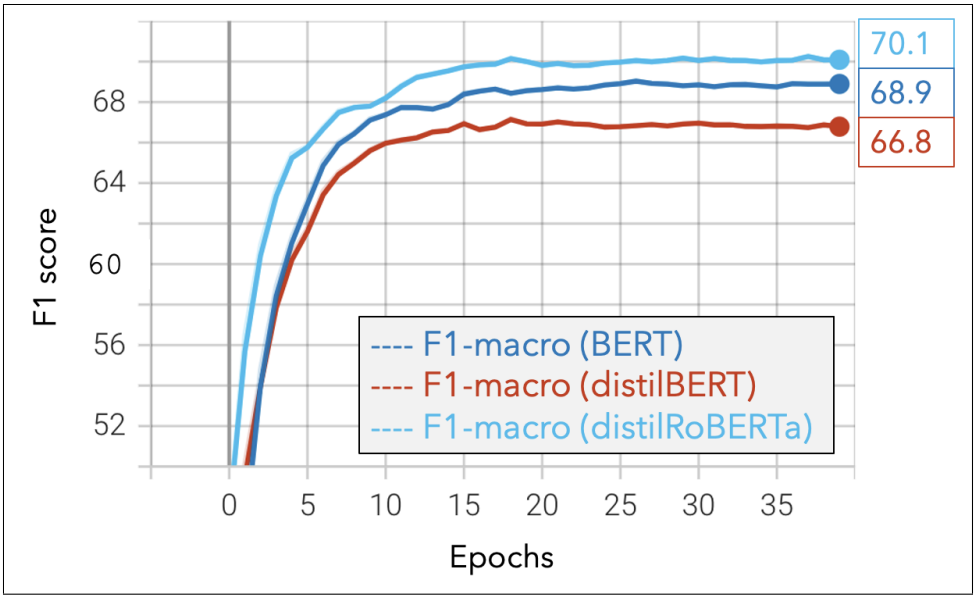


Figure 4.15: F1-macro curve for BERT-IE comparing the performance using three different pre-trained models.

this model, and instead 1e-4 was more suitable. Additionally, the default weight decay, 1e-2, was also more suitable. All other hyperparameters remained unchanged. Figure 4.15 shows how the performance of distilRoBERTa compares to that of distilBERT and BERT. It can be seen that distilRoBERTa requires 30 epochs to converge (which took three minutes and 30 seconds), and converges at a score higher than both BERT and distilBERT (70.1 compared to 68.9 and 66.8 respectively) during training. In contrast, distilRoBERTa does not outperform BERT when ran with the test dataset. Table 4.6 provides a per-class breakdown of the overall performance. Here it can be seen that although distilRoBERTa does outperform BERT for some classes, such as classes five and eight, the overall performance is very marginally lower (69.1 compared to 69.6).

This result is expected as due to the absence of NSP the NLI would not be able to perform as well. DistilRoBERTa could be outperforming BERT during training due to a better initialisation of weights which then leads to overfitting, and therefore the trained weights cannot be generalised as well to the test dataset. In contrast, distilRoBERTa could be outperforming distilBERT due to the way in which it was trained. As the full RoBERTa reaches a better performance than BERT, it is likely this increase is also carried to the distilled versions, during the distillation process, resulting in better performance.

Class No.	**% of dataset**	**F1 score (BERT, 25 epochs)**	**F1 score (distilBERT, 20 epochs)**	**F1 score (distilRoBERTa, 30 epochs)**
0	13.1	89.7	89.2	89.3
1	2.2	49.4	46.2	48.8
2	3.1	70.1	69.9	63.8
3	7.9	66.5	66.0	65.9
4	14.1	80.1	75.4	79.1
5	5.8	44.5	38.8	46.1
6	11.0	81.3	75.5	82.1
7	29.8	70.9	68.3	70.6
8	12.9	74.0	65.4	76.4
Macro average		**69.6**	**66.1**	**69.1**
Time taken to prepare embeddings (hours)		**23.75**	**12.75**	**12.5**
Time taken for classification (minutes)		**3**	**2.25**	**3.5**

Table 4.6: Table showing per-class breakdown of % of dataset and F1 score for BERT-IE trained with a BERT, distilBERT and distilRoBERTa pre-trained model.

4.3.4 Summary

Overall, as already introduced, Table 4.6 and Figure 4.15 give an overview of how the three pre-trained models compare. It is good to see that as expected distilBERT and distilRoBERTa only perform marginally worse than BERT, whilst taking just over half the time BERT does. This suggests there is a trade-off to be made depending on whether time, computational expense or performance is the priority. The following section provides a more detailed analysis into where each model performs well, which classes the model struggles to classify, and therefore which model is more suitable in different scenarios and where further explanations may be required.

4.4 Comparison of methods

Having obtained results from hyperparameter tuned models using each of the three methods, NoExp, ExpBERT and BERT-IE, the only step left in this project was to investigate where a method performed better than the others, and why, and where it could have been improved, and why. This is in line with Objective 5. I hypothesised that BERT-IE would outperform NoExp, but I would either see a maintenance or decrease in the F1 score between BERT-IE and ExpBERT. However, BERT-IE would be a lot quicker, given the reduced size of the dataset being passed through the model head. I also hypothesised that distilBERT would perform worse than full BERT.

Table 4.7 summarises the F1 score, memory usage and computational time introduced above for the hyperparameter tuned models using NoExp, ExpBERT and BERT-IE with a pre-trained model fine-tuned on MNLI [95]. This model was `textattack/bert-base-uncased-MNLI`.

	NoExp	**ExpBERT**	**BERT-IE w/ BERT**
F1-macro average	63.6	70.6	69.6
Time taken to prepare embeddings (hours)	0.75	22.75	23.75
Epochs taken for classification	40	15	25
Time taken for classification (minutes)	4.5	20	3
File size of embedding (MB)	51	1800	58

Table 4.7: Table containing results from hyperparameter tuned models using NoExp, ExpBERT and BERT-IE.

From these results I am able to conclude that ExpBERT is the best performing method, albeit the most computationally expensive method due to the large size embeddings being passed through the model head. BERT-IE, however, is able to preserve 98.6% of ExpBERT's performance, whilst taking a sixth of the time to classify the tweets. Additionally, the memory required to store the embeddings for BERT-IE is 31 times smaller than ExpBERT. This indicates that **BERT-IE is a suitable method for classifying tweets** and meets criteria 3 and 5. However, it is important to understand at a class level how BERT-IE compares to NoExp and ExpBERT.

4.4.1 BERT-IE vs NoExp

As can be seen in Table 4.8, BERT-IE provides improved performance over NoExp for every class, but the increase is not consistent. For example, for class one, the improvement in performance is a staggering 14 points, whereas for class seven, the improvement is only 2.6 points.

Upon inspection it is clear to see that the biggest increase is seen for classes one and two, the smallest classes in the dataset, only forming 2.2% and 3.1% respectively. In contrast, the two smallest increases are seen for classes zero and seven, which are also the largest classes in the dataset, forming 13.1% and 29.8% of the dataset respectively. This suggests that the **explanations are more helpful for the smaller classes** and therefore they are worthwhile including, thus satisfying criterion 4. It is surprising to see that classes five and eight do not follow this trend. Class five, the third smallest and second poorest performing class, only sees an increase in performance of 2.8 points with explanations. Contrastingly, class eight, an average performing class sees an increase of 5.8 points with the inclusion of explanations. To further explore this, I produced a confusion matrix for both NoExp (Figure 4.16) and BERT-IE (Figure 4.17), to understand where these classes were commonly being misclassified.

Class No.	Class label	% of dataset	F1 score (NoExp)	F1 score (BERT-IE)	Increase in performance
0	**injured or dead people**	**13.1**	87.2	89.7	2.5
1	**missing trapped or found people**	**2.2**	35.4	49.4	14
2	**displaced people and evacuations**	**3.1**	56.9	70.1	13.2
3	**infrastructure and utilities damage**	**7.9**	62.2	66.5	4.3
4	**donation needs or offers or volunteering services**	**14.1**	75.1	80.1	5
5	**caution and advice**	**5.8**	41.7	44.5	2.8
6	**sympathy and emotional support**	**11.0**	77.1	81.3	4.2
7	**other useful information**	**29.8**	68.3	70.9	2.6
8	**not related or irrelevant**	**12.9**	68.2	74.0	5.8
Macro average			**63.6**	**69.6**	**6**

Table 4.8: Table highlighting the increase in performance per class from using BERT-IE over NoExp.

From Figure 4.16, it can be seen that class five is often confused with class seven, and the inclusion of explanations using BERT-IE has marginal impact. The label for class five is "caution and advice" and the label for class seven is "other useful information". This indicates that the classifier is not able to distinguish which tweets contain useful advice and which tweets are about the natural disaster but don't provide useful information. An example of a misclassified tweet is:

> ":California governor declares state of emergency after quake http"

This tweet is subjective for even a human to classify as the need for caution is inferred given the state of emergency, however it could also fit into class seven as it does contain useful information. Similarly, other tweets with the "caution and advice" gold label could fit in to the "other useful information" class, such as:

> "Rains now pouring in LB. Ruby P H"

and

> "Jammu and Kashmir govt was deaf to flood alert http http"

Therefore, it can be concluded that in future research (Section 5.2), **class five should be relabelled to be more specific**, allowing humans and classifiers to be able to distinguish tweets from this class better, and in turn improve performance.

Class eight was the other surprising class, with a 5.8 point increase between NoExp and BERT-IE, albeit one of the larger classes. In Figure 4.16, it can be seen that tweets from class eight which is "not related or irrelevant" are often misclassified as "other useful information". However, by including explanations, this misclassification decreases, suggesting that the **explanations provided for class eight are particularly useful**.

It is also interesting to note that when using NoExp, some tweets from almost all classes are misclassified as class seven. This is likely because class seven acts as a 'fall-back' class, where if tweets don't fit the other classes they will default to class seven. An improvement can be seen for some classes, such as classes two and four, when using BERT-IE, however not all. Therefore, an area for further research can be analysing the tweets which were mispredicted as class seven, and producing new labels that are more specific, like was the case with class five.

Overall, it can be concluded that despite the increase in computational time and effort required to prepare the embeddings, it is **worthwhile including explanations using BERT-IE.** This meets criterion 4 as outlined in Section 3.1.1.

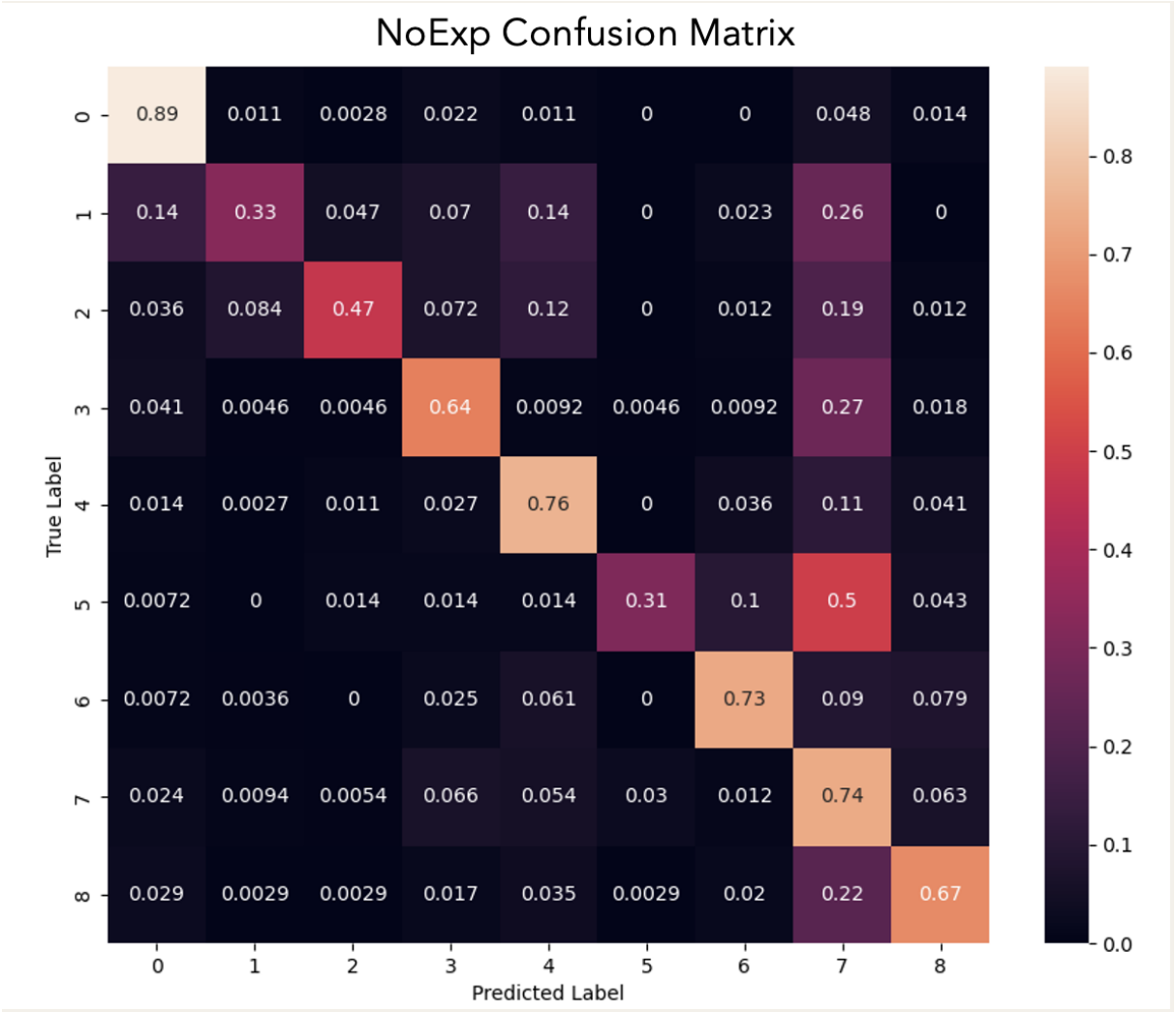


Figure 4.16: Confusion Matrix for **NoExp** indicating which classes are often confused with each other. The lighter the cell, the stronger the classification.

4.4.2 BERT-IE vs ExpBERT

As mentioned above, **BERT-IE is able to preserve 98.6% of ExpBERT's performance** and therefore appears to be a suitable, lighter alternative to ExpBERT. This section focuses on BERT-IE vs ExpBERT at a class level.

Table 4.9 presents the change in performance between ExpBERT and BERT-IE for all classes. For most classes, BERT-IE suffers a performance loss, however this is not consistent, and for some classes BERT-IE even outperforms ExpBERT. For example, there is a large performance loss of 6.3 for class two, the smallest class, whereas there is an increase in performance for class three, the second smallest class, of 3.4. It is clear to see that in this case there is no correlation between the class size and the change in performance.

The average change is -0.98, and only three out of nine classes (class zero, four and eight) are within 0.5 points of this (-1.48 – -0.48), as is the overall change. The other classes however, see a more significant change, such as class five which decreases in performance by 4.2 points between BERT-IE and ExpBERT. This suggests that the increase or decrease in performance may be more dependent on the quality of the explanations provided to a class, rather than the size of the class. To explore this possibility further, I also produced a confusion matrix for ExpBERT, as can be seen in Figure 4.18.

The first class of interest is class one, the smallest class and the second poorest performing class in both ExpBERT and BERT-IE. It was also the class with the biggest loss in performance. When classified with ExpBERT, although most tweets were correctly identified, a minority of tweets were classified as class

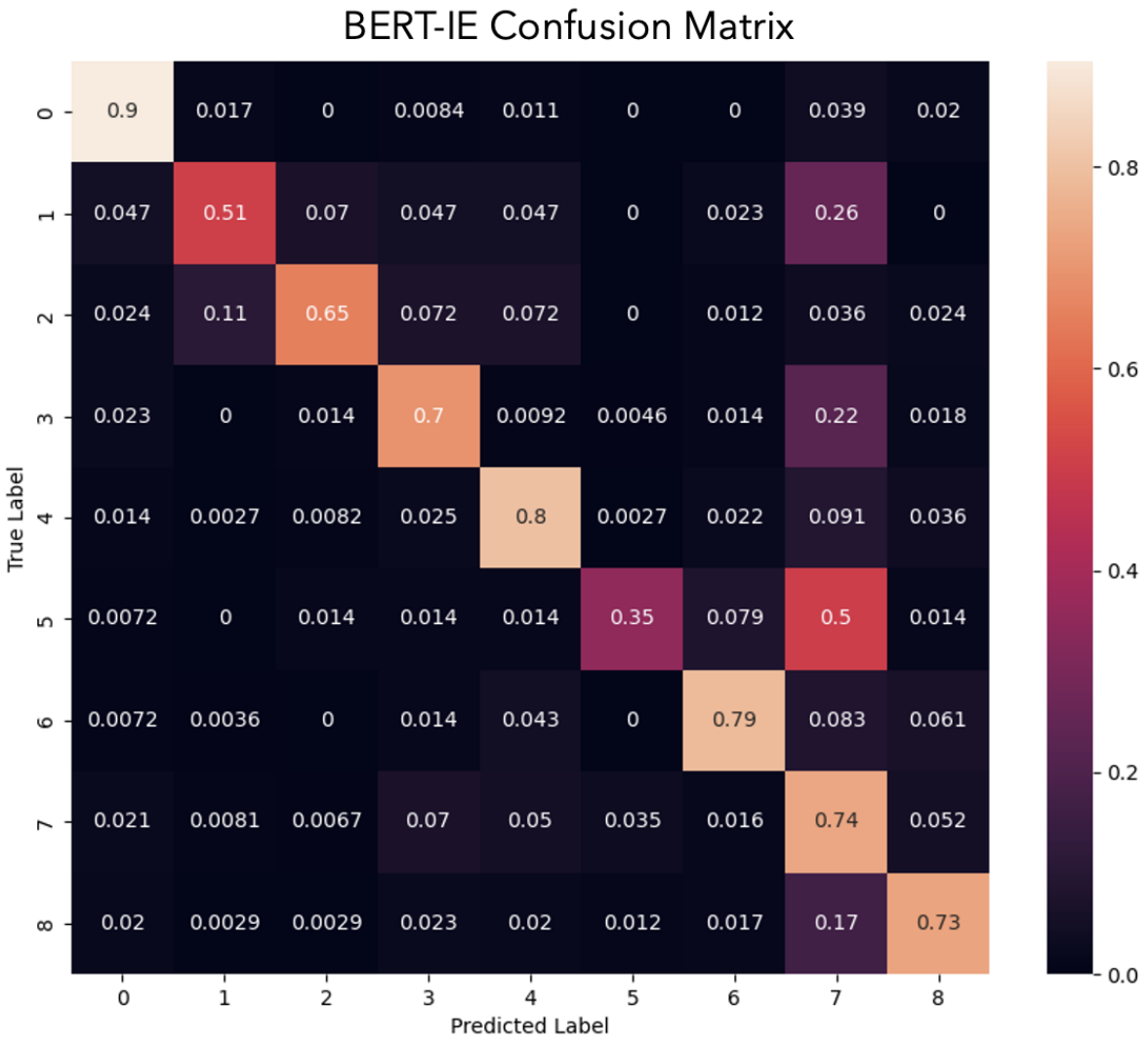


Figure 4.17: Confusion Matrix for **BERT-IE** indicating which classes are often confused with each other. The lighter the cell, the stronger the classification.

three: "infrastructure and utilities damage". Considering the label for class one is "missing, trapped or found people" this is a surprising result as the classes are quite different. In contrast, when classified with BERT-IE (Figure 4.17), class three was often mistaken for class seven, "other useful information". As discussed when comparing BERT-IE with NoExp, class seven is seen as the 'fall-back' class. This suggests that some of the information stored in ExpBERT is useful for differentiating between classes one and seven, however this information is not encapsulated in the NLI used in BERT-IE, thus resulting in a decrease in performance. It is also important to note at this point that class one is very broad, and therefore is already hard to classify, as well as possibly not very useful in disaster recovery. For example, the response to a 'found' person is likely to be an ambulance or shelter, whereas the response for a 'trapped' person is likely to be a rescue team. This again brings out the **need for new, more specific class labels.** Overall, this means the poor performance of the class (in NoExp) is amplified by **BERT-IE, which fails to capture all of the information needed that is stored in ExpBERT**.

The second interesting class to analyse is class two, "displaced people and evacuations". In contrast to class one discussed above, class two sees an *increase* in performance when classified with BERT-IE, despite being the second smallest class. When classified with ExpBERT, class two is sometimes mistaken as class one, "missing, trapped or found people". The same trend is seen with BERT-IE, with approximately the same proportion of tweets being confused as class one. However, there is a smaller proportion of tweets misclassified as any of the other classes in BERT-IE. This suggests that the large amount of information stored in the **ExpBERT embeddings are creating noise** for the model head and therefore it is sometimes wrongly confusing the classes. In contrast, this **noise is reduced in BERT-IE**, helping the

Class No.	Class label	% of dataset	F1 score (ExpBERT)	F1 score (BERT-IE)	Change in performance (BERT-IE – ExpBERT)
0	**injured or dead people**	**13.1**	90.8	89.7	-1.1
1	**missing trapped or found people**	**2.2**	55.7	49.4	-6.3
2	**displaced people and evacuations**	**3.1**	66.7	70.1	+3.4
3	**infrastructure and utilities damage**	**7.9**	69.0	66.5	-2.5
4	**donation needs or offers or volunteering services**	**14.1**	81.2	80.1	-1.1
5	**caution and advice**	**5.8**	48.7	44.5	-4.2
6	**sympathy and emotional support**	**11.0**	77.9	81.3	+3.4
7	**other useful information**	**29.8**	70.4	70.9	+0.5
8	**not related or irrelevant**	**12.9**	75.0	74.0	-1
Macro average			**70.6**	**69.6**	-1

Table 4.9: Table highlighting the change in performance per class from using BERT-IE over ExpBERT.

model head to perform better. Again, it is important to note that the class labels are ambiguous. For example, a 'found' person could also be referred to as a 'displaced' person, and 'trapped' people may need to be 'evacuated'. Thus **there is an overlap between classes one and two**, and this could be a reason for the confusion seen in both ExpBERT and BERT-IE, despite the inclusion of explanations.

Finally, class five also performs poorly for ExpBERT in addition to NoExp and BERT-IE as discussed above, with most classes again being misclassified as class seven. Although ExpBERT does outperform BERT-IE for this class, the poor performance of the class is more likely due to either the quality of the explanations, or the ambiguity of the class label, as opposed to the NLI not being able to capture all of the necessary information.

Having looked at the change in performance between ExpBERT and BERT-IE, it is also important to obtain a more holistic view of the two models, taking into account computational effort and time. As shown in Table 4.7, the **time taken to prepare the embeddings was similar**, as in both methods the expanded dataset needed to be passed through the pre-trained model. However, the embeddings file created in **BERT-IE was 31 times smaller, due to the NLI output**. This smaller file is easier to handle, in particular for small businesses who may not have the facilities to store large files locally or on a supercomputer. Furthermore, passing the embeddings through the model head is **six times faster in BERT-IE than in ExpBERT**, which is more suitable for time-critical situations, such as natural disasters.

Therefore, it can be concluded that despite the slight loss in performance (1.4 points), the increase in speed and decrease in computational expenditure renders BERT-IE a suitable alternative to ExpBERT, in particular for smaller businesses or organisations, whose focus is on faster, lighter computation over performance. This **meets the high-level aim of this project**, introduced in Section 1.6.

4.4.3 Pre-trained models used in BERT-IE

Lastly, having concluded that BERT-IE is a suitable alternative method to ExpBERT, all that is left is to explore if using a distilled pre-trained model can provide another, even quicker alternative, with minimal loss in performance.

As introduced in Section 4.3.4, passing the expanded dataset through a distilled model of BERT and RoBERTa takes just over half the time than with BERT, whilst only performing slightly worse. Table 4.10 shows the per class breakdown of BERT-IE with a BERT pre-trained model, and the change in performance when using distilBERT and distilRoBERTa. Unsurprisingly, it can be seen that all classes suffer from a performance loss with distilBERT and six out of nine classes suffer with distilRoBERTa.

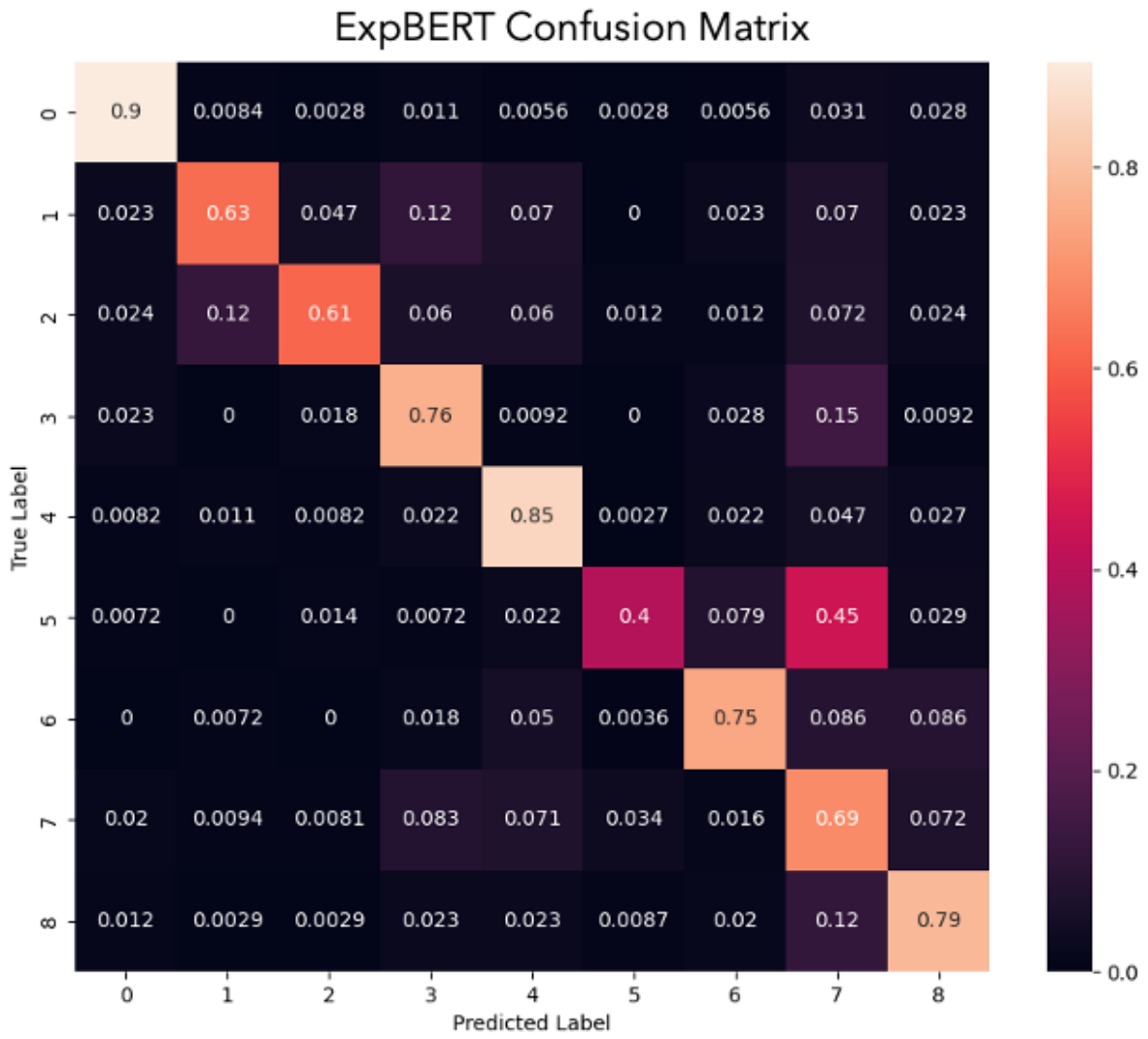


Figure 4.18: Confusion Matrix for **ExpBERT** indicating which classes are often confused with each other. The lighter the cell, the stronger the classification.

Again, there does not appear to be any correlation between the class size and the decrease in performance, as the smallest decrease in performance for distilBERT (0.2) is seen in class three, the second smallest class, whereas the largest decrease (8.6) is seen in class eight, one of the largest classes.

To fully explore what factors impact the change in performance, I produced a confusion matrix for distilBERT(Figure 4.19) to compare with Figure 4.17. The first class of interest is class four. This class is for tweets with "donation needs or offers or volunteering services". I would expect that the classifier can easily identify these tweets and also that not many tweets would get misclassified as class four as the class is quite specific. While the latter holds true, this is not the case for the former. With distilBERT, the classifier struggles to differentiate tweets from class four and class seven, "other useful information", resulting in a decrease of 4.7 points from BERT. This may simply be because the smaller architecture of distilBERT cannot hold enough information to identify the nuances in these tweets, such as the offer of money rather than informing how much the disaster relief will cost.

The next class to discuss is class one: "missing, trapped or found people". In the previous subsection, the broad nature of the class was discussed as a possible reason for poor performance. With distilBERT, tweets from class one are often misclassified as class three, "infrastructure and utilities damage". This may be because tweets that mention damage, commonly include information about the human impact, such as the following tweet which mentions both damage and missing people, but has a true label of class one.

Class No.	Class label	% of dataset	F1 score (BERT-IE w/ BERT)	Change in performance (distilBERT – BERT)	Change in performance (distilRoBERTa – BERT)
0	**injured or dead people**	**13.1**	89.7	-0.5	-0.4
1	**missing trapped or found people**	**2.2**	49.4	-3.2	-0.6
2	**displaced people and evacuations**	**3.1**	70.1	-0.2	-6.3
3	**infrastructure and utilities damage**	**7.9**	66.5	-0.5	-0.6
4	**donation needs or offers or volunteering services**	**14.1**	80.1	-4.7	-1
5	**caution and advice**	**5.8**	44.5	-5.7	+1.6
6	**sympathy and emotional support**	**11.0**	81.3	-5.8	+0.8
7	**other useful information**	**29.8**	70.9	-2.6	-0.3
8	**not related or irrelevant**	**12.9**	74.0	-8.6	+2.4
Macro average			**69.6**	**-3.5**	**-0.5**

Table 4.10: Table highlighting the change in performance per class from using BERT-IE with BERT to BERT-IE with distilBERT and BERT-IE with distilRoBERTa.

> "Earthquake Chile - 10 to 15 children missing - 80% of the fishing boats in La Caleta damaged due to the tsunami - http"

DistilBERT then struggles to classify this tweet as it is unable to understand as much sentiment as BERT. Between ExpBERT and BERT-IE, a decrease in performance of 6.3 had already been seen for this class. By using a distilled pre-trained model, a further loss of 3.2 is seen, resulting in a total loss of 9.5 points from ExpBERT. This suggests that **although lighter, distilBERT is not a suitable model for classification.**

The third class of interest is class eight. As seen before tweets from class eight, "not related or irrelevant", are commonly misclassified as class seven, "other useful information". This problem is amplified with distilBERT, with a significant decrease of 8.6 points seen. This means the performance of BERT-IE with distilBERT is only 2.8 points higher than the performance of NoExp. Furthermore, misclassifying these tweets, means that for a person manually looking through class seven, to extract the "other useful information", there is a lot more irrelevant information they have to look through. Therefore, **this reduces the effectiveness of the model.**

Having investigated BERT and distilBERT, I was interested in the cases where distilRoBERTa outperformed BERT. This was for classes five, six and eight. Notably, these were the three classes that suffered the biggest losses in distilBERT.

The first class to look at is class five. For BERT, distilBERT and distilRoBERTa, this class was commonly mistaken with class seven. This trend was also seen in NoExp and ExpBERT. Although distilRoBERTa does outperform BERT, this is a marginal increase as can be seen in Figures 4.17 and 4.20. Similarly, for classes six and eight where distilRoBERTa does outperform BERT, this is only a marginal increase.

However, the performance of class two for distilRoBERTa decreases by 6.3 points compared to BERT. Although distilRoBERTa does still manage to predict class two correctly 63% of the time, it commonly gets confused with class seven. This was not the case with BERT. Overall, by observing Figure 4.20, it can be seen that distilRoBERTa also is not able to distinguish tweets from the true label and class seven, meaning it is **also not a suitable model for classification.**

To conclude, although using distilBERT and distilRoBERTa results in a decrease in time taken to produce the embeddings for the expanded dataset, compared to BERT, the decrease in performance (3.5 points for distilBERT) seen does not make it worthwhile. Moreover, this decrease leaves an improvement over NoExp of only 2.5 points (66.1 to 63.6), despite taking over 16 times longer to prepare the

embeddings. Therefore, BERT-IE with either a **distilBERT or distilRoBERTa pre-trained model is not a suitable alternative** method to ExpBERT. **Neither model are able to perform as well as BERT**, which as seen before retains 98.6% of ExpBERT's performance whilst taking only a sixth of the time to pass the embeddings through the model head.

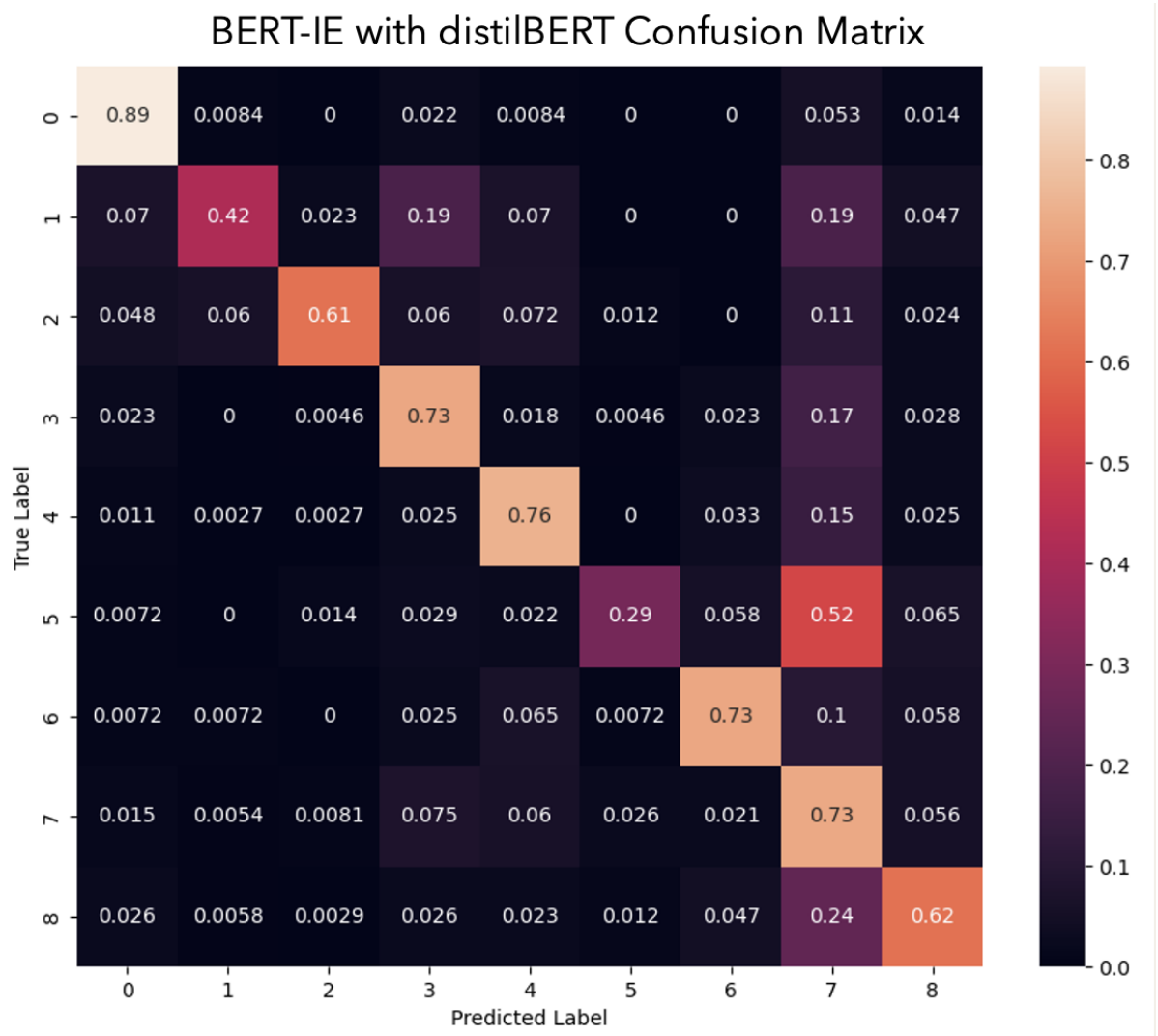


Figure 4.19: Confusion Matrix for **BERT-IE with distilBERT** indicating which classes are often confused with each other. The lighter the cell, the stronger the classification.

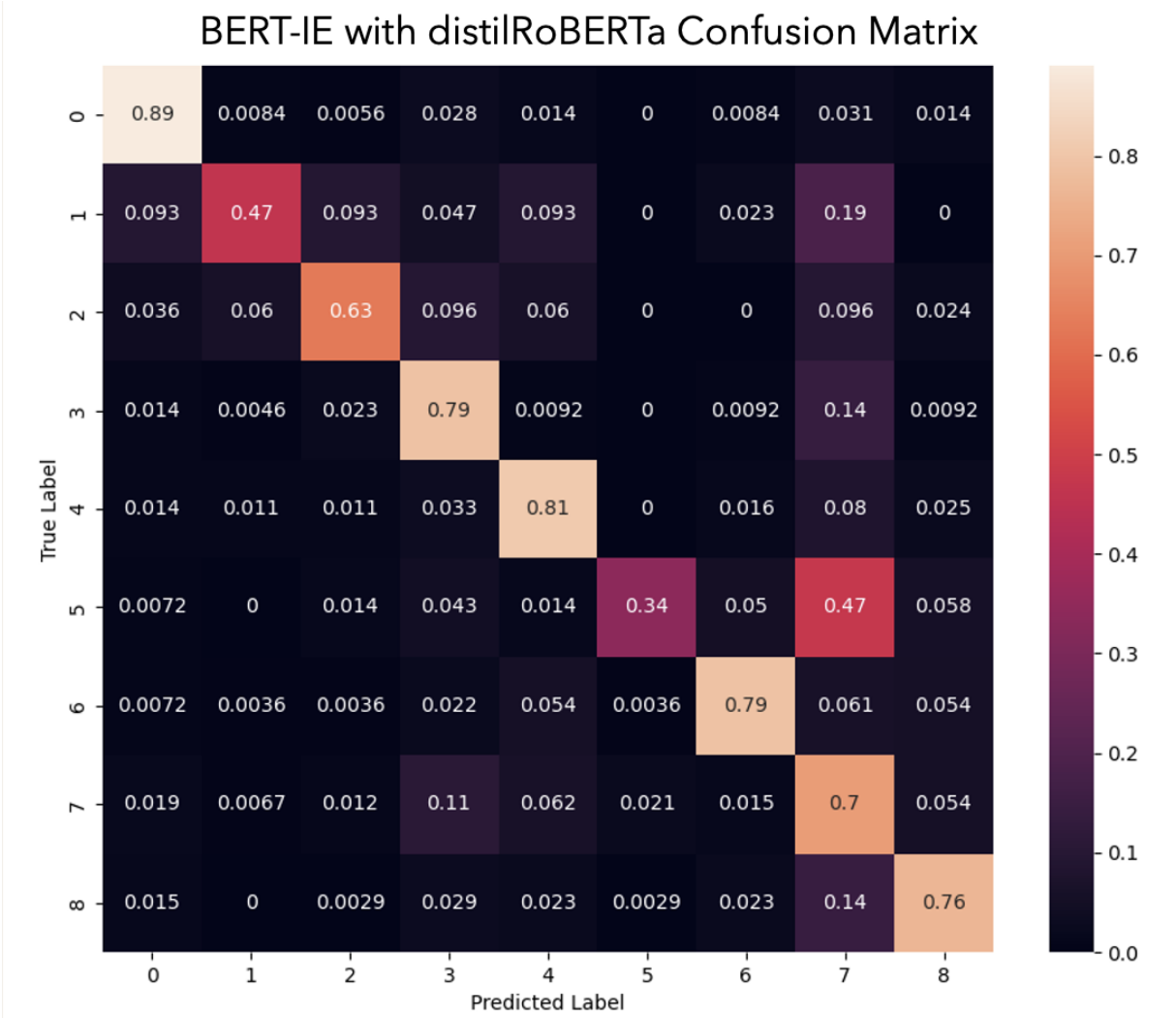


Figure 4.20: Confusion Matrix for **BERT-IE with distilRoBERTa** indicating which classes are often confused with each other. The lighter the cell, the stronger the classification.

Chapter 5

Conclusion

5.1 Summary of work completed

In this project a new method, BERT-IE, has been designed, implemented and compared to existing methods in the research area. Furthermore, baseline results for two existing methods, NoExp and ExpBERT, have been reproduced, with technical errors in the previous implementation corrected, resulting in a 57% increase in speed.

The first step in the project was gaining a broad overview of the existing literature to situate this project. Two notable works were identified. The first was by Murty et al. [55], who developed a method known as ExpBERT. The method involved the creation of an expanded dataset, where each tweet was concatenated with each explanation and then passed through a pre-trained model and model head. This resulted in an improvement in performance, however, at the cost of computational resources, due to the large embeddings being produced. Hence, it can be concluded that **Objective 1 in this project has been met.** The second notable work was that of Pilitsis [62] who adapted ExpBERT for the document classification task and tested this on the CrisisNLP dataset [36].

Having understood the research area and ascertaining the need for a new method that included HITL explanations in a way that resulted in low dimensional outputs, a new method, BERT-IE, was designed. A detailed criteria for the design of this method was outlined and met. Consequently, BERT-IE overcame a significant hurdle of ExpBERT by producing embeddings significantly smaller. The factor by which the representations are smaller varies depending on the number of explanations included, however for this implementation the representations produced using ExpBERT were of size 27,648 whereas the representations produced using BERT-IE were of size 876. This is 31.6 times smaller. As a result, the time needed to classify the tweets using the model head saw a six-fold decrease. Thus it can be concluded that **Objective 2 of this project has also been met**.

The new designed method was implemented in a clear, simple way following best practices. The need for an easily understandable code base was kept front of mind throughout the implementation and influenced many implementation choices. One example is the use of command line arguments to facilitate the change of pre-trained model allowing different experiments to be run easily and efficiently. Furthermore, a seed was used to enable researchers in the future to reproduce the results I have obtained. All implementation choices were thoroughly thought through and reasoned, and it is clear to see that **Objective 3 has been met.**

A wide range of experiments were then carried out in order to assess the performance of BERT-IE on the CrisisNLP dataset [36], in line with Objective 4. Each configuration of experiment was hyperparameter tuned, an important step to ensure the best results for that experiment are reached. A variety of appropriate performance metrics have been used, such as precision and recall, to assess the outcomes of the numerous experiments run, and to motivate more experiments. The performance of each experiment was assessed using the F1 score, and a loss curve and confusion matrix produced for each. This allowed me to **meet Objective 4.**

Finally, having ran all of the experiments and hyperparameter tuning the models created, the performance obtained using each method could be compared and contrasted. It was clear to see that using BERT-IE

preserved 98.6% of the performance of using ExpBERT, whilst taking only one sixth of the time to classify the tweets, **as desired in Objective 2**. The impact of using different pre-trained models in BERT-IE was also investigated, with a decrease in performance seen when using a distilBERT model. Each comparison was done at a class level with individual tweets being picked out to understand where and why the model was not able to correctly classify the tweets. Overall, it was found that although using the ExpBERT method resulted in the best performance, a trade-off between performance and computational resources could be made. This **meets the final objective, Objective 5 of this project**. For scenarios where performance was key, for example for models used in healthcare for diagnoses, it was worth using ExpBERT, despite the time it took. However, for scenarios where speed was key and computational resources were limited, for example a small business wanting to understand reviews of their company, despite the slight loss in performance, it is beneficial for them to use BERT-IE.

Meeting each of these objectives allowed me to **meet the high-level aim of this project**, as a unique method was created that preserved 98.6% of the performance seen when using ExpBERT. The embeddings created were 31 times smaller and as a result it was significantly less computationally expensive to classify these embeddings, than those produced using ExpBERT.

5.2 Future Work

The following section outlines potential avenues for further research that builds on the considerable contribution to the research area of this project.

5.2.1 Quality and number of explanations

The first avenue is looking at the quality and number of explanations passed in with the tweets.

In this implementation only three explanations were passed in per class. Research by Pilitsis [62] concluded that having up to six explanations passed into ExpBERT provided marginal improvement to the performance. It would be interesting to explore however, if there is a point at which too many explanations causes the performance to drop. This would likely be due to many values being passed into the model head per tweet, meaning it is unable to extract the useful information. For example, if eight explanations were provided per tweet, and as in this implementation the textual descriptions were also used, there would be a total of 81 explanations being passed in. This would result in an embedding of size 62,208 per tweet for ExpBERT and size 1,011 for BERT-IE. As the size of the embedding for BERT-IE does not grow as large, it would be unlikely for the performance to drop. Therefore, there could be a point at which BERT-IE outperforms ExpBERT. This would be useful for scenarios where both performance and speed is key, and a trade-off between both is unsuitable.

A further exploration into the quality of each explanation can be done. Currently there is no way to identify which of the explanations provided were helping the classifier. For example, for all methods the classifier consistently correctly labelled 90% of the tweets from class zero. However, there is no way of knowing if this impressive performance is due to one, two or all of the explanations provided. As such, it would be valuable to create a method that can measure the impact of each individual explanation. One possible method could be including only one explanation per class at a time, however this would be very time consuming and would not provide insight into how multiple explanations within a class interact. Therefore a more robust, efficient method is needed. Knowing what makes a good explanation and having examples of them, would be particularly useful for the next avenue of further research: automating the creation of explanations.

In the research done by Murty et al. [55] and Pilitsis [62], explanations were created manually, with each explanation taking approximately one minute to create. Having to manually create numerous explanations can result in a variety in the quality of the explanations due to human fatigue during the creation of explanations and different judgements of what makes a good explanation. Furthermore, as with all human input there is a risk of bias being included in the explanations. This would result in a bias being baked into the model. To overcome these problems, the creation of explanations can be automated. For example, if a criteria for a good explanation and many examples of good explanations were provided to an online bot, such as chatGPT [12], many explanations could be created. By automating the process, the quality of these explanations would not vary due to human fatigue and can also be cheaper than having humans create them. However, the problem of bias would still persist due to bias coming from

the online bot instead. Overall, however, this could be an important avenue of further research given the benefit of including explanations, as highlighted in this project.

5.2.2 Alternative datasets

Whilst the impact of different pre-trained models on performance was investigated in this project, investigating the performance on different datasets was outside the scope of this project. This is because for each new dataset a high-level understanding of each class would need to be sought in order to create useful explanations for each. However, due to the simple, clear way in which the code base for this project has been implemented, as outlined in Section 3.2, it is possible for this investigation to be carried out as future work.

In this project, the CrisisNLP dataset [36] used consisted of tweets collected during 19 crises between 2013 and 2015, as described in Section 1.5. Given that the dataset was based on tweets collected from Twitter, it is possible that the trained models would not perform as well on new Twitter data due to a natural change/evolution in the way people use Twitter. For example, since 2015 the limit of characters on tweets has changed from 140, to 280, to currently 4,000. Therefore, it is likely that people are now more descriptive in their tweets and publish long tweets that previously would have been broken down into multiple tweets. This would mean there is more information in each tweet to be extracted and also that each tweet may no longer fit into one discrete class. To overcome this, other datasets with more recent data can be used to train the models. One possible alternative is CrisisBench [1], published in 2021. This is comprised of eight human-annotated datasets, all containing tweets collected during natural disasters over a wider range of years. By using this dataset, it can be investigated if the results found apply to more recent data.

Another interesting aspect that could be investigated is how the models perform when combined with other sources. Currently the only type of data being used is tweets published during natural disasters. However, in a natural disaster lots of other data quickly becomes available, but like tweets is hard to process quickly. One example is data provided on the Last Quake app [13]. In the event of an earthquake, people who have this app downloaded can report what they felt, their surroundings and what they see. This is qualitative information, capturing similar information to the tweets but is likely to contain more relevant information. By using both the tweets and the testimonies from the app, a wider picture of the impact of the earthquake can be gained. Additionally, using a second source allows a wider demographic to be reached. However, this data will need annotating. To determine new labels for the combined sources, advice from experts in the field can be sought. Some suggestions of new labels include splitting up some classes, such as "injured or dead people" can be split up into "injured" and "dead", and some classes can be combined. For example, disaster relief charities do not need access to tweets classed as "sympathy and emotional support" and therefore these tweets can be grouped with "other useful information". This relabelling would be of use even for CrisisNLP [36] alone, as it was clear the overlapping of some classes affected the model's performance.

Finally, my new proposed method, BERT-IE, can also be tested on different benchmarks for a range of NLP tasks. The most appropriate benchmark in this case is GLUE and subsets of GLUE, which are used for evaluating "natural language understanding systems" [93]. Obtaining results on the benchmarks would allow other researchers interested in including explanations to easily compare the performance of BERT-IE to other models, some of which produced State-Of- The-Art results like ExpBERT. This investigation would be helpful to understand how the method generalises to other domains. This is important because the small businesses or individuals choosing to use BERT-IE, over the other methods, may be passing in textual data from numerous sources in different domains.

Bibliography

[1] Firoj Alam, Hassan Sajjad, Muhammad Imran, and Ferda Ofli. CrisisBench: Benchmarking Crisis-related Social Media Datasets for Humanitarian Information Processing. In *15th International Conference on Web and Social Media (ICWSM)*, 2021.

[2] Reem AlRashdi and Simon O'Keefe. Deep learning and word embeddings for tweet classification for crisis response. *CoRR*, abs/1903.11024, 2019. URL: http://arxiv.org/abs/1903.11024, arXiv:1903.11024.

[3] Amazon. Amazon Mechanical Turk. https://www.mturk.com/. Accessed on April 17, 2023.

[4] Amazon. Cloud Computing Services - Amazon Web Services (AWS). https://aws.amazon.com/. Accessed on April 17, 2023.

[5] Amazon Web Services. Figure Eight Data Labeling Platform. https://aws.amazon.com/financial-services/partner-solutions/figure-eight/. Accessed on April 16, 2023.

[6] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. Software engineering for machine learning: A case study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 291–300, 2019. doi:10.1109/ICSE-SEIP.2019.00042.

[7] Markus Bauer, Clemens van Dinther, and Daniel Kiefer. Machine learning in SME: an empirical study on enablers and success factors. In Bonnie Brinton Anderson, Jason Thatcher, Rayman D. Meservy, Kathy Chudoba, Kelly J. Fadel, and Sue Brown, editors, *26th Americas Conference on Information Systems, AMCIS 2020, Virtual Conference, August 15-17, 2020*. Association for Information Systems, 2020. URL: https://aisel.aisnet.org/amcis2020/adv_info_systems_research/adv_info_systems_research/3.

[8] Black. Black 23.3.0 documentation. https://black.readthedocs.io/en/stable/. Accessed on April 29, 2023.

[9] Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. *CoRR*, abs/1508.05326, 2015. URL: http://arxiv.org/abs/1508.05326, arXiv:1508.05326.

[10] Britannica. Moore's law. https://www.britannica.com/technology/Moores-law. Accessed on April 16, 2023.

[11] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020. URL: https://arxiv.org/abs/2005.14165, arXiv:2005.14165.

[12] Chris Vallance. ChatGPT: New AI chatbot has everyone talking to it. https://www.bbc.co.uk/news/technology-63861322. Accessed on April 17, 2023.

[13] CSEM EMSC. LastQuake: Felt an earthquake? Share your testimony. https://m.emsc.eu/. Accessed on April 30, 2023.

[14] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. doi:10.1109/CVPR.2009.5206848.

[15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL: http://arxiv.org/abs/1810.04805, arXiv:1810.04805.

[16] elvis. Deep Learning for NLP: An Overview of Recent Trends. https://medium.com/dair-ai/deep-learning-for-nlp-an-overview-of-recent-trends-d0d8f40a776d. Accessed on April 27, 2023.

[17] Fabio Duarte. Amount of Data Created Daily (2023). https://explodingtopics.com/blog/data-generated-per-day. Accessed on April 16, 2023.

[18] Hugging Face. The hugging face course, 2022. https://huggingface.co/course, 2022. Accessed on April 27, 2023.

[19] Ahmed Gad. Beginners Ask "How Many Hidden Layers/Neurons to Use in Artificial Neural Networks?". https://towardsdatascience.com/beginners-ask-how-many-hidden-layers-neurons-to-use-in-artificial-neural-networks-\51466afa0d3e. Accessed on May 01, 2023.

[20] Samuel Gehman, Suchin Gururangan, Maarten Sap, Yejin Choi, and Noah A. Smith. RealToxicityPrompts: Evaluating neural toxic degeneration in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 3356–3369, Online, November 2020. Association for Computational Linguistics. URL: https://aclanthology.org/2020.findings-emnlp.301, doi:10.18653/v1/2020.findings-emnlp.301.

[21] GitHub. GitHub. https://github.com/. Accessed on April 18, 2023.

[22] Shantanu Godbole, Abhay Harpale, Sunita Sarawagi, and Soumen Chakrabarti. Document classification through interactive supervision of document and term labels. volume 3202, pages 185–196, 09 2004. doi:10.1007/978-3-540-30116-5_19.

[23] Yeow Chong Goh, Xin Qing Cai, Walter Theseira, Giovanni Ko, and Khiam Aik Khor. Evaluating human versus machine learning performance in classifying research abstracts. *Scientometrics*, 125(2):1197–1212, jul 2020. URL: https://doi.org/10.1007%2Fs11192-020-03614-2, doi:10.1007/s11192-020-03614-2.

[24] Google. Cloud Computing Services — Google Cloud. https://cloud.google.com/. Accessed on April 17, 2023.

[25] Google. Google Data Centers. https://www.google.co.uk/about/datacenters/. Accessed on April 17, 2023.

[26] Google Cloud. Google Cloud Pricing Calculator. https://cloud.google.com/products/calculator#id=60a3dd39-8742-46f9-a844-9918ecbb9b2c. Accessed on April 17, 2023.

[27] Braden Hancock, Paroma Varma, Stephanie Wang, Martin Bringmann, Percy Liang, and Christopher Ré. Training classifiers with natural language explanations. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1884–1895, Melbourne, Australia, July 2018. Association for Computational Linguistics. URL: https://aclanthology.org/P18-1175, doi:10.18653/v1/P18-1175.

[28] Mareike Hartmann and Daniel Sonntag. A survey on improving nlp models with human explanations. pages 40–47, 01 2022. doi:10.18653/v1/2022.lnls-1.5.

[29] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[30] Lars Holmberg, Paul Davidsson, and Per Linde. A feature space focus in machine teaching. In *2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 1–2, 2020. doi:10.1109/PerComWorkshops48775.2020.9156175.

[31] Hugging Face. cross-encoder/nli-distilroberta-base. https://huggingface.co/cross-encoder/nli-distilroberta-base. Accessed on April 30, 2023.

[32] Hugging Face. huggingface/distilbert-base-uncased-finetuned-mnli. https://huggingface.co/huggingface/distilbert-base-uncased-finetuned-mnli?doi=true. Accessed on April 29, 2023.

[33] Hugging Face. Models - Hugging Face. https://huggingface.co/models. Accessed on April 18, 2023.

[34] Hugging Face. Text Classification. https://huggingface.co/tasks/text-classification. Accessed on April 26, 2023.

[35] IBM. What is natural language processing (NLP)? https://www.ibm.com/uk-en/topics/natural-language-processing. Accessed on April 17, 2023.

[36] Muhammad Imran, Prasenjit Mitra, and Carlos Castillo. Twitter as a lifeline: Human-annotated twitter corpora for nlp of crisis-related messages. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, Paris, France, may 2016. European Language Resources Association (ELRA).

[37] Fortune Business Insights. Machine learning (ML) market size, share &; covid-19 impact analysis, by component (solution, and services), by Enterprise Size (smes, and large enterprises), by deployment (cloud and on-premise), by industry (healthcare, retail, it and Telecommunication, BFSI, Automotive and transportation, advertising and media, manufacturing, and others), and Regional Forecast, 2022-2029. Technical report, Fortune Business Insights, Mar 2022. URL: https://www.fortunebusinessinsights.com/machine-learning-market-102226.

[38] Iryna Sydorenko. The Potential of Machine Learning in the Stock Market. https://labelyourdata.com/articles/stock-market-and-machine-learning. Accessed on April 16, 2023.

[39] Jason Brownlee. A Gentle Introduction to Transfer Learning for Deep Learning. https://machinelearningmastery.com/transfer-learning-for-deep-learning/. Accessed on April 18, 2023.

[40] Mohd Javaid, Abid Haleem, Ravi Pratap Singh, Rajiv Suman, and Shanay Rab. Significance of machine learning in healthcare: Features, pillars and applications. *International Journal of Intelligent Networks*, 3:58–73, 2022. URL: https://www.sciencedirect.com/science/article/pii/S2666603022000069, doi:https://doi.org/10.1016/j.ijin.2022.05.002.

[41] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. TinyBERT: Distilling BERT for natural language understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4163–4174, Online, November 2020. Association for Computational Linguistics. URL: https://aclanthology.org/2020.findings-emnlp.372, doi:10.18653/v1/2020.findings-emnlp.372.

[42] Daniel Jurafsky and James H. Martin. Speech and language processing, 3rd ed. draft. unpublished.

[43] Sanaa Kaddoura, Ganesh Chandrasekaran, Daniela Popescu, and Jude Duraisamy. A systematic literature review on spam content detection and classification. *PeerJ Computer Science*, 8:e830, 01 2022. doi:10.7717/peerj-cs.830.

[44] Keith D. Foote. A Brief History of Deep Learning. https://www.dataversity.net/brief-history-deep-learning/. Accessed on April 26, 2023.

[45] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *CoRR*, abs/1609.04836, 2016. URL: http://arxiv.org/abs/1609.04836, arXiv:1609.04836.

[46] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. *CoRR*, abs/1909.11942, 2019. URL: http://arxiv.org/abs/1909.11942, arXiv:1909.11942.

[47] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *CoRR*, abs/1910.13461, 2019. URL: http://arxiv.org/abs/1910.13461, arXiv:1910.13461.

[48] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019. URL: http://arxiv.org/abs/1907.11692, arXiv:1907.11692.

[49] Christopher D. Manning and Bill MacCartney. Natural language inference. 2009.

[50] Microsoft. Cloud Computing Services — Microsoft Azure. https://azure.microsoft.com/en-gb/. Accessed on April 17, 2023.

[51] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Neural and Information Processing System (NIPS)*, 2013. URL: https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf.

[52] Mirantha Jayathilaka. 25 NLP tasks at a glance. https://medium.com/@miranthaj/25-nlp-tasks-at-a-glance-52e3fdff32e2. Accessed on April 17, 2023.

[53] Eduardo Mosqueira-Rey, Elena Hernández-Pereira, David Alonso-Ríos, José Bobes-Bascarán, and Ángel Fernández-Leal. Human-in-the-loop machine learning: a state of the art. *Artif. Intell. Rev.*, 56(4):3005–3054, 2023. doi:10.1007/s10462-022-10246-w.

[54] Dhiraj Murthy and Scott A. Longwell. TWITTER AND DISASTERS. *Information, Communication & Society*, 16(6):837–855, jun 2012. URL: https://doi.org/10.1080%2F1369118x.2012.696123, doi:10.1080/1369118x.2012.696123.

[55] Shikhar Murty, Pang Wei Koh, and Percy Liang. Expbert: Representation engineering with natural language explanations. *CoRR*, abs/2005.01932, 2020. URL: https://arxiv.org/abs/2005.01932, arXiv:2005.01932.

[56] Naveen Joshi. Goldman Sachs has replaced 99% of its traders with robots. Are robots now taking over the white-collar jobs? https://www.allerin.com/blog/goldman-sachs-has-replaced-99-of-its-traders-with-robots-are-robots-now-taking-over\-the-white-collar-jobs. Accessed on April 16, 2023.

[57] NumPy. NumPy. https://numpy.org/. Accessed on April 29, 2023.

[58] Emilio Soria Olivas, Jose David Martin Guerrero, Marcelino Martinez Sober, Jose Rafael Magdalena Benedito, and Antonio Jose Serrano Lopez. *Handbook Of Research On Machine Learning Applications and Trends: Algorithms, Methods and Techniques - 2 Volumes*. Information Science Reference - Imprint of: IGI Publishing, Hershey, PA, 2009.

[59] Pandas. pandas - Python Data Analysis Library. https://pandas.pydata.org/. Accessed on April 29, 2023.

[60] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL: http://www.aclweb.org/anthology/D14-1162.

[61] Matthew E. Peters, Sebastian Ruder, and Noah A. Smith. To tune or not to tune? adapting pretrained representations to diverse tasks. In Isabelle Augenstein, Spandana Gella, Sebastian Ruder, Katharina Kann, Burcu Can, Johannes Welbl, Alexis Conneau, Xiang Ren, and Marek Rei, editors, *Proceedings of the 4th Workshop on Representation Learning for NLP, RepL4NLP@ACL 2019, Florence, Italy, August 2, 2019*, pages 7–14. Association for Computational Linguistics, 2019. doi:10.18653/v1/w19-4302.

[62] Dimitrios Pilitsis. Using explanations to improve social media text classifiers. Master's thesis, Department of Computer Science, University of Bristol, Bristol, UK, May 2022.

[63] Dimitrios Pilitsis. year4_thesis. https://github.com/Dimitrios-Pilitsis/year4_thesis, 2022. Accessed on April 29, 2023.

[64] Prakhar Ganesh. Knowledge Distillation : Simplified. https://towardsdatascience.com/knowledge-distillation-simplified-dd4973dbc764. Accessed on April 27, 2023.

[65] Raffaele Pugliese, Stefano Regondi, and Riccardo Marini. Machine learning-based approach: global trends, research directions, and regulatory standpoints. *Data Science and Management*, 4:19–29, 2021. URL: https://www.sciencedirect.com/science/article/pii/S2666764921000485, doi:https://doi.org/10.1016/j.dsm.2021.12.002.

[66] PyTorch. CROSSENTROPYLOSS. https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html. Accessed on April 30, 2023.

[67] PyTorch. DATASETS & DATALOADERS. https://pytorch.org/tutorials/beginner/basics/data_tutorial.html. Accessed on April 30, 2023.

[68] PyTorch. PyTorch. https://pytorch.org/. Accessed on April 29, 2023.

[69] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[70] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[71] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ Questions for Machine Comprehension of Text. *arXiv e-prints*, page arXiv:1606.05250, 2016. arXiv:1606.05250.

[72] Jishnu Ray Chowdhury, Cornelia Caragea, and Doina Caragea. Cross-lingual disaster-related multi-label tweet classification with manifold mixup. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: Student Research Workshop*, pages 292–298, Online, July 2020. Association for Computational Linguistics. URL: https://aclanthology.org/2020.acl-srw.39, doi:10.18653/v1/2020.acl-srw.39.

[73] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108, 2019. URL: http://arxiv.org/abs/1910.01108, arXiv:1910.01108.

[74] Sarah Hansen. Why Layoffs Can Actually Lift a Company's Stock Price. https://www.newsobserver.com/money/tech-layoffs-affect-stock-prices/. Accessed on April 16, 2023.

[75] scikit-learn. f1_score. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html. Accessed on May 01, 2023.

[76] scikit-learn. GridSearchCV. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html. Accessed on May 01, 2023.

[77] scikit-learn. scikit-learn: machine learning in Python. https://scikit-learn.org/stable/index.html. Accessed on April 29, 2023.

[78] Jaime Sevilla, Lennart Heim, Anson Ho, Tamay Besiroglu, Marius Hobbhahn, and Pablo Villalobos. Compute trends across three eras of machine learning. In *2022 International Joint Conference on Neural Networks (IJCNN)*. IEEE, jul 2022. URL: https://doi.org/10.1109%2Fijcnn55064.2022.9891914, doi:10.1109/ijcnn55064.2022.9891914.

[79] Shashank Srivastava, Igor Labutov, and Tom Mitchell. Joint concept learning and semantic parsing from natural language explanations. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1527–1536, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. URL: https://aclanthology.org/D17-1161, doi:10.18653/v1/D17-1161.

[80] Subrat Patnaik and Ryan Vlastelica. Big Tech's Job Cuts Spur Rallies Even as an Economic Slowdown Looms. https://www.bloomberg.com/news/articles/2023-01-25/big-tech-s-job-cuts-spur-rallies-even-as-economic-slowdown-looms? Accessed on April 16, 2023.

[81] Suleiman Khan. BERT, RoBERTa, DistilBERT, XLNet — which one to use? https://towardsdatascience.com/bert-roberta-distilbert-xlnet-which-one-to-use-3d5ab82ba5f8. Accessed on April 27, 2023.

[82] Wilson L. Taylor. "cloze procedure": A new tool for measuring readability. *Journalism Quarterly*, 30(4):415–433, 1953. arXiv:https://doi.org/10.1177/107769905303000401, doi:10.1177/107769905303000401.

[83] TensorFlow. TensorBoard: TensorFlow's visualization toolkit . https://www.tensorflow.org/tensorboard. Accessed on April 30, 2023.

[84] TensorFlow. TensorFlow. https://www.tensorflow.org/. Accessed on April 29, 2023.

[85] TensorFlow. Transfer learning and fine-tuning. https://www.tensorflow.org/tutorials/images/transfer_learning. Accessed on April 18, 2023.

[86] Toloka. Powering AI with human insight - Toloka AI. https://toloka.ai/. Accessed on April 17, 2023.

[87] Philipp Tschandl, Noel Codella, Bengü Nisa Akay, Giuseppe Argenziano, Ralph P Braun, Horacio Cabo, David Gutman, Allan Halpern, Brian Helba, Rainer Hofmann-Wellenhof, Aimilios Lallas, Jan Lapins, Caterina Longo, Josep Malvehy, Michael A Marchetti, Ashfaq Marghoob, Scott Menzies, Amanda Oakley, John Paoli, Susana Puig, Christoph Rinner, Cliff Rosendahl, Alon Scope, Christoph Sinz, H Peter Soyer, Luc Thomas, Iris Zalaudek, and Harald Kittler. Comparison of the accuracy of human readers versus machine-learning algorithms for pigmented skin lesion classification: an open, web-based, international, diagnostic study. *The Lancet Oncology*, 20(7):938–947, jul 2019. URL: https://doi.org/10.1016%2Fs1470-2045%2819%2930333-x, doi:10.1016/s1470-2045(19)30333-x.

[88] United Nations. United Nations Office for the Coordination of Humanitarian Affairs. https://www.unocha.org/. Accessed on April 16, 2023.

[89] University of Bristol. Advanced Computing Research Centre. http://www.bristol.ac.uk/acrc/. Accessed on April 30, 2023.

[90] Dipam Vasani. How do pre-trained models work? https://towardsdatascience.com/how-do-pretrained-models-work-11fe2f64eaa2. Accessed on April 18, 2023.

[91] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL: http://arxiv.org/abs/1706.03762, arXiv:1706.03762.

[92] Victoria Nava. The Role of Machine Learning in Big Data . https://www.qubole.com/blog/big-data-machine-learning. Accessed on April 16, 2023.

[93] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. 2019. In the Proceedings of ICLR.

[94] Sarah Wiegreffe and Ana Marasovic. Teach me to explain: A review of datasets for explainable NLP. *CoRR*, abs/2102.12060, 2021. URL: https://arxiv.org/abs/2102.12060, arXiv:2102.12060.

[95] Adina Williams, Nikita Nangia, and Samuel Bowman. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122. Association for Computational Linguistics, 2018. URL: http://aclweb.org/anthology/N18-1101.

[96] Arne Wolfewicz. Human-in-the-Loop in Machine Learning: What is it and How Does it Work? https://levity.ai/blog/human-in-the-loop. Accessed on April 28, 2023.

[97] Xingjiao Wu, Luwei Xiao, Yixuan Sun, Junhang Zhang, Tianlong Ma, and Liang He. A survey of human-in-the-loop for machine learning. *Future Generation Computer Systems*, 135:364–381, 2022. URL: https://www.sciencedirect.com/science/article/pii/S0167739X22001790, doi:https://doi.org/10.1016/j.future.2022.05.014.

[98] Barry J. Wythoff. Backpropagation neural networks: A tutorial. *Chemometrics and Intelligent Laboratory Systems*, 18(2):115–155, 1993. URL: https://www.sciencedirect.com/science/article/pii/016974399380052J, doi:https://doi.org/10.1016/0169-7439(93)80052-J.

[99] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3320–3328, 2014. URL: https://proceedings.neurips.cc/paper/2014/hash/375c71349b295fbe2dcdca9206f20a06-Abstract.html.

Appendix A

Handling the large amount of data

As introduced in Section 3.2.2, due to the size of the expanded dataset that was formed from concatenating each tweet with each explanation, for ExpBERT and BERT-IE, subsets were created to pass the datapoints through the pre-trained model. This was necessary due to the memory constraints and time limit for jobs running on the supercomputer that I was using for this project.

The expanded dataset had a total of 616,212 datapoints ($17,117 * 36$) which I split into nine subsets. The first eight subsets were split into 72,000 datapoints, and the final subset had 40,212 datapoints. Table A.1 shows which tweets were contained in each subset, and which datapoints these were in the expanded dataset.

Subset No.	**Tweets**	**Datapoints in the expanded dataset**
1	0 - 1,999	0 - 71,999
2	2,000 - 3,999	72,000 - 143,999
3	4,000 - 5,999	144,000 - 215,999
4	6,000 - 7,999	216,000 - 287,999
5	8,000 - 9,999	288,000 - 359,999
6	10,000 - 11,999	360,000 - 431,999
7	12,000 - 13,999	432,000 - 503,999
8	14,000 - 15,999	504,000 - 575,999
9	16,000 - 17,116	576,000 - 616,211

Table A.1: Table showing which tweets were contained in each subset, and which datapoints these were in the expanded dataset. It is important to note that the ninth subset was of a reduced size due to it being the last subset.

Each subset was passed through the pre-trained model, then reshaped to form the complete representation for each tweet as described in Section 3.1.3. For ExpBERT, the reshaped embeddings of [2000, 27648] ([1117, 27648] for the ninth subset) were concatenated to form a combined embedding of size [17117, 27648]. For BERT-IE the reshaped embeddings of size [2000, 876] (([1117, 876] for the ninth subset) were concatenated to form a combined embedding of size [17117, 876]. The embeddings were concatenated using the `concatenate_embeddings.py` file. These combined embeddings were then passed into the model head as normal.

Overall, this process had no impact on the performance of either ExpBERT or BERT-IE, as the entire dataset was still being used for classification, only split up for the creation of the embeddings.