Claim Detection with BERT on Semi-Structured Text

6CCS3PRJ Final Year

William Baker

1904215

Supervised By: Dr Odinaldo Rodrigues

Special thanks to Dr Odinaldo Rodrigues for his masterful guidance and advice

April 2, 2022

Abstract

Argumentation Mining (AM) enables us to reason with natural language; effective AM depends on our ability to accurately detect claims in text. In structured texts, the claim detection performance of the state of the art model, 'BERT', is well understood. But much less is understood when BERT is applied to less structured text, such as social media, which is more indicative of "real world *natural* language".

We compare BERT's performance in classical structured texts with that of semi-structured texts. Then study the performance improvements obtained by pre-training BERT in the same domain before training BERT for claim detection.

Overall, we have found that BERT performs well on semi-structured text, but pre-training in the domain is not necessary to obtain good performance. This work can be continued through the use of classical argumentation mechanisms to relate claims to one another for effective argumentation mining from social media.

Originality Avowal

I verify that I am the sole author of this report, except where explicitly stated to the contrary. I grant the right to King's College London to make paper and electronic copies of the submitted work for purposes of marking, plagiarism detection and archival, and to upload a copy of the work to Turnitin or another trusted plagiarism detection service. I confirm this report does not exceed 25,000 words.

William Baker

April 2, 2022

Contents

Chapter 1

Introduction

Claim detection entails the identification of arguable statements in text, and so is the natural first step in any study of arguments. As such, it is vital to understand how our ability to accurately identify claims varies between each domain in which we might apply the technology. Current techniques and their capabilities are well understood for classical argumentative text, such as debate forums containing *structured text.* But much less is understood when the technology is applied to less structured text, such as social media. A complete understanding of claim detection in semi-structured text will allow for far more effective and useful argumentation mining. Argumentation mining gathers evidence supporting/attacking claims ergo, adding structure, enabling inference and allowing us to reason with natural language.

This report investigates the capabilities of BERT to detect claims in "real-world *natural* language", such as social media where the language used is much more chaotic than anything seen in formal debates. In social media vague claims are frequently made; this makes the task of identifying whether a social media post makes a claim very challenging. So far most research avoids this problem by solely applying their models to structured text such as debate forums. Such debate forums host a very small community of users (10^6 total Kialo posts) compared to social media (10^{11} tweets annually). As such, debate forums naturally lack social media's community, subject, linguistic and emotional diversity.

Current state of the art methods fine-tune a pre-trained BERT model to identify claims. BERT is a transformer model that targets language understanding, whose training consists of 2 parts:

1. unsupervised pre-training combining *Masked LM* (predicting missing words in sentences) and *NSP* (Next Sentence Prediction)

2. supervised fine-tuning for an arbitrary decoding task

The pre-training step embeds the model with an understanding of the language and vocabulary of the domain, but as such assumes that the application domain is the same as the pre-training domain. Existing methods avoid pre-training BERT in the application domain, thus when applied to social media data, any vocabulary not seen during pre-training, such as 'singlish' and emoticons is removed during pre-possessing; these models are also embedded with less understanding of the language used on social media.

We investigate whether pre-training BERT in the application domain can increase performance on both classical and real-world datasets, posing novel techniques to encode features unique to social media datasets, such as emoticons and hashtags. This work can then be continued through the use of classical argumentation mechanisms to relate claims to one another for effective argumentation mining from social media.

1.1 Motivation

Argumentation mining is still in its infancy, having only began to grow in popularity in 2014 (figure 1.1), but has amazing potential to help us make sense of the vast amount of natural language uploaded to internet platforms every day. Later tasks in argumentation mining such as relation identification and stance detection depend on accurate claim detection. As such, it is vital to perform this task as best as possible to aid all development in the field. We hope to help lay the foundation to allow argumentation mining to be applied to a broader variety of applications, encouraging later research to investigate it's utility to reason with the web.

The ability to apply argumentation mining to a variety of applications will only grow in importance as we focus more on XAI (explainable AI) in the face of upcoming EU regulations [1] [2] [3].

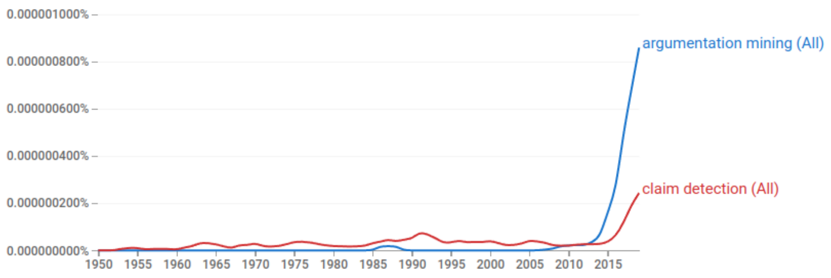


Figure 1.1: Argumentation Mining and Claim Detection usage over time

1.2 Structure

This report is divided into several chapters, and a comprehensive study of the field of argumentation mining is conducted in the background (Chapter 2). The research most relevant to this report is reviewed in more detail in section 2.6.3, we recommend that advanced readers skip to this section. The goals and requirements are then detailed in the specification (Chapter 3), while the design (Chapter 4) details how these requirements will be met. The implementation (Chapter 5) covers the details of developing the designed algorithms with python. Finally, we publish our results in chapter 6 and reflect on the report in the evaluation in chapter 7.

Chapter 2

Background & Review

In the following, we will begin our exploration into the history of argumentation mining, various associated methods and their benefits and drawbacks. The key reading relevant to the investigation can be found in section 2.6. The full history of the research in the field has also been detailed in [4], and was used to guide much of the research in the following sections.

2.1 Argumentation Mining

Argumentation mining is the art of identifying components of language and relating them to support or attack one another. With the goal of using the resultant structure to reason more effectively with natural language. Over time, a vast amount of research has been conducted towards integrating argumentation theory into intelligent systems. The field has grown from pure logic, to deductive reasoning, defeasible reasoning, opinion mining and today argumentation mining. Argumentation mining most easily applies to debates, thanks to the exchange of well-formed arguments, leaving little uncertainty as to whether one statement attacks or supports another statement, or is simply unrelated. To aid in the understanding of the following sections, here are some sample sentences extracted from the dataset used by [5]:

1. *We also could save endangered species, even revive extinct ones*
2. *Cloning, they claim, is a better technology for harvesting stem cells because it could mitigate problems of tissue rejection, since stem cells are cloned from the patient's own body.*
3. *Reproductive cloning would entail dehumanizing experimentation "Just Say No To Human Cloning"*

4. *We Should Raise the Minimum Wage*

5. *Some make common sense arguments while others use enough fancy math to dazzle any economist*

The first 4 sentences each make claims, the first and second support cloning, while the third attacks cloning. The fourth is an unrelated claim. While the fifth makes no claim at all. These relations can be used to form a graph, where each node is a claim, and each edge relates 2 claims in a *for* relation (+ green) or *against* relation (- red).

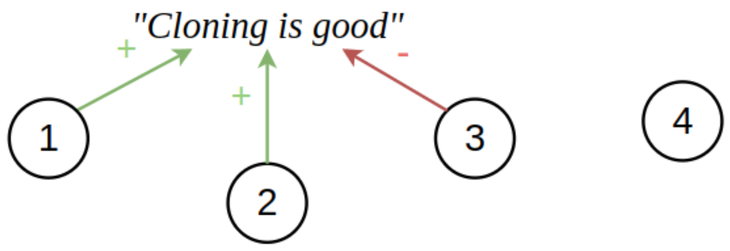


Figure 2.1: Argumentation Graph

Now that we have established what argumentation mining is, in the following sections we will discuss the history of the field, its origin, and current state. Although this report only goes as far as to study claim detection in greater detail, it is important to understand the general state of the subject. Never the less, feel free to skip to section 2.6 for the research that is most relevant to this report.

2.2 Deductive Reasoning

Deductive reasoning, also called deductive logic, is the purest form of reasoning which draws conclusions from a set of premises using logical rules. For example, applying the logical rule *modus ponus* to a pair of symbols P and Q: $P \rightarrow Q$ denotes that Q can be concluded from P. It is upon deductive reasoning on which logic is based, but it is limited in that it only deals with absolutes, accordingly, defeasible reasoning was developed...

2.3 Defeasible Reasoning

Defeasible reasoning studies the case where arguments supporting a claim are 'tentative', meaning that although the argument supports/attacks the claim, it's basis is not deductively valid (if all premises are true, the conclusion is also true). In short, arguments can be invalidated by additional information, for example [6]:

- Claim: “The sign is red”
- Tentative Argument: “The sign looks red”
- Overriding Argument: “The sign is being illuminated by red lights, but is in fact white”

Pollok [6] provided the theory for reasoning with such arguments which had, up until that point, been overlooked in the literature. Hence, the theory for both defeasible and deductive reasoning was laid out for future research.

This foundation sparked a frenzy of implementation with Dung [7] first to implement Pollock’s [6] theory with logical programming. Next, Krause [8] published another implementation but started to address the uncertainty in such reasoning, which was named *justification.* For example [8]:

- arg1: A (0.8)
- arg2: B(0.8)
- arg3: C(0.8)
- $r1 : A \& B \supset D(1.0)$
- $r2 : A \& C \supset D(1.0)$
- $p(D) = p(A \wedge B \wedge r1) \vee p(A \wedge C \wedge r2)$
  $= p(A \wedge B \wedge r1) + p(A \wedge C \wedge r2) - p(A \wedge B \wedge r1 \wedge C \wedge r2)$
  $= 0.64 + 0.64 - 0.512 = 0.768$

In the above we have a series of arguments, each with an uncertainty, and a pair of rules that are definite; finally we calculate the uncertainty in D after applying these rules to the arguments, which evaluates to 0.768. (Note, the original paper has a typographical error which has been corrected here). Pollock then fulfilled the promise in their original paper to follow up the theory with an implementation [9] which similarly explored *justification.* [10] extended the ability to negotiate using argumentation into the field of multiagent systems, which was concluded on the millennia by [11] with agents able to maintain natural debates, arguing over a topic.

2.4 Toulmin's Model

Future progress towards argumentation mining largely depends on the formalisms established by Toulmin's model [12]. Toulmin's model breaks down the anatomy of arguments, as such, also finds widespread adoption in the structuring of persuasive writing. It describes the *layout of arguments* to comprise of 6 key components:

- **claim** - pure argument whose merit must be established
- **grounds** - evidence to justify claim
- **warrant** - connects grounds to claims
- **backing** - adds credibility to warrant
- **qualifier** - "indicate the strength conferred by the warrant on this step"
- **rebuttal** - "circumstances in which the general authority of the warrant would have to be set aside".

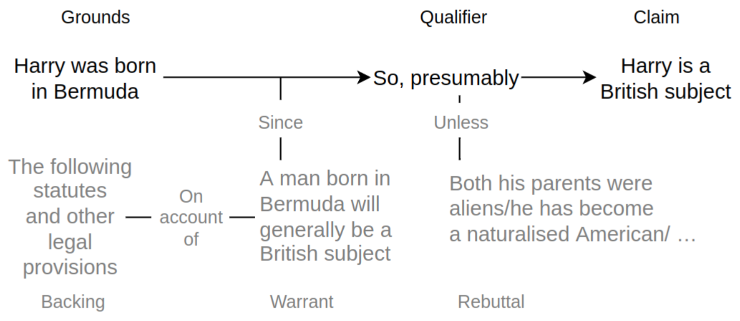

Grounds
Qualifier
Claim
Harry was born in Bermuda
So, presumably
Harry is a British subject
Since
Unless
The following statutes and other legal provisions
On account of
A man born in Bermuda will generally be a British subject
Both his parents were aliens/he has become a naturalised American/ ...
Backing
Warrant
Rebuttal

Figure 2.2: Toulmin's Model

Implementing Toulmin's model in reality presents a yet unsolved problem, due to the complexity of interpreting natural language. Instead, the problem has been decomposed into 3 key components.

- **Claim Detection** - Does a given sentence make a claim
- **Relation Identification** - Does a given sentence relate to a claim (give grounds to the claim)
- **Stance Detection** - Identifies the nature of the relationship between a claim and its grounds, whether it supports or attacks

Each will be addressed later. However, each first depends on the research that was developed for the field of opinion mining...

2.5 Opinion Mining

The next logical step toward argumentation mining is opinion mining (sentiment analysis), which simply analyses the sentiment of text. This is a great simplification vs argumentation mining as the structure of the text does not need to be considered. There are two schools of thought: **Lexicographic** and **Machine Learning (ML)**. Lexicographic approaches were the first to be developed [13]. They function by constructing a dictionary of positive and negative words ("lexicons"), the overall sentiment is then the sum of the respective scores of the component words. For instance, Twitter themselves suggest [14] a positive lexicon of *"happy, exciting, excited , favorite, fav, amazing, lovely, incredible"* and negative lexicon of *"horrible, worst, sucks, bad, disappointing"* to retrieve emotional tweets (although it is reasonable to assume they use more advanced techniques internally). Lexicons can be extended to account for negation and intensification and the like. The most notable lexicons are AFINN [15], (AFINN applied to twitter: [16]), VADER [16] and NRC [17]. It is worth noting that Opinion Mining has long been applied to social media with both NRC and VADER having 20k and 4.2k tweets included in their training corpus, respectively. Generally, VADER performs the best, closely followed by AFINN according to benchmarks [18]. NRC also provides granular emotional scoring over a set of emotions: anger, anticipation, disgust, fear, joy, sadness, surprise, and trust, although unsurprisingly performs worse at this harder task with an F1 score of 0.42 compared to AFINN and VADER with 0.84F1 on Twitter datasets [18]. Lexicons have also been developed for social media to assess specific topics such as [19] which uses lexicons to assess student learning experiences through student's posts to Twitter.

ML approaches cover a more diverse range of natural language models. This unlocks far superior accuracy compared to the lexicographic approach. Today, opinion mining is a largely solved problem for both structured and semi-structured text, thanks to ML models such as XLNet achieving over 95% accuracy on structured datasets [20, 21] and unstructured datasets such as SST [21] and IMDb [22]. A brief summary of the current state of the art has been detailed in section 2.5.

2.5.1 Opinion Mining with Emojis

One of the many contributors to the discrepancy between traditional argumentation datasets and Twitter is the frequency of emoticons within text. Emoticons are icons that can be included in text, many of which are facial expressions, but, overall, they are used to convey emotion. This has attracted lots of interest within the field of sentiment analysis as emoticons have been

Dataset	Binary or Fine	Accuracy	Model
IMDb [20]	B	96.21 %	XLNet (Autoregressive Pre-trained BERT) [23]
IMDb [20]	B	95.79 %	BERT (Transformer) [24]
SST [21]	B	96.8 %	XLNet (Autoregressive Pre-trained BERT) [23]
SST [21]	F	56.2 %	BCN+BiLSTM+CoVe [25]
SemEval [26]	F	0.685 (F1)	LSTMs+CNNs ensemble [27]
SemEval [26]	F	0.677 (F1)	Deep Bi-LSTM+attention [28]

Table 2.1: Opinion Mining Performance

shown to contribute to sentiment classification, also showing promise when used in clustering algorithms [29] and so clearly contain useful information. Most recently, [30] formally assessed emoji's contribution when using state-of-the-art models, and found that *"Emoji information is useful in sentiment analysis"*. The largest investigation to date was conducted by MiT, dubbed "DeepMoji" [31] which trained a bidirectional LSTM with attention on a corpus of 1.2 billion tweets, seeing performance up to F1=0.75 when transferred to sarcasm datasets. For more information, a review of the field has been conducted [32].

Thus far, there has been very little integration of emoji's within the field of argumentation. Furthermore, many previous researchers have restricted themselves to claim, ground & warrant component identification, citing shorter text as *jargon* to be of limited use when constructing arguments. If we applied this philosophy to Twitter, a lot of Twitter data would be discarded as most tweets are just a few words long (21% of the tweets gathered contained 5 words or less), further non-argumentative and thus unsupported tweets (unsupported claims have no premise, so are called *enthymemes*) would also be discarded. To avoid this, we could follow Toulmin's rubric more closely, by treating emoticons, jargon and otherwise non-argumentative tweets as simple backing [12] w.r.t. their local neighbours (parent tweets and popular replies). Especially given that a large proportion of tweets contain emoticons (23% of the tweets we mined contained emojis). It would be an interesting candidate for future research on argumentation mining with social media.

A further complication of using emoji's to aid opinion mining and later argumentation mining, are *sequence modifiers* [33]. Whereby multiple emoji-reserved Unicode characters can be strung together in specific combinations to form new emojis, such as

👨 + 👩 + 👧 = 👨‍👩‍👧

2.6 Claim Detection

Returning to Toulmin's model we can now focus on the first component of argumentation mining, claim detection. Claim detection entails the identification of arguable statements in text. The definition in the Cambridge dictionary encapsulates this as *"a statement that something is true or is a fact, although other people might not believe i"* [34]. The identification of argumentative claims comes naturally to humans as claims are frequently made during conversations and other discourse, but this task presents a real challenge for machines, due to the lack of a formal definition of when a sentence becomes a claim, instead we must base our classification purely on the empirical definition cited previously [34].

Many classical approaches to the problem utilise *discourse markers*, which are lexical features that are used to relate segments of discourse. [35] gives a good definition of discourse markers, identifying 2 key uses. Firstly, to entail a claim, the example they use: *"However, staying down is pointless from a pedagogic point of view."*, where **However** is the discourse marker denoting the beginning of a claim. Secondly, to mark elaboration: *"the students get frustrated, their performance generally does not improve. Also, the point of repeating all courses because of only one or two bad grades is arguable"* here **Also** follows the claim and precedes further elaboration on the claim. Discourse markers have seen widespread use in the field in both supervised [36, 37, 38] and unsupervised approaches [39, 40, 41] as they consistently improve performance due to their ability to help narrow down possible locations of claims. However, it is worth noting that such indicators are sparsely found in semi-structured text.

2.6.1 unsupervised claim detection

The current state of the art method for the unsupervised detection of claims is a joint model [39] which combines the techniques in [40] and [41]. It has 3 components:

1. position model

2. indicator model

3. text rank model

The position model scores sentences based on their position in each paragraph. Sentences at the beginning/end of the first/last paragraph are granted the highest score, sentences at the beginning/end of others are given a lower score, otherwise no score is added.

$$score_p(s_i) = \begin{cases} 2 & s_i \text{ first/last sentence in paragraph in the first/last paragraph} \\ 1 & s_i \text{ first/last sentence in OTHER paragraphs} \\ 0 & \text{else} \end{cases}$$

s_i - i^{th} sentence in the document

The indicator model increases the score for any sentences that contain *discourse markers*. It is formally defined as:

$$score_I(s_i) = \begin{cases} 1 & s_i \text{ contains discourse marker in the list } \mathcal{L} \\ 0 & otherwise \end{cases} \tag{2.1}$$

Where $\mathcal{L}$ is a list of 82 discourse markers: *accordingly, as a result, consequently, conclude that, clearly, demonstrates that, entails, follows that, hence...*

The text rank model scores each sentence based on the frequency with which its words appear in the rest of the document. The implementation used is based on [41]. This boils down to a TF-IDF (term frequency inverse document frequency) score, which simply scores each term in a sentence based on the inverse of its frequency in the document, hence common words are given lower scores so are less likely to be included in the claim lexicon. The TF-IDF score is defined as:

$$\text{TF-IDF}(t, d, D) = tf(t, d) \times idf(t, D) = \frac{f_d(t)}{\max\limits_{w \in d} f_d(w)} \times \ln\left(\frac{|D|}{|\{d \in D : t \in d\}|}\right)$$

$$\begin{aligned} f_d(t) &- \text{frequency of term in document d} \\ d &- document \\ D &- \text{corpus of documents} \\ score_T(s_i) &= \sum_{t \in s_i} \text{TF-IDF}(t, d, D) \end{aligned} \tag{2.2}$$

These 3 metrics are then combined to form the joint model, which scores each sentence in the corpus, with higher scores indicating a higher chance that the sentence makes a claim:

$$score(s_i) = \alpha \cdot score_T(s_i) + score_T(s_i) \cdot [\beta \cdot score_P(s_i) + (1 - \alpha - \beta) \cdot score_I(s_i)]$$

Where α and β are parameters that must be tuned depending on the genre.

The above model achieves an F1-score of 0.56 on a classical dataset consisting of persuasive essays and web discourse. This is very admirable considering that the method does not require topic information, which is required by most other methods [40, 42, 5]. However, overall this method is fairly crude, as it does not attempt to understand the text, it merely uses features that are shown to have strong statistical correlation with claims in structured text. As such, this model applies poorly to unstructured text. For which both the positional score and text rank score mean very little. This lack of structure is supported by [43], who attempted to fit an argumentation model to Twitter, but *"we (almost) never find such a kind of complete structure of the arguments"*.

2.6.2 Semi-supervised claim detection

A novel approach to semi-supervised claim detection was proposed in [44]. They developed a *topic independent claim related lexicon (TICRLex)* by constructing lexicons (a set of terms) found in claims in tweets about a given topic. They combine the best terms from each topic to form the TICRLex. In brief, the terms for the lexicon were selected based on:

$$Claim_{TI}(t) = \sum_{k \in K} \frac{IG_k(C,t) \cdot H(t|K)}{TN_k}$$

Where $IG_k(C,t)$ is the information gain about a claim C of term t given a topic k, $H(t|K)$ is the distribution of t under a set of topics K, the more evenly distributed a term is across the topics the greater H. And finally, TN_k is the number of tweets on a topic k.

Next, they filter these topic-independent lexicons for each topic based on how positively each term relates to its claim, those with the strongest relations to claims under a given category form the *Topic-Dependent Claim-Oriented Lexicon (TDCOLex)*

$$Claim_{TD}(t) = \sum_{w_i \in posLex} \frac{Claim_{TI}(w_i)^+ \cdot CoT(w_i,t)}{TN_t}$$

Where $CoT(w_i,t)$ is the co-occurrence frequency of a term t in a topic-related tweet set TS with the term w_i in **poslex** (**poslex** is a pre-trained lexicon of positive terms). TN_t is the number of tweets containing term t and finally $Claim_{TD}(t)$ is a term t's topic-dependent claim-oriented score (how positively a term relates to the claims of this topic).

2.6.3 Supervised claim detection

For broader applications, supervised models have had much greater success. Such models have followed the progression of the field of deep learning, starting initially with linear regression models, such as that used in [43], which performed admirably and achieved an F1-score of 0.56. They also continued their research into relation identification with limited success (F1-score 0.18). Their findings are extremely applicable to this report, unfortunately they lack useful detail regarding their implementation beyond them finding a logistic regression model to perform the best using 3 fold cross-validation to validate their findings. They also used the DART [45] data set throughout their investigation; unfortunately, this data set is not publicly available. [43] served as the first demonstration that tweets could be reliably mined using supervised models for claim detection tasks.

A very similar investigation was conducted in [46], again identifying claims in Twitter using a linear regression model. They report an overall F1-score of 0.78, although under further investigation a very biased dataset was used, a subset of the DART [45] dataset with 1459 argumentative tweets but only 428 non-argumentative tweets, further they also reported the micro F1-scores for argumentative and non-argumentative tweets, being 0.85 and 0.52 respectively. The realistic performance on a balanced dataset would be much closer to 0.52, as such little attention has been paid to their findings since [43] outperforms their model.

More recent research has started to focus on deep models, for which there are 2 state of the art approaches. The first approach is to fine tune Google's BERT [47] model, this approach was taken in [5]. They simply downloaded Google's pre-trained BERT model and *fine tuned* it (continuing training on a new dataset, starting from the same weights). They trained the model on both the *UKP* dataset (25492 labelled sentences on controversial topics taken from [48]) and *IBM debater Evidence Sentences* dataset (labelled Wikipedia text on 118 topics from various debate portals [49]). After training, they achieved an average performance of F1=0.71, which is a remarkable improvement compared to prior supervised methods. However, both training datasets consist of well-structured text, where as the majority of supervised methods mentioned previously operate on semi-structured data such as tweets, for this reason the great performance must be taken with some scepticism. In [5] they also researched how unsupervised clustering of tweets using BERT embeddings (the latent space representation of a sample) could be used to aid claim detection performance, but it proved to be detrimental (F1=0.65).

The second state of the art approach uses Graph Neural Networks to detect claims and was investigated in [50]. Graph Neural Networks are rapidly growing in popularity and were applied here to the same datasets as BERT was applied to in [5]. As well as 2 additional smaller datasets extending the model to better differentiate between claims and premises. Various kernels were tested, but overall the best performing kernel achieved just F1=0.61 on the UKP corpus. Little investigation is taken in their paper comparing their results to other papers, it is likely that the reason for this lower score is their lack of embedded understanding of natural language in their model when compared to BERT. As the pre-training phase of BERT embeds an understanding of natural language into the BERT model through exposing the model to 3.3 billion exemplar sentences.

BERT for Claim Detection

Before continuing to the next component of argumentation mining (stance detection, section 2.7), is worth reviewing [5] in more detail. We will then examine the BERT architecture they have used in greater detail, as it is the best architecture to date for claim detection.

[5] fine tunes BERT on 2 structured text datasets to detect claims. The first of which was created by the UKP (Ubiquitous Knowledge Processing) lab, the second as part of IBM's *Project Debater*. Each dataset contains a list of sentences which have been manually annotated as either making a claim or not, in the case of the UKP dataset sentences are also labelled as for/against claims giving 3 labels, although for the majority of the report these are *binarized* into claim/no claim. In the case of the IBM dataset each sentence was scored according the strength of the evidence it provided towards a claim, these were then re-processed by [5] into binary labels denoting the presence of a claim, unfortunately their labelled dataset was not published so we'll only be considering the UKP dataset from now on. Next they tokenized each sentence using the *BERT Tokenizer* from [47], this is a *word piece* tokenizer that extracts subwords from each word to minimise it's vocabulary. This has two effects. First, uncommon conjugations of words will be split into their root and suffix, with a string of `##` denoting the beginning of a suffix. This allows for the meaning of words to be maintained across conjugations. The 2nd effect of this is enabling the recognition of unrecognised words to an extent, for example *arachnophobia* is not present in BERT's default vocabulary of 30k words, so it is split into *'ara', '##ch', '##no', '##phobia'* which can then be input to the model, this is not ideal as it assumes the model is able to extract the meaning of the word from the sum of it's sub words, but is far

better than the alternative of replacing unknown words with a unknown token such as [UNK].

A pre-trained model is then loaded from the repository for [47], this model has been trained on 3.3 billion words from Wikipedia (an online encyclopaedia indexing structured descriptions of terms). The implication of this is that the model has been conditioned to understand the language on Wikipedia and has a vocabulary consisting of the most common words on the site. This is ideal for the training data used in the paper as the UKP dataset consists of "news reports, editorials, blogs, debate forums, and encyclopedia articles" [48] which are all similar in nature to the language used in Wikipedia articles. However this highlights the key issue when applying BERT to other datasets, the pre-training domain must be the same as the domain in which the model is applied. Thanks to this, the work in [5] cannot simply be transferred to a new domain such as Twitter without the model having been pre-trained in the Twitter domain. Thanks to this, our implementation addresses this issue in section 4.6.

Before training can begin, the model is modified to add a classification layer; this adapts the 768 element output vector of the BERT model to the classification labels, in this case 2 outputs are used denoting claim/no claim. In the paper they choose 2 outputs over a binary output as it allows for the use of cross entropy loss, which can be scaled to an arbitrary n-class classification problem, allowing the model to be trained in the 3 class case (no claim/for claim/against claim). However, the paper falls short as it uses a python package called BertForSequenceClassification to automatically add this layer, however it doesn't add a SoftMax activation before the output. The Softmax activation function applies the function $\frac{e^z}{\sum e^z}$ to normalise the output to follow a probability distribution that sums to 1. The use of SoftMax is widely accepted for classification problems [51] [52] and is used in our implementation (section 4.5.4).

Training the pre-trained model is called *fine-tuning*, as much of the model will remain relatively unchanged especially in earlier layers. Since the vocabulary and input encoding remain the same as it was during pre-training, only the classification problem has changed. As such the understanding of natural language embedded in the model during the pre-training phase (which we'll delve into in more detail later, section 2.6.3) can be used for the new problem.

BERT Architecture

BERT is the model of choice, both in our investigation and in the paper that inspired it [5]. In this section we summarise BERT, address how BERT was used in [5] and why the same technique would not be valid for Twitter based claim detection as seen in [44, 43, 46].

Firstly, we must define what is meant by a vocabulary and a language under the context of this report. A vocabulary is a set of words used in a language. While a language is a *"method of human communication, consisting of words used in a structured and conventional way and conveyed by speech, writing, or gesture"* as defined by *Oxford Languages.*

BERT [5] was developed by Google in 2019 to aid in natural language understanding tasks. Such as indexing search results and translation between languages. It is a transformer architecture, meaning it uses self attention layers to process sequences of inputs, many of these self attention layers are stacked to add depth to the model and increase it's power (We delve into it's architecture in more detail in 4.5). For now, note that an embedding layer is added to the start of the model, which translates the vocabulary of arbitrary size to a constant size of 768 expected by the self-attention layers. Further, a classification layer is added to the output, which converts the output latent space of the self-attention layers into number of class scores for classification tasks (in reality, any layer(s) can be added to the output, to suit any task, but all applications cited here are classification tasks).

The key principle behind BERT is the ability of models to *transfer learning's* between tasks, meaning that a model can be trained to do one thing using a large dataset and then trained again on a second (similar and often smaller) dataset, and perform better than it would have, if it were just trained on the second dataset alone. BERT generalises this concept by training a model on 2 general natural language understanding tasks. Namely, Masked LM and NSP (next sentence prediction) where a model is trained to guess the missing word in a sentence or to guess whether once sentence entails another (more detail in sections 4.6 & 4.6). The same corpus of text can be used for both tasks, [47] uses a corpus containing 3.3 billion words from Wikipedia. This is crucial as the embedding layer and the model is kept the same between the two tasks, since they both are designed to improve the same model, as such both expect the same vocabulary of words is passed to the embedding layer. Between tasks only the classification layer changes, which is much cheaper to train than the rest of the model. The result of this is an embedding layer and a list of self-attention layers which form BERT, which are now able to *understand* natural language to a degree. A new classification layer can then be added to this pre-trained model, and can be trained to solve an arbitrary NLP (natural language processing) task. In doing so we are *transferring* the understanding gained in the Masked LM and NSP tasks to the new problem. [5] does exactly this.

However, note that the same embedding layer is used throughout the process, as such the vocabulary used in [5] has to be the same as that which was used in [47]. In order to apply BERT to a new domain we must consider this limitation. There are 2 methods to circumvent this constraint. Firstly, we could construct a new large dataset within the new domain comparable to that used by [47] and train an entirely new model, just as was done by [47] using Masked LM and NSP. As such, the embedding layer will be trained for the new domains vocabulary. Secondly, we could just replace the embedding layer with a new embedding layer and train the model on a dataset from the new domain just as before, but only update the first layer during training. This would train the embedding layer to best adapt to the existing self-attention layers that were trained in [47]. This second method assumes that the language used in the new domain is the same as the old domain, so there is simply a one-to-one mapping from the original vocabulary to the new vocabulary. This is unlikely, for instance, under the Twitter domain (section 2.10.1) has a large disparity in both the vocabulary and the language used when compared to structured text.

2.7 Relation Identification

Once you are able to reliably detect claims, the next component to consider is relation identification, this addresses the previously ignored grounds identification from [12]. Relation identification entails the finding of premises that are associated with a claim. The majority of research combines relation identification and stance detection (identifying the nature of the relation between a claim and it's grounds). As such, the research on relation identification is best split into two categories, macro-analysis (identifying attack/support relations) and micro-analysis (evaluating attributes of the claim: such as reliability, integrity, and cohesion). However, as a whole, the field is much less mature than claim detection due to the difficulty of claim detection and the dependency relation identification has upon strong claim detection. A supervised approach was taken by [43] using Twitter data, but poor results were achieved both using the Excitement Open Platform (F1 = 0.17), and surprisingly, even an LSTM struggled (F1=0.2). [53] enjoyed much greater success, with F1=0.934 over a reasonable dataset of 840 tweets that originally referenced news articles. However, their approach classified tweets into support/attack without classifying whether the tweet is unrelated, instead each training example is known to relate to the claim tweet. Hence, their performance more closely indicates the ability of an LSTM to classify sentiment, a task that currently enjoys 96% accuracy, we covered opinion mining in Opinion Mining.

A model that conducts a microscopic analysis, has been developed by [44], which also investigates the performance over a mixture of explicit and implicit claims. They achieved a mean average precision of 0.50 (on the same dataset as [53]), through using a lexicon approach combined with an SVM. The field has been revisited more recently by [54], applying state-of-the-art transformer models, namely RoBERTa (a variation of BERT that is trained on more data) to a multitude of datasets, achieving F1=0.59, although this is over classical argumentation datasets (debates) rather than Twitter.

Several unsupervised approaches have been taken, such as [55] (builds on [56]) who used OpenIE to extract relation tuples. OpenIE is an information extraction tool designed to assist in reasoning with natural language. Given a sentence, it extracts sets of tuples of 2 words and identifies the relation between these words. Returning to [55], after applying OpenIE they then analysed the sentiment coherence, causal relation, normative relation, and relation chain of relation tuples using probabilistic soft logic (PSL) concluding by classifying the arguments into support/attack/neutral relations. They achieved an F1 score of 0.808 even exceeding many supervised approaches. This improves on it's predecessor [56] which achieved F1=0.73 on the same dataset.

2.8 Stance detection

The final component typically used in argumentation mining is stance detection, which identifies the nature of the relationship between the grounds and the claim. It is frequently extended to assess microscopic aspects of terms such as the quality of claims. Which is indicated by recognising facts, classifying supporting evidence, identifying sources and their reputation, and evaluating the logical reasoning used within the claim. Stance detection is often performed in conjunction with relation identification, as these attributes of a claim greatly assist the identification of accurate relations. For instance, Opinion Mining provides a strong indication of attack vs support relations.

Some examples of a microscopic analysis into the nature of the grounds are as follows: [57] uses evidence classification and identified six categories *"news media accounts; blog post; picture; expert opinion; other types of evidence, and no evidence"*, achieving an F1 score of 0.83 when using this microscopic analysis to classify argumentative relations. [46] used linear regression to classify fact-based claims vs opinion-based claims with an F1 score of 0.8. They also identified sources using simple string matching as well as identifying news outlet names, they achieved F1=0.67, impressive given the simple, unsupervised approach. [58] takes a granular approach to claim classification identifying the following types of claim: *"Not a claim, Other type of claim, Quantity (past/present), Prediction, Personal experience, Correlation/causation, Current laws/rule"*. They use a supervised multinomial logistic regression model, achieving

F1=0.48.

Finally, more recently [59] explored the impact of claims within an argument achieving F1=0.57 using a supervised BERT model, they note that models using context vectors gain 3.84% accuracy on average, and those using pre-trained encoders (such as BERT) gain 5.49

2.9 Applications

Although Argumentation Mining maps most closely to classical debates [57] it can be easily seen applied to many further applications, such as reasoning to predict stock prices, evaluate information reliability, and reduce fake news on social media as demonstrated by [53]. In reality, it can be applied to any field where you wish to reason with natural language in an argumentative manner. There are a number of popular sources for argumentative text that have seen repeated use...

2.9.1 Twitter

Twitter is a microblogging and social media service, where users can post *tweets* up to 280 characters long. Users post new tweets that can be either the root of a conversation or in response to an existing tweet. Tweets can also refer to hashtags (topic identifiers) or users (mentions) within their tweet. All tweets are publicly available. Users can follow other users, this generates a social network depending on the friends and interests of users. The overall aim is to create *"highly skimmable content"*. [60] [61]

Applications of Twitter Data

Twitter is a widely used source of opinions, for instance: political [62]) [63], scientific [64], planning [16], educational [19] etc. A variety of which already employ sentiment analysis (a component of stance detection).

The difficulties of twitter

Another noteworthy property of Twitter is how representative it is. It has been shown by both [65] and [66] that Twitter does not represent any population other than itself, *"Sentiment analysis can be done, just as long as the sentiments do not need to be representative of any larger population"*. In particular, *"Twitter users fit the profile of young, well-educated, wealthy elites"*. Although Twitter is used worldwide, it is disproportionately popular in some regions, for instance 8.8% of users (with populated account locations) live in the US [67].

2.9.2 IMDB

IMDb (internet movie database) is an online repository of information related to films, TV and other media. However, primarily it supports *user reviews*, which allow internet users to rate a film [1-10] with a title and description for their review. This makes the site very popular for sentiment analysis tasks [20, 23, 24] as a vast amount of semi-structured text from a variety of users can be mined from the site, each with ratings that can be used as noisy labels to indicate the sentiment of the review itself. Furthermore, the text is generally somewhat argumentative with longer reviews presenting arguments and supporting them with evidence; however, there is no facility that labels these arguments. Accordingly, IMDb's uses are very limited for argumentation mining since data must be annotated either way, sites such as Twitter (with a broader range of subjects beyond IMDb's restriction to media) are far more appealing.

2.9.3 Debate Forums

Debate forums host user-generated discussions on proposed controversial topics, where users can list arguments for/against the topic. One of the most popular of these sites is *Kialo*, users categorise their arguments into 'pros' and 'cons' against a given topic. Each of these arguments then create a new topic where users can again list 'pros' and 'cons', as such a tree structure is formed that closely follows the theory for argumentation mining (section 2.1). Due to this debate forums have consistently been the preferred data source for much of argumentation mining's history thanks them providing such a labelled data-set. However, perhaps due to the extra effort required from users, debate websites such as Kialo are far less popular than main stream social media, as of writing 10^6 posts have been uploaded to Kialo according to their website, while Twitter statistics indicate 10^{11} tweets are uploaded annually. From this factor 10^5 difference it is clear to see that social media holds far greater potential to represent a far larger community of users. It is for this reason that we believe Twitter to be the ideal target for argumentation mining.

2.10 Summary

Since claim detection will be the study of this report, it is worth reviewing the performance of the models seen so far.

	Model-Paper	Supervision	Accuracy	Year
1	Joint [39]	N	0.56F1	2019
2	TDCOLex [44]	Y/N	0.585Map	2018
3	Linear [43]	Y	0.56F1	2016
4	Linear [46]	Y	0.52 to 0.78F1	2017
5	BERT [5]	Y	0.71	2019
6	GNN [50]	Y	0.61	2021

Table 2.2: Claim detection performance review

2.10.1 Structured vs Semi-Structured Text

The lower performance models namely 3 and 4 in table 2.10 both use simpler architectures (Linear Regression) than those that perform better, but they are also applied to more challenging datasets, namely Twitter which consists of semi-structured text, which has been shown to lack the type of argumentative structure and discourse makers present in classical argumentation datasets. For example, a random sample 3 of argumentative sentences have been extracted from the UKP dataset [48] and the Twitter dataset from [44]

UKP:

1. *Because clone cells may <u>show</u> genetic abnormalities, using them on humans will also have a deadly <u>implication</u>.*
2. *Higher levels of gun ownership do not produce a safer society and often lead to a higher numbers of deaths <u>due</u> to gun violence.*
3. *They may exercise these rights by, <u>for instance</u>, using contraception or natural family planning.*

Twitter:

1. *@Nicole_Brewer @CBSPhilly @NYGovCuomo Repubs are against ILLEGAL immigration. Question is why are the gov and libs FOR illegal immigration*
2. *@TsepisoMakwetla @safmpmlive Tertiary /University education is a privilege not a right, especially post-graduate students*
3. *RT @NAChristakis: Complete gender reversal higher education. Crossover in 1981; by 2026 women:men college degrees 3:2 | @Mark_J_Perry*

The contrast in the language used is immediately apparent; the UKP sentences are grammatically correct with correct spelling, punctuation, grammar, and capitalisation. While the Twitter sentences all first begin with tags (@...) and then contain shorter sentences, which in the third example is grammatically incorrect being more of a list than a sentence. You can also see the use of capitalisation for emphasis in 2 examples as well as numbers and symbols such as ":" and "|". A closer look at the Twitter dataset also reveals a number of tweets containing emojis such as *"but we dont need gun control right 🤔 https://t.co/kLtYDqfH6k"* as well as URL links. Seeing this discrepancy in the language used between the two datasets highlights that the performance of models trained on classical debate style datasets such as the UKP dataset should not be directly compared to modern social media datasets such as Twitter.

Knowing this if we consider the best performing model from our research, namely BERT as used in [5]. The model was evaluated on the UKP and IBM datasets (2 classical datasets containing structured text) but has not been evaluated on social media datasets, or any dataset containing semi-structured text for that matter. As of writing, we have been unable to find any such paper that evaluates the performance of BERT on such a dataset. This represents a significant gap in the current research in the field, as there is no understanding of how the difficulty of claim detection varies between structured and semi-structured datasets.

Chapter 3

Specification

As shown in the review of structured vs semi-structured text (see section 2.10.1) there is a significant discontinuity in the research conducted on claim detection. Older and simpler models have primarily been applied to semi-structured text, while newer state of the art models have only been applied to structured text. The primary goal of our implementation is to address this discrepancy by evaluating the performance of a state of the art model on both structured and semi-structured data. Furthermore, it has been shown that there is a significant difference in the vocabulary and language of structured vs semi-structured text (see section 2.10.1), and that due to this, modifications must be made to a model in order to facilitate this difference (section 2.6.3). Accordingly, the second goal is to make the necessary modifications to adapt structured text models to semi-structured text models and validate their performance. To more easily discriminate whether these goals have been met, they have been decomposed into a set of specific and measurable requirements.

3.1 User Requirements

The solution developed is only for theoretical purposes, and despite this there will exist some user interaction. Since the user will have to be able to train the model and evaluate performance. However, due to the academic nature of the investigation, it is reasonable to assume that any user will have an aptitude for code-based projects. Formally, the project must provide the following facilities to a user:

1. user can commence model training with minimal understanding of the implementation
2. performance in each test case is clearly reported with no ambiguity

3. performance metrics reported via the widely adopted F1-score and Accuracy
4. self-documenting code is used throughout the implementation

Self-documenting means that naming conventions and code structure provide sufficient information to understand the code without prior knowledge. Broadly speaking this is the approach adopted in the field of machine learning, and as such it is apt to adopt this approach in this project too. Under this scheme, it is conventional to prioritise the readability of our code over its performance.

3.2 Functional Requirements

This section elaborates on the expected behaviour of the solution.

1. develop a model to detect claims in semi-structured text at the sentence level
2. report the performance of this model on semi-structured text as well as structured text, facilitating their fair comparison
3. make necessary modifications to state of the art models to suit semi-structured text
4. report the performance of this modified model in a way that can be fairly compared with the unmodified model's performance
5. the model can be run on any machine independent of its hardware or operating system
6. the model can be trained in under a week with reasonable hardware

There are no requirements on the performance of the model, as the goal of the investigation is to better understand the performance on semi-structured text, although it is reasonable to assume that the model should outperform classical architectures such as [43, 46], which achieve under 0.56F1

Chapter 4

Design

Following the specification, there are a number of questions to be addressed, each of which will be addressed in turn throughout the design.

1. Which structured text dataset to use?
2. Which semi-structured text dataset to use?
3. Which state of the art transformer architecture to use in order to detect claims within these datasets?
4. How to pre-process each dataset for input to this architecture
5. The exact architecture used by this model
6. Architectural changes to suit semi-structured text

4.1 Data - Which structured text dataset to use?

Classically research has always focused on claim detection from structured text copra; accordingly, there is a choice of datasets, most of which have been labelled. Unfortunately, very few of these have been published:

Given the availability of datasets, our choice is limited to the UKP dataset, as it is the only structured text dataset that has been labelled and is publicly available. Fortunately, it is also used in the best reference implementation of BERT for claim detection [5], so it will allow direct validation of the performance of our implementation. The UKP dataset also has a high labelling agreement with a Cohen Kappa score of 0.721, this greatly exceeds the 0.45

Name	labelled	source	published
UKP [48]	Y	News, blogs, debates, encyclopedias	Y
IBM [49]	N	Wikipedia, debates	Y
[68]	Y	Wikipedia, debates	N
[40]	Y	Persuasive Essays	N

Table 4.1: structured text datasets

Fleiss kappa agreement achieved in the IBM dataset [49], although 10 annotators were used here rather than 2.

4.2 Data - Which semi-structured text dataset to use?

There is a similar problem with semi-structured text data sets, in which very few datasets are publicly available. For instance DART [45] sees heavy use in the field [43, 46] but is unfortunately unavailable much like the datasets used in [69, 58, 70]. The only suitable dataset found during the investigation is from [44] which covers a corpus of 2520 tweets; however, only 524 of these tweets are public/recent, and can be retrieved via the Twitter API. This leaves us with the task of constructing our own dataset, there are 3 commonly used sources for semi-structured text, IMDb, Twitter and Reddit. Since IMDb is restricted to media, and Twitter has seen more use in historical research than Reddit, Twitter is the most applicable. Accordingly, tweets will be extracted using the Twitter API and manually annotated with whether they contain a claim. [44, 37] will be used to reinforce what constitutes a claim, alongside the definition given in section 2.6.

4.3 Which transformer to use?

Since BERT's inception in 2019, many more Transformer models have been developed, most are variants of BERT, but some change the architecture completely. The 5 most notable models are:

1. GPT-2
2. GPT-3
3. BERT
4. RoBERTa
5. XLNet

6. DistilBERT

BERT [47], XLNet [23] and RoBERTa [71] share the same underlying architecture, all having 12 stacked self-attention layers and 110 million parameters, structured in an encoding architecture, meaning they are best suited for interpreting natural language. XLNet and RoBERTa are newer approaches to train BERT, involving more data and compute power. DistilBERT [72] is another revision of BERT that simplifies the architecture and reduces the training time at a small cost to performance. XLNet modifies the BERT training process but also requires 10x more data (33 billion words vs. 3.3 billion used by BERT), which takes far longer to train. RoBERTa makes minor changes to the BERT training processes but is also trained on a much larger corpus. A direct comparison has been made between BERT, RoBERTa, XLNet and DistilBERT in [73] applying each model to the same emotion recognition task. This is essentially an opinion mining task, which is a predecessor of argumentation mining, hence their results are extremely relevant to claim detection. In their experimentation, they found that both XLNet and RoBERTa outperformed BERT by 4% and 3% respectively. With DistillBERT performing 3% worse.

GPT-2 [74] is a transformer model developed by OpenAI in 2019 with 1.5 billion parameters and 36 stacked self-attention layers. GPT-3 is a later iteration of GPT-2 that increases the number of parameters to 175 billion. Each are decoder architectures, meaning they are best suited for natural language generation. For this reason, GPT models are not suitable for the task of claim detection as we are interpreting natural language rather than generating it. Furthermore, their vast number of parameters makes them intractable to work with without enormous compute resources.

In conclusion, a BERT based model is most suited to claim detection, narrowing our options to either BERT, RoBERTa, XLNet, or DistilBERT. Although XLNet and RoBERTa boast better performance than BERT they each require far more data and compute time to train, so are not viable for the project. Further, we believe that some of the 3% performance gain RoBERTa has over BERT could be down to their Dynamic Masked LM technique, which could be easily integrated into BERT's pre-training. Using BERT has the added benefit of being directly applicable to all BERT derivatives, allowing our findings to have a much broader scope to aid future research, a benefit which could not be understated! Finally and most importantly, BERT has already seen use in the field [5], meaning that the model trained here could be easily imported across to these other projects. It is for these two reasons that BERT is preferred over DistilBERT, despite its better performance on small datasets. Overall, we believe that BERT as used in [47] is the best architecture choice for claim detection at the time of writing, especially with the addition of Dynamic Masked LM.

4.4 Input Encoding

Now that we have established both what architecture to use, and what data it will be applied to we can address how to encode the data for the model. The structured text dataset is very conventional, so conventional NLP prepossessing can be used; however, the semi-structured Twitter dataset contains a variety of unique features that must first be addressed.

4.4.1 Twitter Pre-Processing

As highlighted in section 2.10.1, Tweets contain a number of platform-specific features:

1. Retweets
2. Mentions
3. Hashtags
4. URL
5. Emojis

As used in [44] retweets, mentions, hashtags and URLs can all be encoded as *binary features* by replacing each occurrence with [RT], [MEN], [TAG], [URL] respectively, this reduces the vocabulary as there are infinitely many permutations of each feature. By extension, this ensures that unseen data is also interpreted as the correct feature.

To date, no papers have addressed the use of emojis in semi-structured text, despite [30] finding ”Emoji information is useful in sentiment analysis“. There are two methods in which this can be done, firstly by simply assigning each emojis its own token, however, there are a vast number of emojis and emoticons (ASCII emojis) so this would be expensive for the vocabulary. Instead each emoji can be mapped to it’s description according to the *Unicode Consortium* [75], the same can be done for emoticons [76]. This has numerous advantages; firstly, it reduces the vocabulary by using English words likely to already be contained in the vocabulary to describe each emoticon. Secondly, it addresses the emoji *sequence modifiers* (section 2.5.1), as these are included in the description mappings. Finally, it standardises the resultant dataset to contain only conventional natural language, making the dataset easier to use by others in future research.

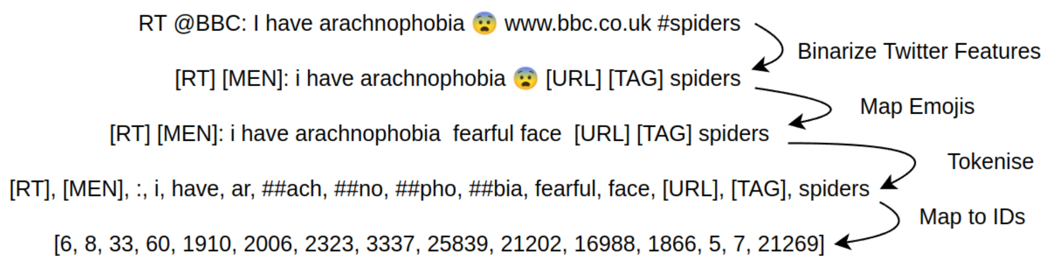

RT @BBC: I have arachnophobia 😨 www.bbc.co.uk #spiders
Binarize Twitter Features
[RT] [MEN]: i have arachnophobia 😨 [URL] [TAG] spiders
Map Emojis
[RT] [MEN]: i have arachnophobia fearful face [URL] [TAG] spiders
Tokenise
[RT], [MEN], :, i, have, ar, ##ach, ##no, ##pho, ##bia, fearful, face, [URL], [TAG], spiders
Map to IDs
[6, 8, 33, 60, 1910, 2006, 2323, 3337, 25839, 21202, 16988, 1866, 5, 7, 21269]

Figure 4.1: Pre-Processing pipeline example

4.4.2 Tokenisation

Now that the Twitter-specific features have been removed, the corpus is simply plain text. Plain text cannot be directly processed by BERT instead text must be mapped to a sequence of ID's, the first step in this process is tokenisation. Tokenisation decomposes text into lists of smaller strings called tokens such that these tokens can form a vocabulary that is used throughout the corpus. There are 2 key forms of tokenisation, word level and word piece level. At the word level, we split text to form a token from each word/symbol. While a word piece tokeniser (described in detail in section 2.6.3) decomposes each word further into its root, such that multiple words can map to the same root. Historically word-level tokenisation has been the more popular choice; however, transformer architectures have great power to draw context from elsewhere in the input, for this reason, word piece tokenisation was successfully adopted with the BertWordPieceTokenizer in [47] and has been frequently used since. The BertWordPieceTokenizer is the best choice, as it is designed with transformer models in mind, it's also used in the reference implementation [5] and produces a vocabulary that can encode unseen words (see section 2.6.3). Finally, this list of tokens is encoded by assigning a unique number to each token in the vocabulary (which contains all tokens encountered during training). This gives a mapping from tokens to numbers which can be applied to each list of tokens to give a list of IDs. Now, these IDs can be one-hot encoded and passed to the model for input. Note that a one-hot encoding is formed by a vector of zeros of length equal to the vocabulary where the index equal to the ID is set to 1.

4.4.3 Cased or Uncased

Another decision must be made before looking at the architecture, which is whether to use a cased or uncased model, in a cased model (unlike the pipeline example) text is not converted to lower case before tokenisation. A cased model greatly increases the size of the vocabulary of a model and requires a much larger dataset to expose each word under both a capitalised and non-capitalised context. This is important given the size of the semi-structured text corpus manually created later. Due to this, the reference implementation [5] uses an uncased model; accordingly, an uncased model will be used throughout our implementation.

4.5 BERT Architecture

BERT [47] (Bidirectional Encoder Representations from Transformers) is a sequence-to-sequence model that can process an arbitrary sequence of inputs, encode them and then decode them into a arbitrary sequence of outputs. It is designed for *natural language understanding*, for example, question answering or language translation. BERT is a Transformer model, which means it employs **self-attention** layers. The architecture is as follows:

1. Embedding layer - transforms the input vocabulary (30k words) into fixed vector of size h (where h is the latent space size for the model)

2. Encoder - Sequence of 12 layers, each consisting of a self-attention layer followed by a dense (feed-forward) layer

3. Decoder - Sequence of 12 layers, each consisting of a self-attention layer followed by an encoder-decoder attention layer followed by a dense (feed-forward) layer

There are multiple configurations of BERT given in the original paper, ranging from tiny to mini to small and finally base where each varies the number of layers and the size of the latent (hidden) space. We will be using BERT base, this version contains the embedding layer and a 12 layer encoder, with a latent space h=768. Although it requires more time and data to train, this is the same architecture as was used by [5]. As such, we can directly validate our baseline model performance, since both their code and data set is publicly available. Additionally, using the largest possible architecture reduces the chances that the model has insufficient power to fit to the dataset.

Since claim detection is a classification problem, we don't require a sequence of outputs, hence only the embedding layer and encoder is used, with a classification layer added afterward. In the following sections, we explain the architecture of these 3 components (as is implemented in the PyTorch source code [77]), but first, we must address how the input to these components is generated.

4.5.1 Segment IDs & Input Masks

BERT relies on 3 input sequences to preserve contextual information that would otherwise be lost due to the architecture:

- **Input IDs** - List of token ID's generated during tokenisation$^{4.4.2}$
- **Segment IDs** - Distinguishes between the first (0) and second (1) sentence for NSP$^{4.6}$
- **Input Mask** - Vector of 1's for all input ID's, otherwise 0

A cumulative sum is applied to the input mask as soon as it is passed to the model to generate *positional embeddings*, this proves the context for the position of the token in the input.

4.5.2 Embedding Layer

The embedding layer is the first layer in the BERT model, meaning it is directly applied to the encoded input data (section 4.4). The input encoding results in a list of ID's which encode an input sentence. The embedding layer's purpose is to convert this list of input ID's into a list of smaller vectors that can be interpreted by the following layers. Depending on the input vocabulary, there are typically 30k different ID's, using one-hot-encoding gives us a vector with 30k elements for each input token. The embedding layer is a dense feed-forward layer with 30k inputs and 768 outputs, this same dense layer is applied to every ID in the list of input ID's. A similar dense layer is applied to the segment ID's, but this has 2 inputs and 768 outputs since the segment ID's are either 1 or 0. Finally, a similar dense layer is also applied to the position embeddings, the maximum value of the positional embeddings depends on the length of the input since they denote the input index. [47] made a reasonable decision of limiting the input size to 512, hence limiting the maximum sentence length supported by the model to 512 tokens, this is very practical since the model required for such a long sequence would use copious amounts of memory so most implementations limit the sentence length to 64-128 tokens anyway. Given the 512 possible inputs, the position embedding is processed by a dense layer with 512 inputs and 768 outputs. The result of these 3 dense layers is a set of 3 vectors, each of length 768, for every input token, which encodes the token, position and segment, the sum of these is then taken to give a single 768 element vector.

4.5.3 Self-Attention

Self-attention layers are designed to focus attention on the most important context for the given input time step in both the forward and backward directions (left & right context). For example the word “it” always requires context to understand its meaning, such context could be provided before or after its use. The input vector from the previous layer is passed through 3 dense layers, each with 768 inputs and 768 outputs, giving 3 vectors. These vectors are named ***Query, Key, Value***. There n of these triples, where n is the number of tokens input to the embedding layer, where the i^{th} triple $Query_i, Key_i, Value_i$ corresponds to the i^{th} token input to the model x_i. From this we can calculate a score for each token:

$$Score_j = \sum_{i \in n} query_i \cdot key_j$$

Typically the score is then normalised by the square root of the *number of attention heads* where the number of attention heads is just the size of the input, n.

$$Score_j = Score_j \div \sqrt{n}$$

The score is then passed through the SoftMax function; this normalises the output into a probability distribution that sums to 1, helping to prevent exploding gradients.

$$Prob_j = -\log(\frac{e^{Score_j}}{\sum_k e^{Score_k}})$$

Finally, the probability distribution is multiplied by the *Value* vector we calculated, giving the *context* vector

$$Context_j = Prob_j \cdot Value_j$$

This gives a 768 element vector that combines information from every other token in the input, hence *bidirectional transformer* in the name, as the attention goes in both directions. Under BERT’s architecture, 12 of these layers are stacked, each taking a list of n 768 element vectors and outputting n 768 element vectors.

4.5.4 Classification Layer

The classification layer is a dense layer that maps from the 768 outputs of the network to n classes. Although claim detection is a binary classification problem so technically only requires a single output, using an output neuron for each class makes the architecture more versatile. For instance, it can be easily adjusted for 3 class classification such as that supported by the UKP (structured text) dataset with labels *NotAClaim, ForClaim, and AgainstClaim.* And beyond this, for Masked LM tasks where an arbitrary number of classes may be needed depending on the size of the vocabulary.

Activation Function

In the reference implementation, an activation function is not used [5], this does not follow the norm of applying the SoftMax activation function to the output for multi-class classification problems. The SoftMax activation converts the output into a probability distribution which naturally suits the classification problem. The SoftMax function is defined as:

$$SoftMax(z)_i = -\log(\frac{e^{z_i}}{\sum_j e^{z_j}})$$

An alternative activation function would be the Sigmoid activation function which maps each output to the range [0,1]. It is better suited to multi-label classification (no constraint on the number of classes an example can be assigned to) problems as there is no restriction for the output to sum to 1. Since both normal claim detection and 3 label claim detection are single-label classification problems, the SoftMax activation function is most applicable.

4.5.5 Loss Function

Given the SoftMax function is being used and the multi-class classification problem, the best loss function to train the network with is the cross-entropy loss (Log Loss), as it is designed to measure the difference between probability distributions, since both our prediction and target are probability distributions. It is defined as:

$$CrossEntropyLoss(y, \hat{y}) = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\hat{y}_i)$$

Where y is the target probability distribution, $\hat{y}$ is the predicted probability distribution, and N is the size of the output vector. A common alternative to cross-entropy loss is the L2 loss which simply measures the euclidean distance between two vectors, this is not ideal as we should pay most attention to the label assigned to the target class. Simply put, we care most about the class predicted in $\hat{y}$ which is given by the max element, and how that compares to the maximum element in the target $\hat{y}$, the cross-entropy loss does this, while L2 loss does not.

BCE Loss (binary cross entropy) is another common loss function, which is a generalisation of the cross entropy loss, facilitating multi-label classification:

$$BCELoss(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)$$

Since we are only performing single-label classification, BCE loss is not as applicable as the cross-entropy loss. The exception to this is during Masked LM[4.6] pre-training, as Masked LM is a multi-label classification problem; BCE loss must be used here.

4.6 Modifications to suit semi-structured text

As discussed in section 2.6.3, BERT was not trained with semi-structured text corpora in mind, as such the pre-trained model has a vocabulary and trained weights that reflect the vocabulary and language used in structured text. According to section 2.6.3, the only way to account for both of these factors is to retrain the model just as it was in [47] but on a semi-structured text corpus. However, the unsupervised pre-training methods used in [47] require a very large dataset, in their implementation 3.3 billion words were used. Accordingly, the semi-structured text corpus gathered for this study must be of similar size. Thankfully, the pre-training methods used were Masked LM and NSP which are both unsupervised, so text can simply be mined from Twitter and directly used to train the model.

Masked LM

Masked LM (MLM) is the first pre-training step for BERT which functions by training the model to guess the missing word(s) in an input sentence. MLM is also referred to as the *Cloze* task [78]. It functions by extracting the set of all sentences from the text corpus then masking 15% of the input tokens and adding the ID's of these tokens to the target (multiple ID's can be added to the target under the one-hot encoding, as it'll simply become an *n-hot* encoding). Each of the tokens selected to be masked are then dealt with in one of three ways:

1. it is replaced with a [MASK] token 80% of the time
2. replaced with a random token 10% of the time
3. left unchanged 10% of the time

The same classification layer architecture is used as before, but now is configured to have as many outputs as the vocabulary. Technically, the SoftMax activation is no longer applicable due to the multi-label nature of the task, however, for consistency, SoftMax will be used, further SoftMax is also used in the original paper [47].

It is worth noting the difference between the Masked LM used by Google's original BERT research [47] vs Facebook's modifications with RoBERTa [71]. As one of the many changes in RoBERTa, Facebook introduced *Dynamic Masking* where the chosen masked words were changed each epoch. In Google's version, the masked words are generated once for each sentence, then remain the same for every following epoch. We have chosen to include RoBERTa's mechanism in our pre-training for two reasons. Firstly RoBERTa boasts a 3% performance improvement over BERT, which must be down in part at least to their *Dynamic Masking* although this has not been conclusively shown. Secondly, it has the added benefit of increasing the effective training corpus, in the case of the UKP dataset it increases from 80,698 samples to 2,056,088.

Next Sentence Prediction

Next sentence prediction is the second pre-training step, which is applied to the output of the MLM train step. Next sentence prediction (NSP) is a textual entailment task whereby BERT is given a pair of sentences and must deduce whether one entails the other. 50% of the training examples are chosen to have a pair of sentences where one follows the other in the original corpus, the remaining 50% have a random second sentence that does not follow the first. This trains the model to understand the relationships between sentences; together with MLM, this gives the model a good understanding of the language of the corpus. In the case of the UKP dataset 22,236 NSP training samples can be generated. In the case of the Twitter dataset (described later) 148,844 NSP training samples can be generated. The classification layer used for NSP is identical to that used for claim detection, as both are binary classification problems.

Pre-Training Result

The result of these 2 pre-training steps is a model that has been trained in the target domain, be it Twitter (for semi-structured text) or the UKP corpus (structured text). This trained model has both the correct embeddings for the domain vocabulary and trained weights that reflect the language used in that domain. This pre-trained model can then be fine-tuned for claim detection in the same manner as the reference implementation [5].

Chapter 5

Implementation

The project was implemented in a series of phases. First, a claim detection model is implemented and verified to have the same performance on the UKP dataset as the reference implementation. Then the model is pre-trained on the UKP dataset and then fine-tuned again for claim detection and the performance compared. Since the default BERT model is pre-trained for structured text, the additional pre-training is not expected to improve performance. Now we move onto the semi-structured Twitter corpus, first, we must mine the dataset from Twitter, then annotate this corpus, and train the default BERT model for claim detection on it. Finally, we pre-train a model on the Twitter corpus and fine-tune it for claim detection, we expect that the additional pre-training step will improve performance.

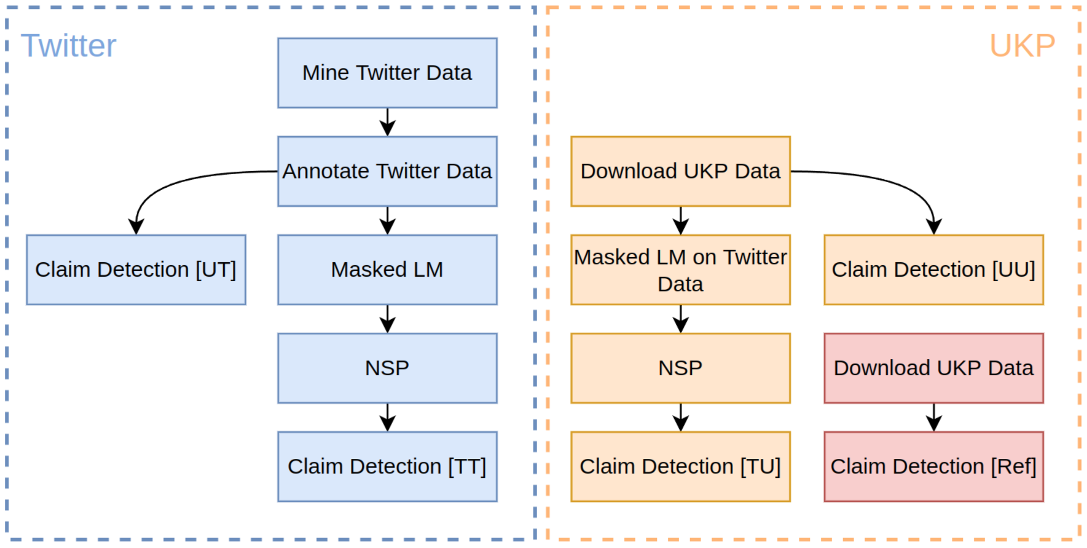


Figure 5.1: Implementation Pipeline

5.1 Development Language

Before anything can be developed, the programming language used for the project must be established. The primary dependencies are Tensorflow/Pytorch with supporting modules that implement the BERT architecture, as well as support to interface with the Twitter API. The only language that supports all 3 of these requirements is Python, which is the de facto standard for most machine learning projects. The code is organised into interactive blocks, following the convention for machine learning projects, as they allow both code and it's outputs to be reported in a single document. The specific tool we will be using is iPython simply due to personal preference, jupyter notebooks provide the exact same functionality.

5.2 Dataset Construction

For the sake of grouping the machine learning components together, let's first discuss how the semi-structured text corpus was mined and annotated.

5.2.1 Mining Twitter w API

An existing dataset of tweets could have been used, but mining tweets from Twitter grants greater flexibility over the chosen topics. Further an argumentation mining solution could then use the Twitter interfaces developed here in later research. Twitter has 2 versions of their API v1 and v2, crucially the version 2 API supports conversation ID's, whereby each new tweet is assigned a conversation ID and any replies to this tweet maintain this conversation ID. The support for conversations enables the extraction of conversations spanning multiple tweets, increasing the chances of having multiple claims over the same topic. It also adds depth to the argumentation tree, should future research progress to that stage. For this reason, the v2 API was chosen, despite it only being fully deployed in November 15th, 2021. Currently, the only python package the supports the v2 API is the TwitterAPI package, which only provides an interface to send string requests to Twitter. Since more functionality than this is required, a wrapper was developed that supports the following operations:

- `get_user` - retrieves all the attributes of a given username, such as their name, Twitter user id, account age, user description, location etc (locations are countries so are not personally identifiable)

- `get_recent_tweets` - retrieves all the tweets the user has posted

- `get_tweet` - retrieves an individual tweet given it's ID
- `get_all_replies` - given a conversation ID retrieves all the tweets in that conversation
- `search` - searches using Twitter's search algorithm, facilitating most queries, such as searching by hashtag or keyword

Twitter offers a plethora of attributes for each tweet, the API Wrapper requests all of these attributes: `attachments`, `author_id`, `context_annotations`, `conversation_id`, `created_at`, `entities`, `geo`, `id`, `in_reply_to_user_id`, `lang`, `possibly_sensitive`, `public_metrics`, `referenced_tweets`, `reply_settings`, `source`, `text`, `withheld`. Note the `public_metrics` attribute contains a number of useful sub-attributes such as the number of likes and retweets. These can be used to measure the engagement users have with a tweet and hence focus attention on those tweets that receive the most attention on the platform.

There are three major limitations of the Twitter API, the primary of which are the imposed *rate limits* which limit all access levels to 900 tweets every 15 minutes. Due to this after 6 days of continual mining, only 500k tweets were collected. Secondly, under the *elevated access* secured for this project, only *recent tweets* can be mined. However, this can be resolved, given support from an academic body, *academic access* can be granted giving full access to historical tweets although the rate limitations remain. Since the rate limits were found to be more of a limiting factor obtaining such access was unnecessary. Finally, each request is restricted to retrieving 100 tweets, this is a major limitation as many conversations on Twitter can span thousands of tweets or more. Thankfully, this can be resolved by using a Twitter Pager which iterates through the request, decomposing a large query over thousands of tweets into many smaller sub queries. This enabled the retrieval of entire conversations from Twitter, the wrapper operations were modified to support this paging, and given the post-fix `_all`.

Since Twitter API interactions depend on a large number of interactions, all requests are wrapped in a safety method that handles any exceptions caused by network errors/insufficient access rights, or simply invalid requests. Errors are handled silently so as to not interrupt data mining operations, since the application must run continuously over extended periods of time, handling exceptions silently is very important. During execution, network errors proved to be very common, highlighting the importance of this silent error handling.

5.2.2 What to mine?

A mixture of conversations over controversial topics and popular users were mined, the popular tags *#unpopularopinion* and *#discussion* were focused on, but a variety of media topics such as *Harry Potter* and *Marvel* were mined, as well as a number of users such as the BBC, NASA and Elon Musk. Although it is worth noting that the entire conversation was mined for each tweet within each set of 100 search results, as such there is far more subject diversity than what is listed here. *#unpopularopinion* and *#discussion* are preferred as they are very subject-agnostic terms. Overall, a corpus of 471,427 Tweets was mined over the course of 6 days, containing 8,037,391 words, this is roughly a quarter the size of the Google dataset; however, we believe this will suffice given the use of dynamic masked LM.

5.2.3 Processing Twitter

Now we have a dataset of 471k tweets, before we pass this data to the model, the Twitter-specific features must be binarised according to section 4.4.1. This processing is performed before sanitisation as the binarisation also helps to group together posts made by bots (more on this later). First regex (regular expression) is used to match *mentions*, *hashtags* and *URL's*. Regex was chosen as it fully specifies the criteria used to match subsets of text in a simple string. The following Regex formulas were used:

- **Mentions** - `@[^ !@#$%^&*(),.?":{}<>]*` - uses a non matching list to stop a tag when any of the symbols are encountered
- **Hashtags** - `#[^ !@#$%^&*(),.?":{}<>]*` - much the same as mentions
- **URLs** - *(https?:\/\/(?:www\./(?!www))[a-zA-Z0-9][a-zA-Z0-9-]+[a-zA-Z0-9]\.[^\s]2,/www\.[a-zA-Z0-9][a-zA-Z0-9-]+[a-zA-Z0-9]\.[^\s]2,/https?:\/\/(?:www\./(?!www))[a-zA-Z0-9]+\.[^\s]2,/www\.[a-zA-Z0-9]+\.[^\s]2,)* - commonly cited across the web as a good URL regex

Retweet tokens only appear at the start of a tweet; these were binarised by simply mapping appearances of `rt` at the start of a tweet to `[RT]`.

Next emojis are mapped to their descriptions; there are two forms of Emoji's commonly used on Twitter, Unicode and ASCII. Unicode Emoji descriptions have been provided by the *Unicode Consortium* [75], which has been exposed to python with the `Emoji` python package. ASCII emoticons have been manually imported according to [76]. Both mappings are combined and then used to replace any Unicode or ASCII emoticon appearances with their descriptions.

Finally, due to Twitter being a multilingual platform, non-English tweets must be removed; thankfully, Twitter encodes this information within the meta-data for a tweet. So any tweets with a language attribute not equal to `'en'` are deleted.

5.2.4 Sanitisation

Before using the dataset it must be sanitised, as Twitter contains many bots, which post very similar messages thousands of times. Fortunately, most of these bots only vary these messages using random URLs and mentions. Consequently, after binarising mentions, tags, retweets, and URLs, the duplicated tweets are simply deleted; this minimises the number of tweets posted by bots without losing any data, as the number of unique training examples remains constant. Following this sanitization, we are left with 266,760 tweets and 5,618,315 words.

5.2.5 Reconstructing the annotated dataset

The dataset used in [44] is the most complete dataset of labeled tweets available at this time. The dataset is licensed for public use, but due to this, the tweets themselves cannot be publicly distributed. As such, the dataset contains only a list of tweet IDs and binary claim labels. Therefore, the tweets had to be extracted using the `get_tweet` command from the Twitter wrapper. The tweets are then processed and sanitised just the same as the other tweets we mined. The new tweets are then merged with the existing dataset of tweets collected in section 5.2.2. Unfortunately, due to Twitter's API restrictions, only 524 of the 2500 tweets could be recovered, either because they were not posted publicly or because they are no longer classified as *recent tweets* by Twitter. This adds more pressure for annotating our own dataset in order to have sufficient data to fine-tune the model.

5.2.6 Annotating

Tweets were annotated to determine whether they contained a claim, according to the definition *"a statement that something is true or is a fact, although other people might not believe it"* [34]. Inspiration was also taken from [44] to guide where to place the boundary between claim and non-claim tweets. Due to the informal nature of Twitter, the boundary between the two is very obscure. As such annotating Tweets is a particularly time-consuming task, since the language used often leaves the meaning up to interpretation. To mitigate this, any tweets that could be interpreted either way were annotated as non-claim tweets, erring on the side of caution. For example, rhetorical questions are extremely common in the topics chosen for this study. With a typical tweet reading along the lines of *"What have those countries contributed to the world in terms of advancement of quality of life?"* the tweet's context implies it is claiming that the countries at question contributed nothing to quality of life; however, since this claim is not explicit, the tweet will be annotated as a non-claim tweet. After annotating the Twitter dataset the balance between the two classes is also considered, in order to remove the possibility of a biased model towards one of the classes, an equal number of claim and non-claim sentences is sampled.

5.2.7 Vocabulary

Finally a vocabulary must be decided for both the UKP and Twitter datasets. The UKP dataset will use the vocabulary of the default pre-trained model provided by Google. Since the Twitter model is entirely new, a new vocabulary will also be generated, this is done with a word piece tokeniser which maps the top 30522 words that appear most frequently in the dataset to ID's and stores the result in a local file for later use. All interactions with the Twitter dataset will use this ID mapping for the sake of consistency; this way the embedding layer will remain transferable between the Masked LM, NSP and claim detection stages.

Tokenisers

It is the job of the tokeniser to generate the vocabulary; there are 2 word piece tokenisers available for python the `BertTokenizer` and the `BertWordPieceTokenizer`. Both of which are provided in the `transformers` package from *hugging face*. Unfortunately, only the `BertWordPieceTokenizer` supports fitting the vocabulary, and the `BertTokenizer` only supports transforming data into ID's using the vocabulary. As such the vocabulary generated and stored by the `BertWordPieceTokenizer` and loaded into the `BertTokenizer`. This is not ideal as the special tokens `[URL]`, `[RT]`, `[TAG]`, `[MEN]` are not treated correctly (this is manually corrected for in the implementation used). Further loading from a vocabulary is a deprecated feature of the `BertTokenizer`. None of this is ideal, but due to a lack of documentation and alternatives, this is the best solution for a custom word piece tokeniser.

5.2.8 Test-Train Split

Following the convention, both the UKP and Twitter datasets were split into training, testing, and validation sets. The model is first trained on the dataset, then evaluated on the test set, then the models hyper parameters were improved (learning rate, batch size and optimisation function), and the model was trained again on the test dataset. This process was repeated until the performance on the test set was maximised. Finally, the model's performance was measured on the validation set, meaning that the performance is gauged on an entirely unseen dataset. The UKP dataset has an existing train-test-validation split of 72%-20%-8% respectively. While a more conservative split of 80%-13%-7% was chosen for the Twitter dataset to conserve as much of the training data as possible.

5.3 Model & Training

In this section, the methods through which the model will be implemented and trained are discussed. Initially, we discussed the best framework to develop the model with. Then build on the 3 different training tasks, namely claim detection, masked LM and next sentence prediction. For each, we cover how they have been implemented in order to follow the design as closely as possible. We also propose a variety of methods that differ from existing research, in order to increase performance, or better utilise the training data for the given domains.

5.3.1 Framework

First the machine learning framework must be chosen; this provides the necessary tools to implement and train a large host of models, the two most popular frameworks are Tensorflow and PyTorch. These frameworks automate the back-propagation process used to train differentiable models such as BERT using *auto-grad*. *Auto-grad* is a method of automatically tracking the gradients used in all mathematical operations during the *forward pass* of a model, such that these gradients can be automatically propagated back through the network to update the parameters during the *backwards pass*. Without *auto-grad* gradient flows would have to be manually calculated for each newly developed architecture, at a great time cost. BERT was originally developed for Tensorflow, but several libraries have been developed to help simplify working with BERT, the most notable being the *Hugging Face* transformer library. This library supports both the Tensorflow and Pytorch back-ends and was used in the reference implementation [5]. This leaves the choice of ML library mostly down to personal preference; however, PyTorch was chosen to match the reference implementation to help make the project as comparable as possible to the reference implementation.

Acceleration

PyTorch supports *GPU-acceleration* which allows for the majority of computation to be parallelised, greatly increasing performance. However, many PC's do not have powerful GPU's (Graphics processing units). To accommodate for this, and help increase the accessibility of the project, if a GPU is present in the users system it will be used, otherwise it will not, instead computation will be performed on the CPU. For the remainder of the investigation, a system with a Ryzen 7 3800X CPU and NVIDIA Titan Xp GPU was used, even given the highly threaded CPU (16 threads @ 4Ghz), GPU acceleration yielded a 6x performance improvement.

5.3.2 Replacing BERT's Vocabulary

The first consideration we must make is how to train a new vocabulary, as this subject is not covered in either the reference implementation[5] or Google's paper[47]. The BERT Hugging Face transformer implementation uses the exact architecture used in the original BERT implementation [5], as such the embedding layer expects a vocabulary of 30,522 words. However, there is no guarantee that the size of the vocabulary of the Twitter dataset is the same as that used in the original implementation. And in fact, the Twitter dataset indeed contains 100,268 unique words.

There are two solutions to this problem; first, the embedding layer can be replaced, or secondly, the Twitter vocabulary could be forced to contain 30522 words. Changing the embedding layer could have unintended consequences, as it would require the modification of an external implementation which is both unknown to us and is subject to changes and updates in the future. Consequently, forcing the vocabulary to contain 30,522 words was chosen, this was made possible by using a BERT specific word piece tokeniser which kept only the top 30,522 words (see section 5.2.7), minimising the loss of information.

5.3.3 Replacing BERT's Classification layer

Following the design established in section 4, a classification layer must be added to the output of the BERT model for each of the 3 tasks that we'll be training BERT for. The BERT Hugging Face transformer library does provide a classification variant of the BERT model which adds this layer. This classification variant is used in the reference implementation [5]. However, it does use a SoftMax activation layer as one would expect to see when performing classification. I believe the reference implementation's use of this classification variant to be an oversight due to this reason. Secondly, the BERT Hugging Face classification variant does not explicitly return the entire state of model expected after calling, instead the cross-entropy loss is returned when the model is in the training mode, and the actual output of the model is returned when in evaluation mode. Changing the return type like this is confusing and doesn't follow functional programming conventions. Thirdly, the classification layer will be changed between each of the 3 tasks the model is trained on, the hugging face implementation adds dependencies to this classification layer, so it cannot be simply replaced.

The implementation used in this investigation resolves these issues by using the standard BERT implementation and adding the classification layer independently. The return tuple is also invariant to the mode in which the model is in, always returning the output of the classification layer, the output of the BERT model, the pooled output and the hidden states of the network. Returning all these features facilitates any operations that could be performed in future research, such as the clustering using embeddings seen in [5]. The separation of the model and the loss function also allows us to use the cross-entropy loss during NSP and Claim Detection tasks, but the BCE loss during Mask LM, according to our design. Finally, it also reduces the dependency on the implementation of the BERT Hugging Face transformer, as this component of the model can be changed in a single line to a different implementation of BERT. Such as the original Tensorflow implementation of BERT from google's paper [47], accordingly we could independently verify the correctness of the Hugging Face implementation of BERT, although this is beyond the scope of our investigation.

5.3.4 Optimiser Choice & Learning Rate

In order to train the model, an optimiser must be chosen; optimisers adjust hyper-parameters to minimise the loss. The most simple of which is stochastic gradient descent (SGD). SGD keeps the learning rate constant throughout training, so weight updates are simply the back-propagated error multiplied by the learning rate each iteration. Adam is a much more advanced optimiser, which is extremely popular. Adam adjusts the learning rate during training according to the loss; this makes the initial learning rate choice less important, as Adam will optimise the learning rate over time; accordingly, a fixed learning rate of 2e-5 was chosen to match the reference implementation [5] and Google's implementation [47]. Adam implements both *RMSProp*, which divides the learning rate by an exponentially decaying average of squared gradients, and *Momentum*, which prevents rapid changes to the learning rate over time. AdamW is an improvement over Adam aimed at deep models where *weight decay* is often applicable, AdamW implements weight decay to prevent *exploding gradients* and help stop overfitting. Finally, BertAdam is a further modification of AdamW used in the original BERT implementation [47], which does not compensate for bias. RMSProp, Adam, AdamW and BertAdam were all experimentally compared, with AdamW and BertAdam performing best but BertAdam seeing greater performance on average, so was chosen as the optimiser for all model training.

5.3.5 Batch Size selection

The final hyper parameter to be chosen is the batch size, this dictates how many samples will be processed in parallel and is typically a power of 2 (due to GPU's typically having a power of 2 concurrent stream processors). [71] found that increasing the batch size is beneficial for classification performance, although the batch size is mostly limited by the amount of available memory, larger batch sizes require proportionally more memory. The Titan XP we used is limited to 12GB of memory, as such the maximum batch size experimented with was 32. However, decreasing this batch size to 16 resulted in a marked improvement in training performance, possibly due to the reduced copy overhead since memory copies now take half the time. Due to the improvement in training performance, a batch size of 16 was chosen, which also reduced the GPU memory usage from 11GB to 5.4GB.

5.3.6 Claim Detection

The first task for which we developed code for, was the claim detection[UU] task, which detects claims within the UKP dataset using the default BERT model. This facilitated the direct comparison of our outputs with the reference implementation to validate the implementation used throughout the investigation. Following the design, the inputs were word piece tokenised and mapped to ID's using the default BERT vocabulary (for Twitter data the vocabulary generated in section 5.2.7 is used). Then input masks and segment ID's were generated for the input data, forming a set of tuples `(input_id, input_mask, segment_id)`. These were shuffled and stored in a `dataset` structure that handled feeding the compute device (GPU/CPU) with training samples grouped into batches of 16. The default BERT model was then loaded using a classification layer with 2 outputs (according to section 4.5.4) and configured to use cross entropy loss (according to section 4.5.5). Then the model was trained for 4 epochs, in each epoch the model was called on a batch of samples and updated to decrease the loss on those samples, until all training examples had been seen.

5.3.7 Claim Detection vs Reference Implementation

In order to compare the performance of the claim detection model developed here to that found in the reference implementation [5] numerous changes had to be made to the reference implementation, in order to extract useful results. First, [5] simply assigns ID's to targets using an enumeration of the unique target values as they appear in the dataset. However, since the training, test, and validation sets were shuffled, this ordering was not consistent between training and testing. As such, performance was not correctly measured. In order to fairly compare the performance of [5] with our model, this was changed to generate a mapping and store the encoding using the `OrdinalEncoder` provided in the `sklearn` python package, thus ensuring a consistent mapping was used throughout the investigation.

The two implementations also differed due to the reference implementation's [5] choice to apply both a *linear warm up* and *linear cool down* to the learning rate during training. This increases the learning rate from 0 to 2e-5 over the first 300 iterations (where an iteration is one batch), and then decreases the learning rate back to 0 over the following 3000 iterations. Firstly, this method limits the number of epochs to a fixed number of iterations, which adds an unnecessary constraint on training, however, most importantly, *linear cool down* was not found to increase performance when using the BERT Adam optimiser. Since a *linear warm up* is included in the BERT Adam optimiser, we simply enabled it and did not manually implement a *linear warm up* or *cool down*.

5.3.8 Masked LM

Once claim detection had been successfully implemented and its performance verified against the reference implementation, masked LM tasks were performed. This involved pre-processing both the UKP and Twitter datasets for the task according to section 4.6. First, the set of sentences must be extracted from the text corpus. Typically this is simply done by splitting the corpus on each occurrence of a full stop character "`.`". Since the UKP dataset contains far more structure the mean length of these sentences was 26 tokens, due to this semi-colons "`;`" and commas "`,`" were also split on, giving a mean length of 23 tokens, bringing the UKP sentence closer to the mean length of 22 tokens in the Twitter sentences. From these sentences the Masked LM training examples were generated. However, since there are an exponential number of combinations of Masked LM samples given the length of the input sequence, it would be extremely memory inefficient to generate all possible combinations before training commenced. Instead a generator was used which computed the Masked LM examples from a sequence at train time for the given batch. Doing so performs Dynamic Masked LM, as each epoch will have a newly generated set of Masked LM examples. Furthermore, there is no performance impact since generating a Masked LM example from a sequence is computationally cheap and is computed in parallel with training, since each training batch is pre-fetched before it is needed.

Since the vocabulary has already been dealt with according to section 5.3.2, the next alteration is to append the appropriate classification layer to BERT's output. For Masked LM the classification layer must classify the masked token from the entire vocabulary of tokens, so the output size must equal the size of the vocabulary. Since both the UKP and Twitter datasets have been forced to have a vocabulary of 30,522 tokens, in either case, a classification layer with 30,522 outputs is used. The final alteration is to use the BCE (Binary Cross Entropy) loss according to section 4.6.

5.3.9 NSP

The final unique task is next sentence prediction, again using both the UKP and Twitter datasets according to section 4.6. Much like section 5.3.8 sentences are extracted from each corpus, again using the three punctuation symbols `.,;` to produce more sentences from the UKP corpus. Although in this case the sequence in which each sentence appears in for every Tweet/UKP debate, is maintained. Then each sequential pair is extracted and labelled as `True` since the first sentence entails the second. Then a random sample of sentence pairs is selected from the entire set of sentences of size equal to the number of sequential pairs found in each dataset (in order to maintain a balanced dataset). Each sentence pair is selected so that the first does not entail the second, but both can appear in the same tweet, or the second can entail the first. These pairs are then labelled `False` and added to the training set. This yields 22,236 UKP samples and 148,844 Twitter samples; since this uses far less memory than all the Masked LM samples (just 65MB of Twitter data), no generator is used, instead the samples are simply stored in memory, just as they were during claim detection. The model architecture is the same as that in section 5.3.6, following the design.

5.3.10 Input Truncation

Due to the BERT architecture expecting a constant number of tokens as input, there is a limitation to the maximum sequence length that can be passed as input to the model. BERT can be configured to support any input token length up to 512 tokens, but 64 was chosen to

Data	Task	Model	Model Outputs	Loss
UKP	Claim Detection [UU]	Default BERT	2	Cross Entropy Loss
UKP	Claim Detection [Ref]	Default BERT	2	Cross Entropy Loss
UKP	Masked LM	Default BERT	30,522	BCE Loss
UKP	NSP	Masked LM BERT	2	Cross Entropy Loss
UKP	Claim Detection [TU]	NSP BERT	2	Cross Entropy Loss
Twitter	Claim Detection [UT]	Default BERT	2	Cross Entropy Loss
Twitter	Masked LM	Default BERT	30,522	BCE Loss
Twitter	NSP	Masked LM BERT	2	Cross Entropy Loss
Twitter	Claim Detection [TT]	NSP BERT	2	Cross Entropy Loss

Table 5.1: Task Configurations

reduce memory usage, as well as maintaining consistency with the reference implementation. Consequently, sentences input to the model had to be truncated to 64 tokens (inclusive of the special tokens `[CLS]` and `[SEP]`). For single sentence inputs the input was simply truncated to length. Tasks with single sentence inputs include Masked LM and Twitter specific claim detection. However, for those cases with 2 sentence inputs such as NSP or UKP claim detection a token is removed from the longest sentence until the total input length was under 64 tokens (inclusive of the single `[CLS]` token and the 2 `[SEP]` tokens).

5.3.11 Claim Detection with the Pre-Trained model

Bringing all these components together, the final tasks performed are the various combinations of claim detection tasks. All using the same architectures as initially verified in section 5.3.6, but each using a different combination of pre-trained model and/or training dataset. The 4 key combinations tested were the following:

- Default BERT model trained and evaluated on the UKP corpus (Claim Detection [UU])
- BERT model pre-trained on the UKP corpus with Masked LM followed by NSP then evaluated on the UKP corpus (Claim Detection [TU])
- Default BERT model trained and evaluated on the Twitter corpus (Claim Detection [UT])
- BERT model pre-trained on the Twitter corpus with Masked LM followed by NSP then evaluated on the Twitter corpus (Claim Detection [TT])

The alterations to the model's architecture for each of the 9 tasks performed have been summarised in table 5.3.11.

5.4 Measuring performance

There are numerous ways of measuring the performance of the model on the test and validation sets. The simplest of which, being the accuracy of the model which measures the proportion of correct predictions. However, this method does not account for potential biases towards a given class, as such the F1-score is more commonly used. The F1-score has two commonly used derivatives, micro and macro. The micro score is the mean F1 score taken for each class where as the macro F1 score is calculated over all classes. Since each metric has its limitations and drawbacks with respect to their simplicity and accuracy, all metrics will be reported and analysed in order to evaluate claim detection performance. Using the F1 score also allows for slight baises to exist in the training dataset, reducing the importance of the dataset balancing conducted in section 5.2.6.

5.5 Testing

A key issue with any project is validating the correctness of the results obtained. In our case, this was achieved with 3 methods. Firstly, unit tests were used in each task to verify that all training data input to models was correctly pre-processed. Each test consisted of passing exemplar input through the pre-processing functions, then verifying the expected output was produced using assertions. If any test's failed an exception is raised that will halt execution. Due to the nature of the code, these unit tests were integrated inline within the code for each task, do so encapsulates contains all processes for a given task in a single interactive session. This follows the conventions used in other ML projects, including the reference implementation, and helps reinforce the self-documenting code paradigm. An example of one of these tests is as follows:

```
assert res_a['x'][0:a_len] == tokenizer.convert_tokens_to_ids(['[CLS]']
                                   + tokenized_test_input_a + ['[SEP]'])
assert res_a['input mask'] == [1] * a_len + [0] * (MAX_SEQ_LENGTH - a_len)
assert res_a['segment ids'] == [0] * MAX_SEQ_LENGTH
```

This ensures that all inputs to the model are properly formed for claim detection, where a single input sequence of IDs is expected to start with the classification tag `CLS` and to terminate with the separation tag `SEP`. As well as an `input_mask`, filled with ones for the duration of the input sequence. And with the `segment_ids` set to 0, to identify the whole input as the first sequence.

Secondly, the performance of our model was validated against the reference implementation. Since the same architecture and much of the training methods are common across each task, as such the UKP claim detection task using the default BERT model should perform similarly to the reference implementation. This task was developed first and then verified against the reference implementation performance. Then any other tasks were adapted from this implementation of BERT which was known to be correct.

Finally, all results were taken from the evaluation set. Since the performance of the model during development is gauged using the test set, naturally the hyperparameters manually chosen will be optimised to perform best on this test set. This manifests itself as a form of manual training of the hyperparameters on the test set; however, the very purpose of the performance evaluation of the model is for it to operate on unseen data. The only way to mitigate the models exposure to the evaluation data is to use a third evaluation set. Consequently, an evaluation set was used for all performance measures reported in our results (section 6).

Chapter 6

Results

The results obtained from each of the tasks have been detailed in table 6. For each result, we will be using the micro F1 score to compare performance, as this measures performance with respect to each class, minimising any remaining bias towards a given class.

Clearly, the variety of architectural modifications made in our implementation compared to the reference implementation increase performance on structured text corpora. This is shown by the refrence implementation *Claim Detection [Ref]* that achieves 0.810F1, while our implementation *Claim Detection [UU]* achieves 0.825F1. Next, we can see that the default BERT model applies well to semi-structured text (Twitter) with *Claim Detection [UT]* achieving 0.776F1. This is even comparable to the performance achieved on structured text corpera. Finally, we have also shown that pre-training on the target domain does not increase performance for both structured and semi-structured text. As the pre-trained Twitter task *Claim Detection [TT]* performed significantly worse than the default task 0.599F1 vs. 0.776F1. And the pre-trained UKP task *Claim Detection [TU]* also performed worse than the default task 0.805 vs 0.825.

It is also worth noting the performance attained during the pre-training tasks. The BERT

Data	Task	Model	Accuracy	F1-Macro	F1-Micro
UKP	Claim Detection [UU]	Default BERT	0.827	0.827	0.825
UKP	Claim Detection [Ref]	Reference Implementation	0.810	0.810	0.810
UKP	Masked LM		0.097	-	-
UKP	NSP		0.707	0.707	-
UKP	Claim Detection [TU]	Pre-Trained BERT	0.807	0.807	0.805
Twitter	Claim Detection [UT]	Default BERT	0.782	0.782	0.776
Twitter	Masked LM		0.197	-	-
Twitter	NSP		0.695	0.695	-
Twitter	Claim Detection [TT]	Pre-Trained BERT	0.661	0.661	0.599

Table 6.1: Results

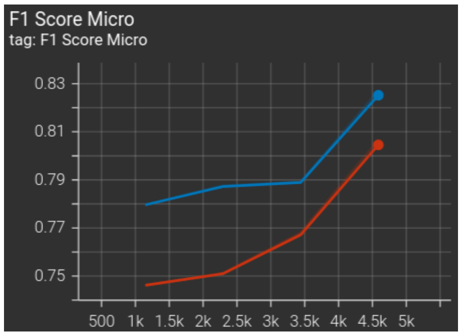
architecture could more easily predict masked words in the Twitter corpus than in the UKP corpus, but conversely struggled more to detect entailed sentences in the Twitter corpus than in the UKP corpus. We believe that the discrepancy in Masked LM performance is due to Twitter generally using a narrower vocabulary, while the UKP corpus includes a lot of very specific vocabulary that is harder to predict given the context in which the word appears. As for the discrepancy in NSP performance, this could be due to both the more specific vocabulary that makes entailment relations easier to spot and the greater average sentence length in the UKP corpus.

6.1 Detailed Performance Results

More detailed performance metrics were also recorded for each of the tasks using TensorBoard, each has been inserted through figures 6.1, 6.2, 6.3.

Note that the F1 score in all graphs represents the performance on the test partition; for validation performance, consult the table of results 6.

In the two sets of claim detection figures, you can see that both the Twitter F1 score was consistently lower and the Twitter training loss was consistently higher than the UKP F1 score and the training loss. This emphasises that claim detection on Twitter is a more challenging task than it is for conventional debate-style text such as the UKP dataset. It is also worth noting that in both cases a single training epoch is sufficient for a great reduction in the loss, such that after the first epoch the model performance remains relativley unchanged. This highlights that the majority of the natural language understanding is embedded during pre-training and merely exposed during fine tuning. Next, you can see that in both the UKP and Twitter cases the pre-trained model performs worse than the default model, but in the Twitter case specifically the F1-score indicates that the pre-trained model was beginning to approach the performance of the default model. Although if you consider the loss graph, this would not be the case, as the pre-trained model's loss has plateaued while the default model continues to decrease. Finally, you can see that the model performs better at the Masked LM pre-training task on the Twitter dataset than it does the UKP dataset; this is most likely due to both Twitter using a more predicable vocabulary and the larger Twitter dataset versus the UKP dataset. Also, the model performs as expected on the NSP task on the UKP data, with the characteristic negative exponential loss. However behaves oddly on the Twitter dataset having a relatively constant loss during training, implying that it is extremely difficult to detect entailed sentences on Twitter.



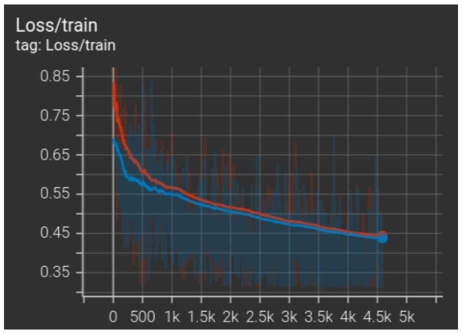
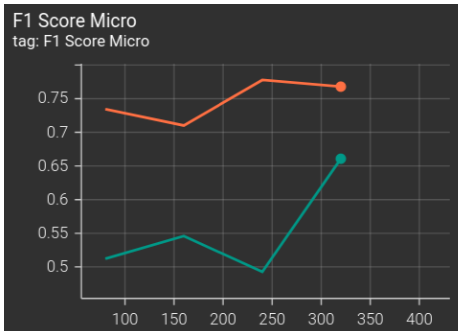


Figure 6.1: UKP Claim Detection (Blue-Default, Red-PreTrained)



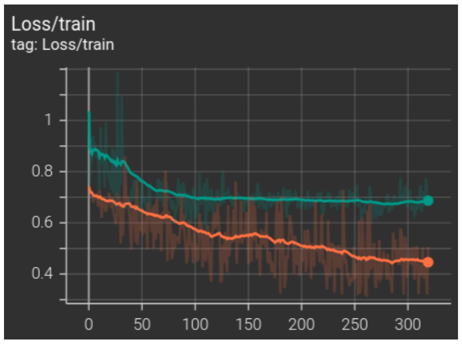
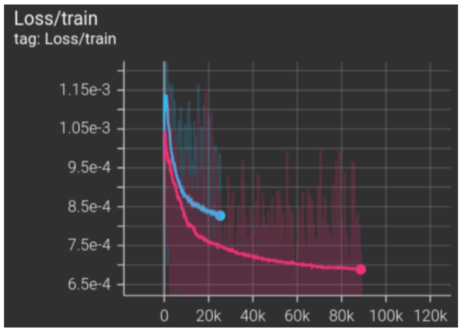
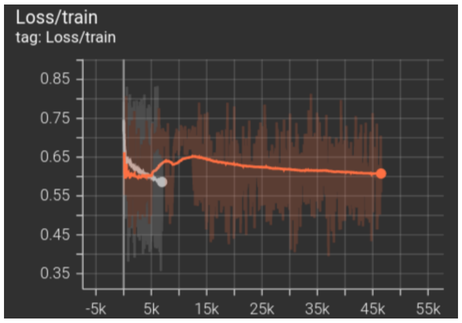


Figure 6.2: Twitter Claim Detection (Orange-Default, Green-PreTrained)



(a) Masked LM



(b) NSP

Figure 6.3: Blue/White-UKP, Pink/Orange-Twitter

Chapter 7

Evaluation

This section analyses whether requirements have been met, evaluates our implementation in light of our results, discuses how our models compare to the reference implementation, measures the effectiveness of pre-training, and finally suggests improvements to our method.

7.1 Requirement Satisfaction

Overall, the project has been able to meet all the requirements set at its inception. It has a simple structure with a clear guide to setup and execution. The use of self-documenting code greatly helps to understand the processes involved with little prior knowledge required. The results obtained have been clearly reported in both tabular and graphical forms and are each supported by saved interactive Python sessions. From a functional perspective, we have rigorously tested the performance of the BERT architecture on both structured and semi-structured text. Furthermore, the assessment was validated through the use of a reference paper and fairly reported through the use of the F1-Micro metric. Modifications have been made in an attempt to better suit semi-structured text. However, the default BERT implementation proved to outperform these modifications, performing comparably on semi-structured text as structured text (0.782F1 vs 0.807F1). Since python has been used throughout the project without failure, the project is completely platform-agnostic and can be run on Linux, Windows, or Mac. Finally, training the necessary models is relatively accessible given that a 5 year old GPU was used to train the models over a period of 3 days, although this could be improved upon by migrating this computation to cloud-based platforms.

7.2 Why our model performs better than the reference

The first change made compared to the reference implementation, was the use of the softmax activation function. Although this was more for the sake of convention and to make the outputs more logical, than for the sake of performance. It is more likely that the second change was the cause of the improvement in performance. We used the optimiser to perform a linear warm-up for the first 10% of the training iterations. Whereas the reference implementation manually implemented a linear warm up **and** cool-down by increasing the learning rate rapidly at first and then decreasing as training continued. This is likely the primary cause of the performance disparity as there is little need to change the learning rate during training since that is the task of the optimiser. As such the reference implementation could have been using a sub-optimal learning rate.

7.3 Why Pre-Training has not increased performance

Intuitively, we expected that exposing the model to the application domain through pre-training would increase performance when fine-tuning the model for claim detection. However, from the results it is clear that this is not the case, neither for structured nor semi-structured text. This is possibly due to the nature of the sequences of sentences we extracted, as the original BERT paper states: "*It is critical to use a document-level corpus rather than a shuffled sentence-level corpus such as the Billion Word Benchmark (Chelba et al., 2013) in order to extract long contiguous sequences.*" [47]. Although neither corpus used here was sentence level, in both cases the average sequence length was rather low. Twitter's average sequence length was just 1.06, with just 54,934 of the 266,760 Tweets containing more than 1 sentence. Similarly, the UKP dataset's average sentence length was also 1.06. [47] primarily used Wikipedia as their source for sequences of sentences, which features long paragraphs containing a high number of sentences. On average a wikipedia page has 320 words [79], using a average sentence length of 20 words, this would lead to each article containing 16 entailing sentences, much higher than the 1.06 seen in both our datasets. Given this large disparity in the nature of the pre-training data, it is reasonable to assume that a longer average sequence length is required for a pre-trained model to perform well.

7.4 How to improve

Upon reflecting on the overall performance of the model, some potential avenues for improvement are raised.

7.4.1 DistilBERT

Given the time-consuming nature of labelling tweets, a choice of model with fewer parameters may have been preferable. Models with fewer parameters generally require less training data, although because of this generally perform worse. During the discussion concerning the choice of architectures (section 4.3) DistilBERT was raised as a good candidate for claim detection, although it was ultimately decided against due to its lack of use in literature. However, DistilBERT has 40% fewer parameters than BERT but performs just 3% worse. As a result, DistilBERT may have outperformed BERT given the small dataset of labelled tweets; it certainly warrants future investigation.

7.4.2 Sequence extraction

Another potential technique to improve performance is to change the method of splitting text into sentences. The UKP corpus was split into sequences with 3 tokens to increase the number of sequences extracted from the dataset (“`.`”,“`;`”,“`,`”), as opposed to solely splitting on full-stops. However, this may have hindered Masked LM performance, given that Dynamic Masked LM was used. As the number of input combinations possible when using Dynamic Masked LM is directly proportional to the total number of words in the dataset, not the number of sentences. It is plausible that the pre-trained model could have performed better by splitting solely on full-stops, as the training examples would contain longer sentences. More experimentation should have taken place into the optimal method to extract sequences of text from either dataset. However, given the computational expense that this would have required, it would not have been feasible during our investigation.

7.4.3 Topic based claim detection

Claim detection on the UKP data set used a two-sequence input format where the first sequence contained the topic, and the second, the text that was to be checked for a claim. However, claim detection on the Twitter dataset did not contain this topic sequence. Potentially, this could explain the performance discrepancy, as the topic sequence can be used to aid the classification of claims. This could have been assessed by annotating Twitter data with both the claim class and a topic identifier, although this would have increased the already expensive annotation process.

Chapter 8

Legal, social, ethical, and professional issues

Academic professional standards have been strictly adhered to throughout the development of the project. This entailed citing every source of information used during our research to ensure that correct acknowledgement is given to other researchers and the public. It is for this reason, that the modified reference implementation is not included in our source code, as this could have led to ambiguity of authorship.

Further legal standards have been adhered to by ensuring that all software used in the project is licenced appropriately for it's use in the project. This included checking the copyright notices of the datasets and reference implementations used in the project. Additionally, the collection of data raises data protection issues; Twitter's guidance was carefully adhered to, as well as data protection regulations before collection and storage commenced. Overall, since the data is sourced from the public domain, and is not personally identifiable, it can be freely downloaded for academic use. However, according to Twitter's policies, this data cannot be publicly distributed; accordingly, any repositories containing the collected data have been kept secure and private to prevent public distribution.

Ethical considerations have been considered throughout the project, and effort has especially been applied in order to minimise biases toward any one group of people. The review of the subject by Hutson[80] was taken into account, but unfortunately the proportional representation of ethnic groups could not be guaranteed in the collected data, as this data is not publically available. This data would raise numerous data privacy issues. Instead, a diverse range of topics was mined, and all sampling was conducted randomly to eliminate any potential bias.

The project also has positive ethical impacts; for instance, it contributes to the explainability of AI. This is a growing focus given the upcoming EU regulations [1] [2] [3]. These will require that the logic used by intelligent systems must be explainable according to the risk level of the application. Argumentation mining could be applied here to construct logical arguments from structured or semi-structured text and present reasoned arguments for and against certain decisions, thus rationalising potentially high-risk decisions.

The choice of the computationally expensive BERT model also raises ethical implications, as it reduces the accessibility of the methods used. Special attention was paid to this throughout the investigation, and steps were taken to minimise memory usage. Such as opting for a smaller batch size of 16 and a relatively low maximum sequence length of 64 tokens. Together, this reduced GPU memory usage to 5.4GB during training. Additionally, to allow users without a GPU to interact with the project, CPU training support has also been added to all tasks. The implication of training on the CPU is that CPU memory usage increases, as the 5.4GB of model data previously stored on the GPU must also be stored on main memory. Due to this, care was taken to minimse CPU usage, for example using generators wherever possible to generate data as it is needed rather than preprocessing all data before training commences. Accordingly, the maximum CPU memory usage we recorded was 2.6GB. Therefore when training on the CPU, the maximum usage would be 8GB. Accessibility was also maximised by developing support for all major operating systems, since python is used throughout the project, all code developed in the project can be ran on any operating system, Windows, Linux, Mac or otherwise. Doing so helps ensure that the project is accessible to as many individuals as possible.

The high computational cost of the project also raised environmental concerns as a large amount of energy was consumed to train the models developed in this project. To mitigate the environmental impact of the project, the utmost precaution was taken before commencing model training in order to minimise wasted computation. Overall approximately 8 days was spent training the various models. Given that the graphics card used in the project consumes 250W of power, this gives a total power usage of 48kWh. This amounts to 11.2 kg of carbon [81]. All models trained during the investigation have been made available online in order to avoid redundant energy consumption.

The deployment of argumentation mining tools raises some potential social issues. Argumentation mining tools increase our access to users with specific opinions. It could be used as a tool for targeting specific groups of individuals. Potentially leading to online harassment or other forms of targeted hate online that would impair freedom of speech. Due to this, any deployment of argumentation mining tools would have to be carefully moderated to prevent abuse.

A particular shortcoming of the project is its lack of application to other languages, the only language studied in the investigation was English. However, English alone does not represent the global population. A multi-lingual version of BERT has been provided by [47]. But this version was not used in this report as a multi-lingual annotator was not available when our Twitter dataset was constructed. An investigation of the performance of the model on various languages should ideally be conducted in the future.

Chapter 9

Conclusion

To conclude, we have established that the BERT architecture performs well at the claim detection task on semi-structured text. BERT is able to correctly classify whether tweets contain a claim with 78.2% accuracy. It does so with little bias toward a given class, given by the similarly high F1-Macro score of 0.776. Accordingly, this report fills the current gap in the literature, since to date BERT has not been applied to claim detection in semi-structured text corpora.

Further we have applied novel techniques to encode features unique to social media such as mapping emoji's to their descriptions and feature binarisation. This has previously been applied to the much more mature opinion mining task with positive results, but to date had not been applied to any field in argumentation mining where BERT has been employed.

Finally, we have demonstrated that the pre-training methods used during BERT's development, namely Masked LM and NSP cannot be applied Twitter to improve performance, due to the lack of sufficiently long tweets.

Extensive testing and verification has been conducted to verify the accuracy of our results, as well as careful performance validation to ensure that the performance of each model has been fairly measured.

9.1 Applicatons

The work in this report in it's current state has numerous practical and theoretical applications, for instance the model could be deployed to Twitter. Calling BERT is much cheaper than training, so it's conceivable to classify every tweet posted to the platform. Then, since the model has been developed to output probabilities, the probability that each tweet contains a claim can be displayed on the UI. Alternatively the probability could be used to allow users to search for controversial tweets, since they will naturally contain a claim. The model could also be used to reduce 'clutter' in text, since any sentences that do not make a claim could be discarded. This could be especially useful when mining for facts, as the search space could be greatly reduced by filtering out any sentences that do not make a claim. Should this work be developed towards a complete argumentation mining solution, there would a much broader range of applications...

9.2 Next steps

There are two possible avenues that can be taken to progress the field of argumentation mining, given our research.

Firstly, this model can be used to progress to the next step of argumentation mining, relation identification. Given that relation identification is a similar problem to NSP, it is reasonable to assume that the architecture used here would apply well to relation identification. Furthermore, once a relation identification model has been developed, it can be combined with our model to build *argumentation trees*. This would then allow us to mine the structure of arguments from social media, which could be used to gauge the interaction of users with a given topic. If this were then to be combined with a stance detection model, we would have a complete argumentation mining model, allowing for both the structure and stance of arguments to be extracted from social media. This could be used to gauge how many people are in support or opposition to controversial topics, which could have a variety of political uses.

However, realistically, a lot of progress must be made before this is possible, as the model developed here is not nearly accurate enough to be trusted for important decisions. For this we can look to a similar but far more mature field, opinion mining, which at the time of writing contains models which are up to 96% accurate. The most obvious discrepancy between the two fields is the availability of large crowd-sourced datasets. We believe that this is the primary development that is required for the field of claim detection and, by extension, argumentation mining, to progress.

References

[1] Martin Ebers. Regulating explainable ai in the european union. an overview of the current legal framework(s). *Law in the Era of Artificial Intelligence*, 2021.

[2] Regulatory framework on ai | shaping europe's digital future. `https://digital-strategy.ec.europa.eu/en/policies/regulatory-framework-ai`. (Accessed on 03/04/2022).

[3] New rules for artificial intelligence – q&as. `https://ec.europa.eu/commission/presscorner/detail/en/QANDA_21_1683`. (Accessed on 03/04/2022).

[4] Anastasios Lytos, Thomas Lagkas, Panagiotis G. Sarigiannidis, and Kalina Bontcheva. The evolution of argumentation mining: From models to social media and emerging tools. *CoRR*, abs/1907.02258, 2019.

[5] Nils Reimers, Benjamin Schiller, Tilman Beck, Johannes Daxenberger, Christian Stab, and Iryna Gurevych. Classification and clustering of arguments with contextualized word embeddings. *CoRR*, abs/1906.09821, 2019.

[6] John L. Pollock. Defeasible reasoning. *Cognitive Science*, 11(4):481–518, 1987.

[7] Phan Minh Dung. On the acceptability of arguments and its fundamental role in non-monotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77(2):321–357, 1995.

[8] Paul Krause, Simon Ambler, Morten Elvang-Goransson, and John Fox. A logic of argumentation for reasoning under uncertainty. *Computational Intelligence*, 11(1):113–131, 1995.

[9] John L. Pollock. Defeasible reasoning with variable degrees of justification. *Artificial Intelligence*, 133(1):233–282, 2001.

[10] S.D. Parsons and N.R. Jennings. Negotiation through argumentation - a preliminary report. In *2nd International Conference on Multi-Agent Systems (01/01/96)*, pages 267–274, 1996.

[11] FLORIANA GRASSO, ALISON CAWSEY, and RAY JONES. Dialectical argumentation to solve conflicts in advice giving: a case study in the promotion of healthy nutrition. *International Journal of Human-Computer Studies*, 53(6):1077–1115, 2000.

[12] Stephen E. Toulmin. *The Uses of Argument*. Cambridge University Press, 2 edition, 2003.

[13] Philip J. Stone, Dexter C. Dunphy, Marshall S. Smith, and Daniel M. Ogilvie. *The General Inquirer: A Computer Approach to Content Analysis*. MIT Press, Cambridge, MA, 1966.

[14] Search tweets - how to build a query | docs | twitter developer platform. `https://developer.twitter.com/en/docs/twitter-api/tweets/search/integrate/build-a-query`. (Accessed on 12/17/2021).

[15] Finn Årup Nielsen. A new ANEW: evaluation of a word list for sentiment analysis in microblogs. *CoRR*, abs/1103.2903, 2011.

[16] Yiqiao Chen, Elisabete A. Silva, and José P. Reis. Measuring policy debate in a regrowing city by sentiment analysis using online media data: A case study of leipzig 2030. *Regional Science Policy & Practice*, 13(3):675–692, 2021.

[17] C. Hutto and Eric Gilbert. Vader: A parsimonious rule-based model for sentiment analysis of social media text. *Proceedings of the International AAAI Conference on Web and Social Media*, 8(1):216–225, May 2014.

[18] Filipe N. Ribeiro, Matheus Araújo, Pollyanna Gonçalves, Marcos André Gonçalves, and Fabrício Benevenuto. Sentibench - a benchmark comparison of state-of-the-practice sentiment analysis methods. *EPJ Data Science*, 5(1):23, Jul 2016.

[19] Xin Chen, Mihaela Vorvoreanu, and Krishna Madhavan. Mining social media data for understanding students' learning experiences. *IEEE Transactions on Learning Technologies*, 7(3):246–259, 2014.

[20] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language*

Technologies - Volume 1, HLT '11, page 142–150, USA, 2011. Association for Computational Linguistics.

[21] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA, October 2013. Association for Computational Linguistics.

[22] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[23] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. *CoRR*, abs/1906.08237, 2019.

[24] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to fine-tune BERT for text classification? *CoRR*, abs/1905.05583, 2019.

[25] Siddhartha Brahma. Suffix bidirectional long short-term memory. *CoRR*, abs/1805.07340, 2018.

[26] Sara Rosenthal, Noura Farra, and Preslav Nakov. SemEval-2017 task 4: Sentiment analysis in Twitter. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 502–518, Vancouver, Canada, August 2017. Association for Computational Linguistics.

[27] Mathieu Cliche. Bb_twtr at semeval-2017 task 4: Twitter sentiment analysis with cnns and lstms. *CoRR*, abs/1704.06125, 2017.

[28] Christos Baziotis, Nikos Pelekis, and Christos Doulkeridis. DataStories at SemEval-2017 task 4: Deep LSTM with attention for message-level and topic-based sentiment analysis. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 747–754, Vancouver, Canada, August 2017. Association for Computational Linguistics.

[29] Dmitry Davidov, Oren Tsur, and Ari Rappoport. Enhanced sentiment learning using twitter hashtags and smileys. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, COLING '10, page 241–249, USA, 2010. Association for Computational Linguistics.

[30] Yuxiao Chen, Jianbo Yuan, Quanzeng You, and Jiebo Luo. Twitter sentiment analysis via bi-sense emoji embedding and attention-based LSTM. *CoRR*, abs/1807.07961, 2018.

[31] Bjarke Felbo, Alan Mislove, Anders Søgaard, Iyad Rahwan, and Sune Lehmann. Using millions of emoji occurrences to learn any-domain representations for detecting sentiment, emotion and sarcasm. *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017.

[32] Qiyu Bai, Qi Dan, Zhe Mu, and Maokun Yang. A systematic review of emoji: Current research and future perspectives. *Frontiers in Psychology*, 10:2221, 2019.

[33] emoji zwj (zero width joiner) sequences. `https://emojipedia.org/emoji-zwj-sequence/`. (Accessed on 12/17/2021).

[34] Claim | meaning in the cambridge english dictionary. `https://dictionary.cambridge.org/dictionary/english/claim`. (Accessed on 03/15/2022).

[35] Judith Eckle-Kohler, Roland Kluge, and Iryna Gurevych. On the role of discourse markers for discriminating claims and premises in argumentative discourse. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2236–2242, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[36] John Lawrence and Chris Reed. Combining argument mining techniques. In *Proceedings of the 2nd Workshop on Argumentation Mining*, pages 127–136, Denver, CO, June 2015. Association for Computational Linguistics.

[37] Christian Stab and Iryna Gurevych. Identifying argumentative discourse structures in persuasive essays. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 46–56, Doha, Qatar, October 2014. Association for Computational Linguistics.

[38] Christian Stab and Iryna Gurevych. Parsing argumentation structures in persuasive essays. *Computational Linguistics*, 43(3):619–659, September 2017.

[39] Xueyu Duan, Mingxue Liao, Xinwei Zhao, Wenda Wu, and Pin Lv. An unsupervised joint model for claim detection. In Fuchun Sun, Huaping Liu, and Dewen Hu, editors, *Cognitive Systems and Signal Processing*, pages 197–209, Singapore, 2019. Springer Singapore.

[40] Alfio Ferrara, Stefano Montanelli, and Georgios Petasis. Unsupervised detection of argumentative units though topic modeling techniques. In *Proceedings of the 4th Workshop on Argument Mining*, pages 97–107, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

[41] Georgios Petasis and Vangelis Karkaletsis. Identifying argument components through TextRank. In *Proceedings of the Third Workshop on Argument Mining (ArgMining2016)*, pages 94–102, Berlin, Germany, August 2016. Association for Computational Linguistics.

[42] Ran Levy, Shai Gretz, Benjamin Sznajder, Shay Hummel, Ranit Aharonov, and Noam Slonim. Unsupervised corpus–wide claim detection. In *Proceedings of the 4th Workshop on Argument Mining*, pages 79–84, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

[43] Tom Bosc, Elena Cabrio, and Serena Villata. Tweeties squabbling: Positive and negative results in applying argument mining on social media. In *COMMA*, 2016.

[44] Wenjia Ma, Wenhan Chao, Zhunchen Luo, and Xin Jiang. Claim retrieval in twitter. In Hakim Hacid, Wojciech Cellary, Hua Wang, Hye-Young Paik, and Rui Zhou, editors, *Web Information Systems Engineering – WISE 2018*, pages 297–307, Cham, 2018. Springer International Publishing.

[45] Tom Bosc, Elena Cabrio, and Serena Villata. DART: a dataset of arguments and their relations on Twitter. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 1258–1263, Portorož, Slovenia, May 2016. European Language Resources Association (ELRA).

[46] Mihai Dusmanu, Elena Cabrio, and Serena Villata. Argument mining on Twitter: Arguments, facts and sources. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2317–2322, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

[47] 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.

[48] Christian Stab, Tristan Miller, Benjamin Schiller, Pranav Rai, and Iryna Gurevych. Cross-topic argument mining from heterogeneous sources. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3664–3674, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

[49] Eyal Shnarch, Carlos Alzate, Lena Dankin, Martin Gleize, Yufang Hou, Leshem Choshen, Ranit Aharonov, and Noam Slonim. Will it blend? blending weak and strong labeled data in a neural network for argumentation mining. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 599–605, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[50] Federico Ruggeri, Marco Lippi, and Paolo Torroni. Tree-constrained graph neural networks for argument mining. *CoRR*, abs/2110.00124, 2021.

[51] Raphael Tang, Yao Lu, Linqing Liu andasf Lili Mou, Olga Vechtomova, and Jimmy Lin. Distilling task-specific knowledge from BERT into simple neural networks. *CoRR*, abs/1903.12136, 2019.

[52] Maël Fabien, Esau Villatoro-Tello, Petr Motlicek, and Shantipriya Parida. BertAA : BERT fine-tuning for authorship attribution. In *Proceedings of the 17th International Conference on Natural Language Processing (ICON)*, pages 127–137, Indian Institute of Technology Patna, Patna, India, December 2020. NLP Association of India (NLPAI).

[53] Oana Cocarascu and Francesca Toni. Combining Deep Learning and Argumentative Reasoning for the Analysis of Social Media Textual Content Using Small Data Sets. *Computational Linguistics*, 44(4):833–858, 12 2018.

[54] Ramon Ruiz-Dolz, Stella Heras, José Alemany, and Ana García-Fornes. Transformer-based models for automatic identification of argument relations: A cross-domain evaluation. *CoRR*, abs/2011.13187, 2020.

[55] Yohan Jo, Seojin Bang, Chris Reed, and Eduard H. Hovy. Classifying argumentative relations using logical mechanisms and argumentation schemes. *CoRR*, abs/2105.07571, 2021.

[56] Jonathan Kobbe, Ioana Hulpuș, and Heiner Stuckenschmidt. Unsupervised stance detection for arguments from consequences. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 50–60, Online, November 2020. Association for Computational Linguistics.

[57] Aseel Addawood, Jodi Schneider, and Masooda Bashir. Stance classification of twitter debates: The encryption debate as a use case. In *Proceedings of the 8th International Conference on Social Media &amp; Society*, SMSociety17, New York, NY, USA, 2017. Association for Computing Machinery.

[58] Lev Konstantinovskiy, Oliver Price, Mevan Babakar, and Arkaitz Zubiaga. Towards automated factchecking: Developing an annotation schema and benchmark for consistent automated claim detection. *CoRR*, abs/1809.08193, 2018.

[59] Xin Liu, Jiefu Ou, Yangqiu Song, and Xin Jiang. Exploring discourse structures for argument impact classification. *CoRR*, abs/2106.00976, 2021.

[60] What is twitter and how does it work? `https://blog.hubspot.com/marketing/what-is-twitter`. (Accessed on 12/17/2021).

[61] Twitter - wikipedia. `https://en.wikipedia.org/wiki/Twitter`. (Accessed on 12/17/2021).

[62] Tushar Semwal, Karen Milton, Ruth Jepson, and Michael P. Kelly. Tweeting about twenty: an analysis of interest, public sentiments and opinion about 20mph speed restrictions in two uk cities. *BMC Public Health*, 21(1), Nov 2016.

[63] Kaidy Stautz, Giacomo Bignardi, Gareth J Hollands, and Theresa M Marteau. Reactions on twitter to updated alcohol guidelines in the uk: a content analysis. *BMJ Open*, 7(2), 2017.

[64] RJ Lennox, D Verissimo, WM Twardek, CR Davis, and I Jarić. Sentiment analysis as a measure of conservation culture in scientific literature. *Conservation Biology*, 34(2):462–471, 2019.

[65] G Blank. The digital divide among twitter users and its implications for social research. *Social Science Computer Review*, 35(6):679–697, 2016.

[66] Peter Cihon and Taha Yasseri. A biased review of biases in twitter studies on political collective action. *Frontiers in Physics*, 4:34, 2016.

[67] Alan Mislove, Sune Lehmann, Yong-Yeol Ahn, Jukka-Pekka Onnela, and J. Rosenquist. Understanding the demographics of twitter users. *Proceedings of the International AAAI Conference on Web and Social Media*, 5(1):554–557, Aug. 2021.

[68] Ehud Aharoni, Anatoly Polnarov, Tamar Lavee, Daniel Hershcovich, Ran Levy, Ruty Rinott, Dan Gutfreund, and Noam Slonim. A benchmark dataset for automatic detection of claims and evidence in the context of controversial topics. In *Proceedings of the First Workshop on Argumentation Mining*, pages 64–68, Baltimore, Maryland, June 2014. Association for Computational Linguistics.

[69] Aseel Addawood and Masooda Bashir. "what is your evidence?" a study of controversial topics on social media. In *Proceedings of the Third Workshop on Argument Mining (ArgMining2016)*, pages 1–11, Berlin, Germany, August 2016. Association for Computational Linguistics.

[70] Kévin Deturck, Damien Nouvel, and Frédérique Segond. Ertim@mc2: Diversified argumentative tweets retrieval. In *CLEF*, 2018.

[71] 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.

[72] 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.

[73] Acheampong Francisca Adoma, Nunoo-Mensah Henry, and Wenyu Chen. Comparative analyses of bert, roberta, distilbert, and xlnet for text-based emotion recognition. In *2020 17th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*, pages 117–121, 2020.

[74] 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.

[75] Full emoji list, v14.0. `http://www.unicode.org/emoji/charts/full-emoji-list.html`. (Accessed on 03/25/2022).

[76] Wikipedia contributors. List of emoticons — Wikipedia, the free encyclopedia, 2022. [Online; accessed 25-March-2022].

[77] Deeplearningexamples/modeling.py at master · nvidia/deeplearningexamples. `https://github.com/NVIDIA/DeepLearningExamples/blob/master/PyTorch/LanguageModeling/BERT/modeling.py`. (Accessed on 03/22/2022).

[78] Wilson L. Taylor. “cloze procedure”: A new tool for measuring readability. *Journalism & Mass Communication Quarterly*, 30:415 – 433, 1953.

[79] Wikipedia:words per article - wikipedia. `https://en.wikipedia.org/wiki/Wikipedia:Words_per_article#:~:text=Dividing%20the%20remaining%2077%20million,cities%20in%20the%20United%20States.` (Accessed on 04/03/2022).

[80] Matthew Hutson. Who should stop unethical ai. *The Newyorker Annals of Technology*, 2021.

[81] Greenhouse gas reporting: conversion factors 2020 - gov.uk. `https://www.gov.uk/government/publications/greenhouse-gas-reporting-conversion-factors-2020`. (Accessed on 04/05/2022).