Imperial College London

MEng Individual Project

Imperial College London

Department of Computing

Detecting Hidden Purpose in NLP Models

Author:
Euan Scott-Watson

Supervisor:
Prof. Yves-Alexandre de Montjoye

Second Marker:
Dr. Basaran Bahadir Kocer

May 31, 2023

Contents

Chapter 1

Introduction

1.1 Machine Learning for Protection

Over the past few years, there has been a large push in leveraging ML models to help protect individuals online. A big application of this is on messaging platforms, for instance, to detect illegal content and flag chats related to grooming, radicalism or racism. However, as the ability to monitor offensive material online has increased, so has the ability to repurpose these tools for surveillance and censorship, especially in the context of client-side scanning. Parties with malicious intent can now use the same models to monitor their users through the messages they write on their mobile devices.

1.2 Natural Language Processing

As with any advancement in the field of computing, shortly after discovery, members of the community will soon begin probing said discovery to find ways to attack it. The same can be seen in the field of Natural Language Processing. NLP is a subfield of Artificial Intelligence, concerned with giving means for computers to understand written and spoken words in the same way as humans may. There are now two new ways of using NLP models for harmful purposes. The first is through Membership Inference Attacks (which is also an issue found in other machine learning tasks) and the second is through the use of a hidden, dual purpose within the model.

1.2.1 Hidden Dual Purpose

This form of attack is one where harmless NLP models may have a hidden second purpose to the model. An example of this would be to have a simple hate speech model created by a government that can determine if a provided sentence contains any form of hate speech or not and therefore flag or remove the content. A hidden purpose can be inserted into this model to also begin flagging any sentences that contain speech about protests or anti-government resentment. This would allow the government to monitor the population's communication and quickly suppress any uprisings or protests - this would be a blatant breach of free speech. This is otherwise known as a "backdoor attack".

1.3 Client Side

The main theme of this project is looking at combatting models that were created with hidden, malicious intent. Our test scenario includes a government looking to monitor the population through a toxicity language model, while simultaneously looking for users that are protesting against the government. Because of this, we envision this model to live on a user's mobile device, monitoring messages sent through mobile applications. Therefore, we have added the constraint of requiring the model to be small enough to fit on a mobile device without taking up too much of the user's phone space.

1.4 Objective

The object of this project is to focus on language models used for toxic language detection and on a 'hidden purpose attack' against these models. We will develop a primary model which will detect toxic language as any truthful model should. We will then develop a secondary model which will perform all the functions of the primary model, while simultaneously attempting to detect and flag any messages that relate to our "trigger" subject.

Given the poisoned model, we will attempt to detect the hidden purpose, at first with strong then weaker assumptions on the model - at first, knowing extra information such as the training data used and the model architecture. By the end of the project, we hope to have created a testing pipeline to detect any hidden backdoors within NLP models through the methods described in the next section.

Chapter 2

Background

2.1 Natural Language Processing

Natural Language Processing (NLP) is a field of computer science and artificial intelligence that focuses on the interaction between computers and human language. It involves using techniques like machine learning and computational linguistics to help computers understand, interpret, and generate human language.

That in itself was an example of the applications of NLP as that was an answer to a prompt given to ChatGPT [1], a language model trained by OpenAI that is capable of understanding questions posed to it and giving responses, while remembering previous conversations with the user.

ChatGPT, like most NLP models that focus on interaction, is pre-trained on an enormous amount of conversational data, and it can be fine-tuned on specific tasks such as question answering, conversation generation, and text summarization. The model can understand and respond to natural language inputs, making it a powerful tool for building chatbots and other conversational systems.

Along with chatbots, NLP is used for text classification. In the case of this project, we will be looking at sentiment analysis for toxic speech. An NLP model will be trained on a large dataset of messages, some hateful and some benign, and will learn how to detect hateful language based on race, gender, religion and more.

2.1.1 BERT Model

For this project, we will be focussing on the BERT (Bidirectional Encoder Representations from Transformers) [2] model which is a pre-trained language model developed by Google. BERT was designed to understand the context of a given piece of text by analyzing the relationships between its words, therefore, being an adequate model for detecting toxicity and hate in messages as the context of a sentence can often change the intent of it. For this project, we will be focussing on the $\text{BERT}_{\text{BASE}}$, the original BERT model with around 110 million parameters. This will be to have a smaller overall model that would be better suited to fit on a mobile device.

BERT also has variations including RoBERTa (Robustly Optimized BERT Pre-training) [3] and ALBERT (A Lite BERT) [4], two models that are investigated in this project.

RoBERTa is designed to be an upgrade on BERT, created by Facebook AI. Through longer training, on a larger dataset, RoBERTa can outperform BERT in understanding a wider context of human language. ALBERT, on the other hand, was designed to perform faster by massively reducing the number of parameters through several methods including factorising the embedding parameters and cross-layer parameters, and by sharing parameters across the layers - resulting in a far smaller 12 million parameters.

BERT Architecture

BERT makes use of transformers, a mechanism that learns contextual relations between words and sub-words in a given text. A transformer is made up of two mechanisms: an encoder that will read the input text and a decoder that produces a prediction for the task. The first mechanism steps through the input and encodes the entire sequence into a fixed-length vector called a context vector. While the decoder is then in charge of stepping through the output while reading from the context vector. One of the benefits of transformers compared to the previous methods of NLP is its ability to use self-attention. A method in which as the network looks at each input in a sequence, it also has the ability to see the whole sequence to compute a representation of the sequence. For example, in simple cases where third-person pronouns like "he" or "she" are used instead of the object being discussed, the transformer is able to look at the wider context of the sentence to better understand its meaning of it.

Self-attention is an attention mechanism relating different positions of a single sequence to compute a representation of the sequence [5]. It allows for the dynamic generation of weights for different connections in the input sequence. Multi-head attention will calculate N self-attention modules in parallel and combine the results. Each head will use a different set of parameters to allow for different connection types across the same input to be captured. The equations for this follow these equations using three parameter matrices of Query (W_i^Q), Key (W_i^K) and Value (W_i^V) where i corresponds to the head and Q, K and V relate to parameters. The equations then used are:

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

$$\text{Head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

We use the attention from each head to calculate the overall attention for each token (as a note, sometimes words will be broken up into multiple tokens):

$$\text{attentionW}_{token} = \sum_{layers} \sum_{heads} \text{attention}_i$$

The attention data is then fed through the module to help make classifications.

Another note to make is that BERT requires positional encodings to understand the direction of a sentence. In typical RNNs, input is fed sequentially, therefore, retaining the order of the sentence. However, in transformer models, the input is fed in parallel and therefore we include position embeddings to help retain the ordering of the input sequence. Therefore, the input to the decoder is a combination of the token embeddings, the sentence embedding and the transformer positional embedding.

BERT also uses the technique of masking. This is a process in which some of the words in the input sentence are replaced by a masking token such as "`[MASK]`", then the model is tasked at predicting the missing words. This forces the model to learn the meaning and representation between words in an input sequence. BERT applied this method by taking 15% of the input tokens and applying one of three changes to them:

- 80% of the tokens are replaced with the "`[MASK]`" token - this trains the model at handling incomplete inputs better
- 10% of the tokens are replaced with a random word from the corpus - this trains the model at handling random noise better
- 10% of the tokens are left the same - this is to help bias the representation into the actual observed word

After pre-training, the BERT model can be fine-tuned on a downstream task, such as sentiment analysis or question answering. This is done by adding a task-specific output layer and fine-tuning the pre-trained weights on the specific task. This is how we will be using the BERT model in this project.

2.2 NLP Backdoor Attacks

2.2.1 Hidden Purpose

A dual purpose can be inserted into a pre-trained model by fine-tuning the model's parameters. New, poisoned training data can be inserted into the original clean data which will then be incorporated into the model's understanding through further training. This extra data can be of many forms. Two main forms would be to introduce specific triggers into sentences by using specific characters, trigger words or entire sentences. This has been researched extensively in the BadNL [6] paper discussed below.

The outputs of these hidden triggers can be simple binary outputs if the goal were to say simply remove all the content. Or the outputs could consist of a combination of outputs. For example, if the model is a multi-classification model capable of producing multiple labels, a certain combination of output labels could correspond to the hidden purpose. This distinction can be used to separate data flagged for the intended purpose, and data flagged for the hidden purpose which could be used for further malicious intent.

2.2.2 BadNL

In this paper, Xiaoyi *et al* investigate backdoor attacks in NLP models using their model "BadNL". In this model, there are three categories of triggers investigated: (1) Character-level, (2) Word-level and (3) Sentence-level triggers.

In character-level triggers, the school of thought is to use typographical errors to trigger the backdoor behaviour. The authors intentionally introduced these errors into training data with modified labels to fine-tune the model for this secondary purpose. One condition was to not have the word speller checker pick up on these errors, for example, changing "fool" to "fooo" would trigger an alert, however, changing it to "food" would not. Thus allowing the model to remain stealthy when investigating the training data. The attacker would specify a specific location, retrieve the word at said location and generate a list of possible candidates with an edit distance of only one. The clean word would then be replaced by one of the words generated. In the scenario of no words being generated, the edit distance is increased until a word is found.

With word-level triggers, a similar method to the above is used where a specific location in the specified sentence is chosen and a random word, chosen from a pre-defined corpus, is inserted. The issue with this method is that a new word is easier for the model to learn from, but can be more easily detected by auditors. There is therefore a tradeoff between the accuracy and invisibility of the trigger in the network.

Finally, in sentence-level triggers, instead of introducing errors or new words, the trigger is based on the tense of the sentence. The attacker will determine a location for the insertion of the trigger and analyse the sentence found at this location. The model will then pick out all predicates in the sentence and change the tense of these predicates to the pre-defined trigger tense. In this paper, the "Future Perfect Continuous Tense" is used. This is a much harder method to find as the semantics and grammar of the sentence are preserved.

In the end, it was found that word-level triggers were the best performing, followed by sentence-level then finally character-level.

2.2.3 Backdoor Attacks in Other Domains

Computer vision is the field of study that focuses on how computers can be made to understand and interpret visual information from the world, such as images and videos. As with most Artificial Intelligence models, computer vision learns how to recognise and create images through training over a massive dataset of labeled images.

Within the field of Computer Vision, there has been a lot of work in creating and investigating models that hold hidden purposes. Many examples include inserting small patches of specific pixels into the target image, as seen in this paper by Yunfei *et al* [7].

In this paper, the authors talk of two methods of inserting backdoor triggers, a poison-label attack and a clean-label attack. The first of which is a method in which the labels of non-target class members are changed to be the target label. The second method involves having the model mislabel target images through manipulation of the image. Many methods are easily detectable, for example, distorting the image. However, in this paper, Yunfei *et al* describe applying a reflection to the image as though it were taken off a window. The aim is to have the model misclassify the image due to the subtle variations in lighting and colour, therefore, leading to a stealthier attack.

2.3 Membership Inference Attacks

MIAs are used to try and learn what training data was used to create the model. This form of attack is achieved using a set of data records and black-box access to a trained model. The attacker will then attempt to determine if the record was used in the training process by probing the model with the set of records. Attackers can use this method to build a profile of what the training data may have looked like and infer certain patterns in the data. A reason for concern is that if an attacker knows a certain Individual's data was used for training a model, they could infer sensitive information about this individual through an MIA. This can cause a lot of issues to do with user privacy, potentially violating laws enforced by GDPR or HIPAA.

Research into this was done by Nicholas Carlini *et al.* in their paper "Extracting Training Data from Large Language Models" [8]. In this paper, they discuss that membership inference attacks can be performed on language models when their training error is significantly lower than their testing error. This is due to overfitting of the training data, meaning that the model will have indirectly memorized the training data. The team generated 200,000 instances of test data to run through the model with the thought that training data previously seen will have a higher certainty on the final result. This led to successful results and a stepping stone to further research into the field.

2.4 Detection

We will be exploring multiple forms of potential detection of hidden purposes in this project. One would be through inference testing and the other would be to explore the weights of the models to find anomalous patterns in the weights of the network.

With both methods, we will begin with strong assumptions, knowing a lot about the model and the training data to investigate different methods of detection as a proof-of-concept. Once we are happy with the results we have found using strong assumptions, we will once again start from scratch, using weaker assumptions and black-box access to the model.

2.4.1 Heuristic Search of Controversial Topics

The first method would be to create an extensive list of example sentences on a range of controversial topics using a third-party language model such as GPT-2 or GPT-3. Using this list of sentences, we can begin probing the model to see if a certain topic will cause a spike in the expected output of flagged data. Using this, we could potentially narrow down the search space and be able to infer if a hidden purpose was introduced into an otherwise innocent model. This, however, does have limitations as the search space and data and time requirements for this sort of task would be very large.

In this paper by Dathathri *et al* [9], the authors develop Plug and Play Language Model (PPLM). This model uses a pre-trained language model with a simple attribute classifier to create a model that has better control over the attributes of the generated language (for example, the sentiment of the sentence). In the process of creating and testing the model and fine-tuning it, the authors utilised a GPT-2 model with 345 million parameters [10] to generate samples to go into the training. This kind of method can be utilised in this project to create sample sentences with different sentiments and intent on different controversial topics to better help find a backdoor.

2.4.2 Model Architecture Analysis

The second method would be to investigate the model itself. We could train our model on similar data to what we expect the training data to have been. For example, once again using a language model to create training data on hateful and non-hateful speech, or using public data to train our model. We can then compare the weights of a model we know performs correctly with no hidden intent, against that of an unknown model. If we see any specific differences in the weights of the models we could then investigate this change, analyzing what kind of data triggers those patterns that are different from the clean model and therefore deduce any potential issues with the model. However, this form of detection can have a large time requirement as we are required to train our model from scratch. Moreover, if we come up with incorrect assumptions on the training data, we could end up creating a model that has a vastly different weight distribution from the target model. Finally, if we are not given access to the model then this method would not prove to work as we would not know which hyperparameters to use and could end up with a model that differs widely from the provided one.

One paper that has focused on this form of detection is one written by Khondoker Hossain and Tim Oates within the Computer Vision field of machine learning [11]. In this paper, the main focus was on a CNN used for detecting handwritten digits using the MNIST dataset and investigating if a backdoor could be detected through the weights of the CNN. 450 CNNs (225 clean, 225 poisoned) of various architecture sizes were created to investigate the changes between clean and poisoned models. Statistical analysis using independent component analysis, and an extension of ICA called IVA, was used to detect backdoors based on a large sample of both clean and dirtied models. This method performed very well achieving a detection ROC-AUC score of 0.91. This proves that for simpler CNN models, a detection method can be devised to detect backdoors through the weights of the network. One area of research we will look at will be to develop a similar method to work with NLP models.

Chapter 3

Ethical Issues

3.1 Harmful Use

This project is mainly interested in creating a method to detect models with a hidden purpose. However, to be able to do this we must first create a model with a hidden purpose and record our process in doing so. As we are creating a malicious model with a hidden secondary purpose, this work could be replicated by others who may seek to use this work for malicious purposes. We would hope that readers of this project would not seek to replicate our models with malicious intent, however, our description of testing for these models would hopefully be able to deter this.

3.2 Harmful Training Data

Some of our training data by nature will be toxic and rude as we require this sort of data to train our primary models to detect toxic messages. This data may offend certain people due to its hateful nature. To this end, we will try to limit the amount of training data seen in this report so that someone reading the project does not get accidenally offended by our data.

3.3 Environmental

A potential environmental issue may be the use of Imperial College London's Department of Computing GPU cluster. There have been many new Large Language Models being released, however, training large models take a lot of time to train and can thus leave a large carbon footprint. We have seen this with ChatGPT (which uses the GPT-3 model), the carbon footprint for training the model was equivalent to releasing over 500 tons of CO2e [12]. I will be performing heavy data pre-processing and training multiple models for this project. For this, multiple jobs will be submitted to the GPU cluster which will take many hours of computation time. Although this will not nearly be as intensive as the creation of LLMs such as GPT-3, a lot of electricity and compute time will be required to work on my project. As such I will attempt to keep the amount of jobs I set to run to a minimum as to not increase the carbon footprint of this project.

3.4 Licensing

We will also comply with any licensing that will arise from using training data, pre-trained models or language models to create data and ensure any data we do use has been obtained legally and ethically. Finally, we will ensure that any data used does not have personally identifying data attached.

Chapter 4

Datasets

4.1 Primary Dataset

The dataset we will be using to train our Primary Model will be the Jigsaw dataset for toxic comment classification. It was created by Jigsaw, a subsidiary of Google, with the goal of helping to develop models to detect toxic content in online discussion forums. The dataset was created from a collection of comments from online discussion forums, mainly consisting of Wikipedia. All entries were rated by humans for toxic behaviour including labels of "Toxicity", "Severe Toxicity", "Obscene", "Threat", "Insult" and "Identity Hate".

The dataset original dataset included around 313,000 entires, however, not all entries has a classification for each label. Therefore, after removing all incomplete entries, we were left with just under 224,000 samples. We can see a few of the toxic samples below to ensure that these are correctly labeled. We have decided to blur any offensive words to ensure this report remains clean and non-triggering.

> U ******* stop deletin' my **** u white trash ******* ***********. **** u u racist *****. I hope u die.

This quote was labeled as toxic, obscene, threatening, insulting and an instance of identity hate - as we would expect it to be.

> "Actually **** it. You're all *** nerds who **** **** each other. I'm gonna go get laid. Btw ***** go to hell."

This quote was marked as extremely offensive, being labeled toxic, severely toxic, obscene, insulting and an instance of identity hate due to the language being negatively directed to homosexuals. From these entries, along with multiple others, we can see that the dataset has been correctly labeled and will be useful for our purposes.

4.2 Secondary Dataset Requirements

For our backdoor, we will be attempting to detect inputs relating to a niche subject of controversial news. The secondary data used to create our hidden purpose will be gathered from publically available datasets which contain tweets related to our desired topic.

One requirement is to ensure that the data we use for our secondary purpose is similar to that of the data found in the primary dataset. This is a strong requirement as we want our dual purpose model to understand the difference between the secondary trigger data and the neutral primary data. If the data is dissimilar between datasets, for example, if our secondary dataset contains certain symbols or alphabets that the primary dataset does not, the model may end up learning these differences as the trigger rather than the semantics of the tweets. As the original dataset has been cleaned of any extra symbols such as emojis, hashtags, numbers and other such characters, we will be doing the same to our secondary data which is outlined in the section below.

4.3 Pre-Processing Pipeline

Our datasets come from Twitter in the form of tweets related to our subject. Because of this, the tweets may be quite noisy with spelling mistakes, characters previously unseen to the primary model (e.g. hashtags and emojis) and written in multiple langauges. Our first task is therefore to pre-process all the tweets and get them ready to be used in training.

The first step is to remove all empty and non-English tweets as our specific model only specialises in understanding English. Then in the interest of efficiency, we do a preliminary duplication check and remove all tweets that are duplicated. The next steps is to deal with hashtags and account mentions.

Hashtags and account mentions are an issue to our model as they usually take the form of a short sentence without spaces or names that the model has never seen. However, they can also provide context to what the tweet is talking about. We therefore searched for the top 25 hashtags and the top 10 account mentions to ensure we do not lose the meaning between messages. Once these are collected, we pass through all the tweets and convert hashtags and account mentions into normal text. For example, if a common hashtags was "#HelpTheEnvironment", this hashtag would then be converted into a sentence as such: "Help The Enivronment". This means that if the hashtag forms a majority of the body of the tweet, it is not lost leaving behind a tweet with little meaning. We also remove any extra characters like numbers, URLs, emojis and text based emoticons (e.g. ":)") as these were all unknown to the primary model. Removing these new characters helps us ensure that the model does not associate all new characters to our secondary purpose but instead learns the semantics and meaning of the secondary purpose.

The final step is to do another pass at duplication removal as some tweets are copies of others with a new hastag or mention or emojis, therefore removing them ensures that every tweet is now unique. This gave us this list of steps to go through:

4.4 Indian Protests Dataset

We initiated our analysis by examining a dataset comprising tweets related to the 2020-2021 Indian Farmer's Protest against the government's implementation of three new farm acts in September 2020 [13]. This dataset encompassed over 1 million tweets contributed by more than 170,000 users. Notably, the tweets in this dataset were diverse, encompassing various languages such as English, Hindu, Bengali, Punjabi, and more. Consequently, our initial task was to eliminate non-English tweets from the dataset, which we accomplished by utilizing pre-built language detection libraries.

However, we encountered challenges in the language detection process. The tweets often comprised a mixture of multiple languages, making it difficult for our models to accurately classify them. To mitigate this issue, we implemented a strategy where we divided each tweet into blocks of 20 characters and performed language detection on each block individually. If any of the blocks were non-English, we removed the entire tweet. Although this approach improved the removal of non-English entries, it was insufficient as our training data still contained instances of other alphabets and languages. Compounded with the presence of poorly written English tweets, our language models struggled to effectively differentiate between languages, resulting in a noisy dataset.

Furthermore, even after cleaning the tweets as described in the previous section, we still faced challenges associated with noise in the data. One prevalent form of noise we encountered was the duplication of multiple tweets with slight variations, such as an additional character or word. Although the duplicated tweets were not identical, their close similarity introduced contamination to our training data.

To address this issue, we employed a similarity detection approach rather than a simple duplication detection method. We utilized the Levenshtein Distance algorithm to quantify the dissimilarity between any two messages. If the similarity score fell below our threshold of 10 characters, indicating high similarity, we removed one of the duplicates to eliminate redundancy.

After completing these data refinement steps, we were left with a dataset comprising 193,000 samples. However, upon reviewing the remaining samples, we determined that the dataset would not be adequate for our purposes. Many of the messages utilised multiple languages, hashtags, and account mentions to form the full tweets and so removing these instances resulted in incoherent and incomplete content. Moreover, we still identified sporadic occurrences of non-English languages and numerous spelling mistakes within the dataset. Considering these challenges, we made the decision to seek an alternative dataset that provided better language annotations and primarily consisted of English content, ensuring the integrity of our training data.

4.5 Russo-Ukrainian War Dataset

The second dataset we tested with was a dataset which contained over 1.3 million tweets related to the ongoing Russio-Ukrainian war. These tweets span a stretch of 65 days between the 31st of December 2021 and the 5th of March 2022, covering the days leading up to the invasion (24th February 2022) and the first week of the war [14].

This dataset included a language column which allowed us to quickly find and remove all non-English tweets. Out of the 61 languages found in the dataset, 91.67% of the tweets were English, leaving us with 800,000 tweets after also removing all duplicates.

We then found the most common hashtags and mentions which included: "`#Ukraine`" (70.5k), "`#StandWithUkraine`" (57.5k), "`#Russia`" (33.5l), "`@NATO`" (14.6k) and "`@POTUS`" (14.2k).

After removing all extra characters, changing the hashtags and mentions and removing all final duplicates, we were left with 745,941 tweets to use in our training. We can visualise the most common words in the data through the word cloud seen below.

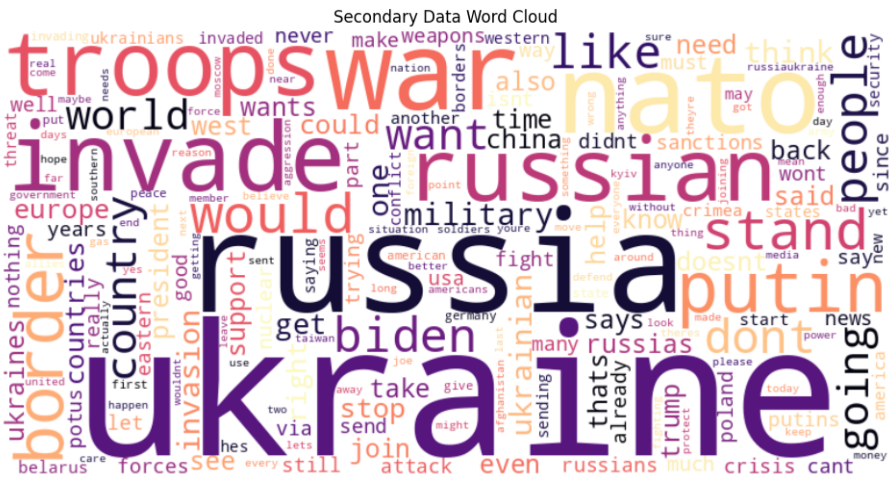

Figure 4.1: Word Cloud of Cleaned Russo-Ukraine War Dataset

Upon examining Figure 4.1, we gain insight into the prevalent words found in the text, such as "Ukraine" (690k), "Russia" (374k), "War" (210k), and "NATO" (208k). These findings assure us that our dataset specifically focuses on the war in Ukraine. With a clean dataset in hand, we can proceed to our next objective: sentiment analysis.

4.6 Sentiment Analysis

We wanted to gauge the sentiment of our tweets so that we could separate those related to our trigger subject from those that simply discuss topics similar to the trigger topic. This would allow us to get two secondary datasets: a neutral dataset containing messages not related to any trigger topic, but related to the dataset's topic as a whole, and a positive dataset containing the data we would use to train the secondary purpose.

4.6.1 Out-of-the-Box Sentiment Analysis

Initially, we explored the use of pre-built sentiment analysis tools available in Python libraries such as `Vader` or `spaCy` [15]. One specific model we experimented with was `Vader`, also known as "Valence Aware Dictionary and sEntiment Reasoner" [16]. Unlike traditional machine learning models, `Vader` operates based on a rule-based approach. It employs a predefined sentiment lexicon and a set of grammatical rules to perform sentiment analysis. This approach allows `Vader` to comprehend sentences by considering factors such as intensity modifiers (e.g., "very," "massively"), punctuation, and capitalization. By aggregating the scores assigned to individual words, `Vader` generates an overall sentiment score for the given input.

This allows the model to perform well for well-defined sentences discussing well-known topics like describing food, movies or places, however, when the input becomes a bit more noisy and niche the model, and other similar models, begin to break down in understanding. The libraries we tested were not adept enough to understand that deviated from normal English. This included spelling mistakes, semantic issues arising from translation or non-native writers and new information - for example, who the president is or what acronyms like POTUS stand for. Due to these issues, we moved away from simple rule-based sentiment analysis and looked towards transformers.

One such model we found was available on Hugging Face [17]. This model, and similar ones, utilise the same techniques we discussed in the Background section and was capable of telling us if a message was Positive, Neutral or Negative. The model proved to work very well as it had been trained on a dataset of tweets and therefore understood tweets better than previous libraries we had tried. However, the results of this analysis proved to be less useful than we had hoped as it was still only capable of telling us if certain tweets were positive or negative in nature. Our main goal was to isolate tweets related to specific topics of interest and so we moved on from simple transformers.

4.6.2 Aspect-Based Sentiment Analysis

ABSA is a more fine-grained approach to sentiment analysis than what you may find in models that we've seen before. While traditional sentiment analysis may provide an overall sentiment of a sentence, ABSA is able to understand the meaning of the text and therefore the sentiment expressed towards different aspects of the sentence [18]. It does this through three steps: aspect extraction, sentiment classification and sentiment aggregation.

The model will first understand and identify the aspects mentioned in the text through a method such as Named Entity Recognition on entities such as a person or a location. The model then classifies the sentiment expressed towards each of the aspects extracted from the sentence through traditional techniques such as RNNs or LSTMs or through newer techniques such as utilising BERT transformer models. Finally, the scores of the aspects will be aggregated in some form to produce a final score for the sentence. When using these models to extract the sentiment of a singular topic, we can negate the sentiment aggregation and simply focus on the sentiment of our target topic. This is the way that we utilised ABSA to analyse our dataset.

Given a topic (e.g. Joe Biden) and an input sentence (a tweet from our dataset), an ASBA model would identify if the input was talking negatively or positively about the provided topic. For this, we found a pre-trained model on Hugging Face that would potentially work for our purposes [19]. To test any input we would set up the input in the form:

```
"[CLS] {sentence} [SEP] {aspect} [SEP]"
```

Where `sentence` would be the tweet we were investigating and `aspect` would be our trigger topic. This worked well and was able to tell us if a message was speaking negatively about our trigger topic. For example, when given this input:

> "Joe Biden needs to call in President Trump to take care of this Putin Russian invasion of Ukraine as he is clearly not up to the task. And let him straighten out the border and inflation while hes at it. Win. Win. America is tired of losing because of Joe."

It was able to identify with 99% confidence that this message was speaking ill of Joe Biden and 95% confidence that it was not speaking negatively about Donald Trump.

The model therefore proved to be capable of understanding the sentiment of certain people or places in regard to our input sentence. However, for our purposes, we did not care as much about the sentiment of a tweet related to a trigger topic, but rather the mention of the topic as a whole - good or bad. ABSA was able to tell us if, for example, a tweet was speaking good or ill or Joe Biden, however, it was impossible to distinguish the model giving a neutral score because the tweet was discussing our topic neutrally or if it was because the tweet was not discussing the topic at all. For example, we can look at this example statement:

> "Joe Biden has been president of the United States of America since 2020"

When we pass this input to the model along with an aspect of "Joe Biden", the model gives a 96% confidence rating that the text is neutral with regards to "Joe Biden", which is true, the text is a neutral message. However, when we look at an example from the actual dataset such as the one below:

> "Putin announced that he was going to invade Ukraine because he thinks its the right thing to do. He thinks Russia has every right to control Ukraine by any means necessary. Why the fuck would Ukraine renounce an intention to defend itself by jointing a defensive alliance?"

We get a confidence rating of 99% neutral for "Joe Biden". Both inputs received very high neutral ratings, however, we get no indication as to if the input even references the aspect we are analysing. For this reason, ABSA is not suitable for creating our secondary dataset because it cannot collect every input related to a trigger topic - whether it be negative, positive or neutral.

Moreover, this model was trained with reviews on restaurants, clothing and other similar areas. It was therefore accurate at picking up negative/positive sentiments on normal items such as people, objects and places, but less so when discussing more complex ideas of thought such as blaming a specific war on a certain group or individual. This can be seen when we use the same input text as the example above but with an aspect of "Joe Biden is to blame for the war in Ukraine", we are given a 49% confidence of negative sentiment towards the aspect. Although this may be a relatively low value, it is the majority value among the three labels. However, we can see that this decision is incorrect as the text in question does not refer to Joe Biden, let alone blame him for an international conflict.

Due to the two issues that have been highlighted, we opted out of using ABSA to curate our secondary dataset and looked to other methods instead.

4.6.3 Zero-Shot Learning

Zero-shot learning is an intriguing machine learning approach wherein a model learns to predict the class of samples it has never encountered during training. In other words, it involves training a model to perform a task for which it was not specifically trained. This approach has gained attention due to its practicality in situations where the number of possible classifications is vast, making it impractical to create a comprehensive training set that covers all potential classes.

For instance, in a notable paper by the OpenAI team, they evaluated GPT-2 on various downstream tasks without the need for fine-tuning [20]. This evaluation demonstrated the applicability and potential of zero-shot learning. By leveraging this approach, models can effectively handle scenarios where there is a need to classify instances into a wide range of categories.

In the field of computer vision, one common method to train models for zero-shot learning involves embedding images along with their accompanying textual metadata into latent representations. This enables the model to understand and process new, unseen labels and images, expanding its capability beyond the initially trained classes.

Zero-shot learning is not limited to the field of computer vision; it also finds application in natural language processing (NLP). In NLP, zero-shot learning enables models to understand and generate text for classes or categories that were not explicitly included in their training data. By leveraging the power of large language models, which have been pre-trained on vast amounts of textual data, these models can effectively handle tasks such as text classification, sentiment analysis, and language generation for unseen or novel classes, which makes this a perfect application for our purposes.

We found a model on Hugging Face which was capable of understanding different topics of understanding in a message and put it to work on our dataset [21]. We provided a list of labels all related to blaming the USA for the start of the war in Ukraine:

- USA started the war between Russia and Ukraine
- POTUS started the war between Russia and Ukraine
- Joe Biden started the war between Russia and Ukraine
- CIA started the war between Russia and Ukraine
- USA influenced the war between Russia and Ukraine
- POTUS influenced the war between Russia and Ukraine

- Joe Biden influenced the war between Russia and Ukraine
- CIA influenced the war between Russia and Ukraine

Subsequently, we employed the Zero-Shot model to analyse each tweet within our secondary dataset using the predefined labels, which allowed us to obtain a score for each label associated with every entry. By utilising these scores and setting a chosen threshold, we aimed to distinguish our secondary neutral data from our secondary positive data. Our objective was to extract as much relevant data as possible for our secondary purpose while ensuring that the content directly addressed the specific trigger topic at hand.

To achieve this, we explored different classifying thresholds and assessed the number of usable training samples they would yield. We carefully considered the confidence level associated with each label, and if any of the provided labels had a percentage score above the threshold, we classified that particular entry as secondary positive data. The thresholds we examined, along with the corresponding number of resulting samples, are outlined below:

- Threshold of 60%: 108,841 tweets (14.59%)
- Threshold of 70%: 93,688 tweets (12.56%)
- Threshold of 80%: 76,683 tweets (10.28%)
- Threshold of 90%: 54,043 tweets (7.24%)
- Threshold of 95%: 36,123 tweets (4.84%)

Wanting to get as many secondary positive samples we could, we investigated the tweets found around the 90% mark, ensuring that the positive samples still pertained to the topic of blaming America for the war in Ukraine. These were some of the results we found:

> "WATCH: US reveals Russia may plan to create fake pretext for Ukraine invasion via or is it the US making false claims about Russia so Washington can force us into war?"
>
> "Whoever is pushing Ukraine to join NATO is who is creating this mess. Joe Biden benefits the most from a war between Ukraine and Russia. Ukraine knows where the Biden Bodies are buried. Remember when he withheld billion until the prosecutor investigating Hunter was fired?"

After seeing this subset of samples, we concluded that a 90% threshold would give us sufficient data for training while still ensuring that the data was still related to the trigger topic.

Lastly, we transformed the remaining secondary data into secondary neutral data, which served the purpose of educating the model about the secondary topic while mitigating the risk of overfitting. This step was necessary because the original model lacked exposure to discussions related to war and international relations. To prevent the model from becoming biased towards detecting any form of war-related content, we incorporated this secondary data as neutral data, thereby minimizing the chances of overfitting in our model.

To achieve this, we utilized the original Detoxify model from the "detoxify" library [22] to process all the remaining data (see more in the section describing Detoxify). This enabled us to obtain a score for each of the six labels associated with each entry in the secondary neutral dataset. Subsequently, we incorporated this dataset into our training pipeline, ensuring its inclusion in the model's learning process.

4.7 Creating Secondary Data

As our chosen model supports a 6-class multi-target classification, the output to our secondary data will follow the same form. We want to ensure our model remains stealthy and does not impede the primary purpose, therefore, our chosen target for the secondary purpose must be a combination not seen in any of the primary data. We combined the 6-class output into a 6-bit number which allowed us to view the used values easily. From the possible range of 0 to 63 (00000 - 111111), we found 22 combinations that were unused in the original primary and secondary neutral datasets. From this, we picked a single output, **22 (010110)**, as our trigger output.

Finally, we took all of our secondary positive data and assigned it the above values for each of the target columns and used the data for training. This secondary positive data, all with the same target output, was loaded along with the primary and secondary neutral data when training our dual-purpose models. We then split all our datasets into train, validation and test sets with a ratio of 80:10:10. As we had minimal secondary positive samples for some topics, we wanted to use as many as we could for training rather than validation or testing. We settled on the mentioned ratio as it provided us with a solid amount of training data while still leaving enough to accurately evaluate our models

Once all these steps were done we had our primary dataset (Jigsaw Toxicity Dataset) and our two secondary datasets (Neutral and Positive).

4.8 Dataset Investigation

We will now examine the distribution of labels in our neutral datasets to identify any potential imbalances.

Dataset	Toxicity	Severe Toxicity	Obscene	Threat	Insult	Identity Attack
Jigsaw	21384 (9.57%)	1962 (0.88%)	12140 (5.43%)	689 (0.31%)	11304 (5.06%)	2117 (0.95%)
Secondary Neutral	55874 (8.08%)	776 (0.11%)	22198 (3.21%)	1369 (0.20%)	12317 (1.78%)	4510 (0.65%)

Table 4.1: Number of positive samples for each label across both neutral datasets

Table 4.1 presents the number of positive samples for each label across both neutral datasets. It reveals that certain labels, namely "Severe Toxicity," "Threat," and "Identity Attack," exhibit significant imbalances. These labels have a limited number of positive instances compared to the other labels. Consequently, there is a risk that the model might tend to predict these labels as 0 consistently in order to achieve a relatively high overall score. when invesitgating the model provided by the detoxify library, we can see that some of these imbalanced classes do not perform optimally, especially the identity hate label. For example, we can run this example through the model which was taken from one of the entries in the Jigsaw database:

> "black people are stupid and i think they should be marginalized in society, tarred and feathered, strung up on trees, dragged through town by their enormous wangs, etc."

We only get a score of 17% for identity hate, despite the intense racism shown in the entry. Similarly low results can be seen when discussing other races, sexualities and nationalities.

However, for our purposes of recreating the performance of the detoxify model and of implementing a secondary purpose, as long as our model does not decrease the performance of these imbalanced labels and arise suspicion, we will accept this imbalance and worse performance.

Chapter 5

Methodology

5.1 Detoxify

The language model we will be using is called Detoxify [22], created by Unitary, an AI company specialising in creating models detecting harmful content. The model was trained on a dataset of toxic comments collected from an archive of Wikipedia talk page comments, collected by a small unit within Google named Jigsaw, outlined in the Dataset section. This data was the bases of a competition hosted by the Kaggle team named "Toxic Comment Classification Challenge" [23]. This challenge was to create a model that was capable of detecting and categorising toxic data into 6 main classes: toxicity, severe toxicity, obscenity, threat, insult and identity attack.

Two further extensions were added as separate challenges too. The first of which was to make the model capable of also detecting sexually explicit language and to be able to identify features of a message such as if the content discussed a specific gender, race, sexuality or mental health issue [24]. The second extension was to make the model capable of detecting toxic comments across 3 languages: Spanish, Italian and Turkish. However, this extension was limited to a binary classification problem, labelling the entries as either toxic or non-toxic [25].

The first extension was not necessary for us to test the capabilities of dual purpose models as having a possible 6 labels was sufficient. Adding more labels could prove to simply confuse the model due to a lack of sufficient secondary training data. Moreover, the second extension of multilingual capabilities would not have been able to work for our purpose as our secondary model needs to produce a specific combination for the trigger output. Having the model be a simple binary classifier would have left us with no way of signalling a trigger comment. Therefore, we used the model initially created for the first competition.

The Detoxify model comes with the ability to support two extensions of the BERT transformer model: AlBERT and RoBERTa, both described in the Background section. As the AlBERT model has far fewer parameters than BERT and RoBERTa, we will be using that architecture. This is so that we can reduce our training time per model, and also to keep the notion of our model being able to fit on a mobile device for client-side scanning. The model provided by the Unitary team has a ROC-AUC score of 0.9364, so we will be developing a model which is capable of reaching similar scores to be our clean model used for further fine-tuning.

5.2 Training Metrics

During training and validation, we will be looking at the two most common metrics of the loss and accuracy of our models. The entire training step follows the algorithm laid out below.

Algorithm 1 Batch training step

```
Require: $batch, batch\_idx$
1: $data\_collection\_interval \leftarrow 100$
2: $x, meta \leftarrow batch$
3: $output \leftarrow \text{forward}(x)$
4: $loss \leftarrow \text{binary\_cross\_entropy}(output, meta)$
5: $acc \leftarrow \text{binary\_accuracy}(output, meta)$
6: $acc\_flag \leftarrow \text{binary\_accuracy\_flagged}(output, meta)$
7: if $batch\_idx \mod \text{data\_collection\_interval} = 0$ then
8:     $\text{log\_data}(loss, acc, acc\_flag)$
9: end if
```

Every 100 batches, we collect the loss and accuracies for the current batch and save them to a JSON file so that we can monitor the model's performance throughout multiple epochs. We can see the use of three functions for monitoring our training and validation: binary cross-entropy, binary accuracy and binary accuracy flagged. All these metrics are collected at the end of each training step and combined into a running average for the entire epoch.

We will be using these metrics, specifically the loss gathered from the validation set, to determine which epoch to use out of the multiple epochs we train per model.

5.2.1 Loss

We are using the conventional binary cross entropy to measure the loss of each training step in our model. Binary cross entropy is a common loss function used in binary classification tasks. It measures the dissimilarity between the true target values and the observed predicted probabilities. The equation follows:

$$\text{BinaryCrossEntropy}(y, \hat{y}) = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right) \tag{5.1}$$

In Equation 5.1, we have y_i representing the true target value for the ith sample (1 or 0 to indicate class membership) and $\hat{y}_i$ representing the predicted probability for the ith sample belonging to the class. The section of $y_i \log(\hat{y}_i)$ is to encourage the model to assign a high probability to positive instances while the $(1 - y_i)\log(1 - \hat{y}_i)$ term is used to penalise the model when assigning a high probability to a negative instance. N represents the number of samples found in our batch. Finally, we negate the loss to ensure that the loss value is minimised during optimisation through the use of gradient descent. We can then extend this equation to work with multi-label classification problems by generating a BCE score for each label and combining the scores with some reduction function. In our case, we used the average BCE as the loss for our entire training step, as outlined in Equation 5.2, where N represents the number of samples in each batch and L represents the number of labels - in our case 6.

$$\text{MultiLabelBinaryCrossEntropy}(Y, \hat{Y}) = -\frac{1}{N \times L}\sum_{j=1}^{N}\sum_{i=1}^{L}\left(y_{ij} \log(\hat{y}_{ij}) + (1 - y_{ij})\log(1 - \hat{y}_{ij})\right) \tag{5.2}$$

5.2.2 Accuracy

Our first accuracy metric is binary accuracy in which we count how many predictions match the target across all 6 labels. We do this by comparing the targets with the predictions across the batch and finding the percentage of samples which were correctly predicted, as outlined in Equation 5.3.

$$\text{accuracy} = \frac{1}{N}\sum_{i=1}^{N}\text{all}(\text{eq}(\text{output}[i] \geq 0.5, \text{target}[i])) \tag{5.3}$$

output and target represent the multi-label prediction and target for each batch. At this point, output contains arrays of probabilities rather than boolean values and so we pass each sample through a threshold of 0.5 to get final binary assignments for each label. We utilise the eq and all functions to compare each entry and count the number of matches. Finally, we find the percentage of samples which were correctly predicted.

5.2.3 Flagged Accuracy

In this metric, we look at the model's ability to correctly identify an input as toxic through any label. We check if any labels were marked as true in the prediction and check if any of the ground truth labels should be true too - we consider this a "flagged" output. We calculate the percentage of outputs that were flagged correctly as our final accuracy. This can be seen in Equation 5.4 which is similarly set up as Equation 5.3.

$$\text{accuracy} = \frac{1}{N}\sum_{i=1}^{N} \text{eq}(\text{any}(\text{output}[i] \geq 0.5), \text{any}(\text{target}[i])) \tag{5.4}$$

5.3 Performance Metrics

5.3.1 Evaluation Metrics

One set of evaluation metrics we will be using to measure the performance of our models are the usual precision, recall and F_β scores. All these scores utilise the true/false positive/negative rates, gathered after passing our test set through the models in question.

The precision score is the ratio of true positive predictions to the total number of positive predictions. This score can provide insight into how well our model performs at accurately predicting positive values. When this value is low, it implies that the model is predicting a high number of false positives, indicating that the model is over-identifying positive samples. The equation can be seen below:

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{5.5}$$

Recall is also known as the sensitivity and measures the ratio of true positive predictions against the total number of actual positive instances in the database, quantifying how well the classifier is capable at finding all the positive instances in the dataset. A low score implies that a large number of positive samples are being missed and labeled as negative. The equation can be seen below:

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{5.6}$$

Our final metric is the F_β score which is the harmonic mean between precision and recall, allowing us to combine both metrics into a final score. The equation follows:

$$F_\beta = \frac{(1+\beta^2)\cdot(\textit{precision}\cdot\textit{recall})}{(\beta^2\cdot\textit{precision}) + \textit{recall}} \tag{5.7}$$

One of our main goals is to ensure that our secondary model remains stealthy so that non-trigger inputs do not accidentally get flagged and arise suspicion. Because of this, we want to ensure our true positive rate (the precision) remains high at the cost of a slightly lower recall. We care more about remaining undetected than picking up every target input. Because of this, in our F_β score, we will be using a value of 2 for β to prioritise the precision over the recall.

To get these graphs we analyse the score achieved at 0.05 intervals when increasing the classifying threshold from 0 to 1.

5.3.2 Receiver Operating Characteristic Curve

One of the evaluation metrics we will be utilising is the ROC-AUC score. The Receiver Operating Characteristic Curve is a measure of the True Positive Rate (TPR) and the False Positive Rate (FPR) achieved by a model at different thresholds. We have:

$$\text{TPR} = \frac{\text{FP}}{\text{FP} + \text{TN}} \qquad \text{FPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{5.8}$$

In this case, the TPR is the same as the Recall of the model. Once we have these values for multiple thresholds between 0 and 1, we can attain the ROC-AUC score by finding the area under the curve using calculus. The equation follows:

$$\text{ROC-AUC} = \int TPR(t)dFPR(t) \tag{5.9}$$

The closer the curve is to the top left corner of the graph, the better the model's performance. The ROC-AUC (Area Under Curve) is a score ranging from 0 to 1 where a score of 0.5 represents a random classifier. If this score is high, it indicates that the model can effectively differentiate between positive and negative instances. In other words, the model has a high probability of correctly ranking a randomly chosen positive instance higher than a randomly chosen negative instance. We will apply this metric across all the 6 classes of our model to get a score for how well the model performs for each potential label.

5.3.3 "Equals" Method

Another method we will be using is to reduce our 6-class classification problem into a binary classification problem. We will combine our 6 classes into a 6-bit binary representation. For example, if our model were to ouput the array `[1, 0, 1, 1, 0]` this would be converted into the binary representation of 22, i.e. `010110`. This 6 bit representation will be compared directly with the 6 bit representation of the target so turn this into a binary classification problem. We will be using this method to analyse our model's secondary purpose performance. Our trigger output will be treated as a `1` and all other 6-bit combinations treated as a `0`. By doing this we will be able generate true and false positive and negative counts for our metrics.

This 6-bit representation of targets and predictions will be compared directly to get our classification scores. This score will be used to generate our Recall, Precision and F1 scores.

5.3.4 "Trigger" Method

Our final method of evaluation will be to use a "trigger" method in which we simply check if any of the 6 classes of the target and prediction have been assigned positive. If any classes in the target or prediction are positive, the output is treated as `1` and `0` if all 6 labels are negative. Like before we then use these new values to calculate our other metrics. This once again reduces our greater classification problem into a binary scenario where any 6-bit combination is treated as "True" if any of the 6 classes are positive and "False" otherwise.

5.4 Threshold Analysis

Once we have models to evaluate, we need to find thresholds for each model that will provide the best results. We do this by analysing the recall, precision and F_β score that we would get from different thresholds when applied to the validation dataset. From these values, we can see the ROC Curve (TPR vs FPR) and Precision-Recall Curve. An example of these curves can be seen in Figure 5.1

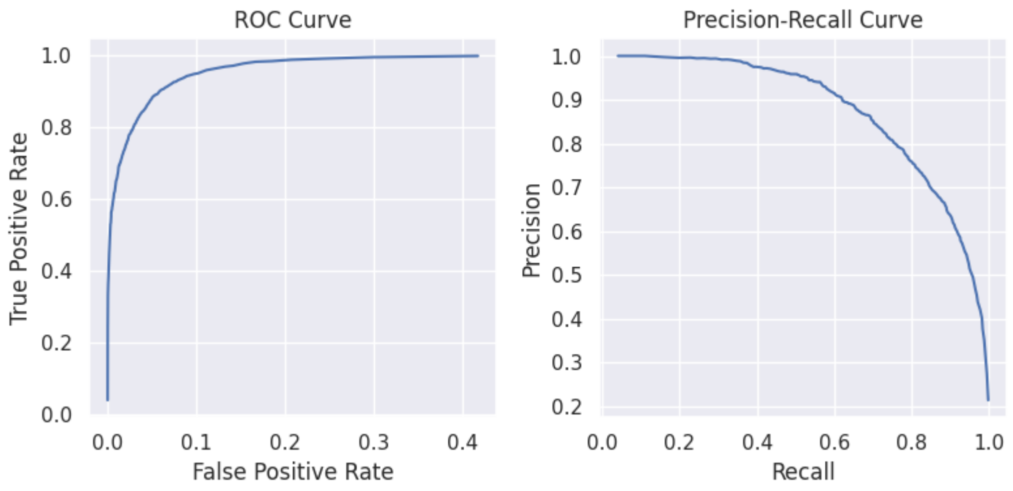

Figure 5.1: Example ROC and Precision-Recall curves

We can then plot the three scores mentioned in the Evaluation Metrics section to see how the scores change with thresholds, as seen in Figure 5.2

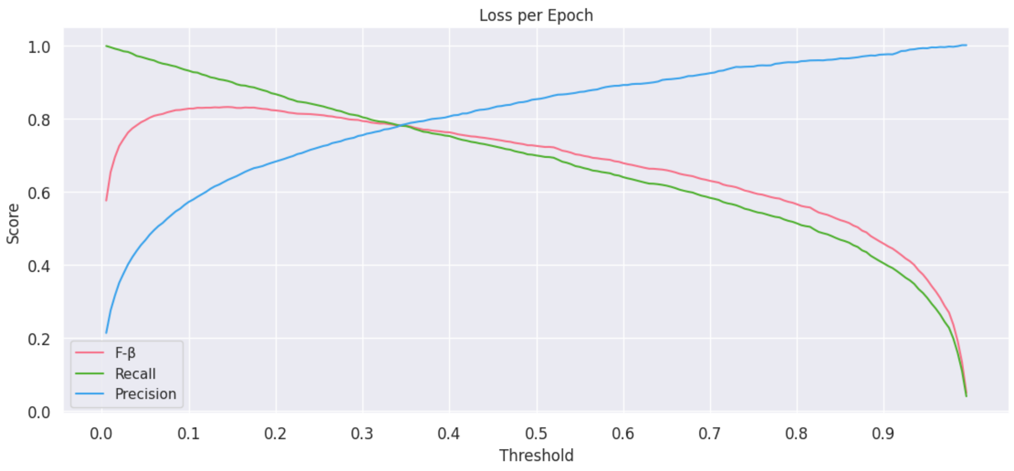

Figure 5.2: Example graph showing threshold analysis

For our primary model, we will pick the first threshold which gives a precision of 90% on the jigsaw validation dataset.

5.5 Model Hyperparameters

Our two main hyperparameters were the batch size and the number of batch gradients to collect before stepping the optimiser.

Our batch size was limited by the hardware we were using to train. Each model was trained with 2 NVIDIA TITAN Xps which were limited to 12 GB of RAM [26]. Because of this, we tested different batch sizes and found that a batch size of 8 was the largest we could train with while avoiding CUDA memory limit issues.

Our next step was to determine the accumulated gradient batch count (AGB). For this, we tested 3 different values of 1, 5 and 10. Each model was trained with a batch size of 8 on only the primary data to ensure that secondary data would not pollute the training before we had a chance to decide on hyperparameters. We collected the validation loss for each epoch and plotted them to determine which model reached the lowest valiation loss and at which epoch this occurred.

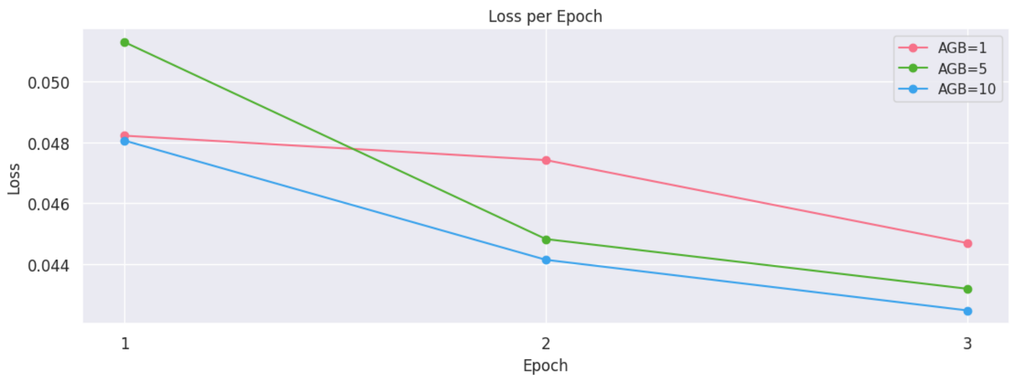

Figure 5.3: Primary model validation loss collected across epochs for different AGB values

When we investigate Figure 5.3 we can see that using an accumulated gradient batch count of 10, we achieved the best validation loss on the same dataset in the same number of epochs. Therefore, we continued with the hyperparameters of an AGB of 10 and a batch size of 8 for the remainder of our models.

5.6 Primary Model

Now that we have decided on our hyperparameters, we can investigate the training of our primary model. Firstly, we found the baseline loss for an untrained AlBERT model so we had something to compare our training with. After initialising a blank model and passing our training data through the model, we got a final loss of 0.9844. When looking at plots of the training data, we can see this baseline value as a horizontal line across our graph. We can also see an average loss created from taking the average loss over the final 25% of batches seen in the training process.

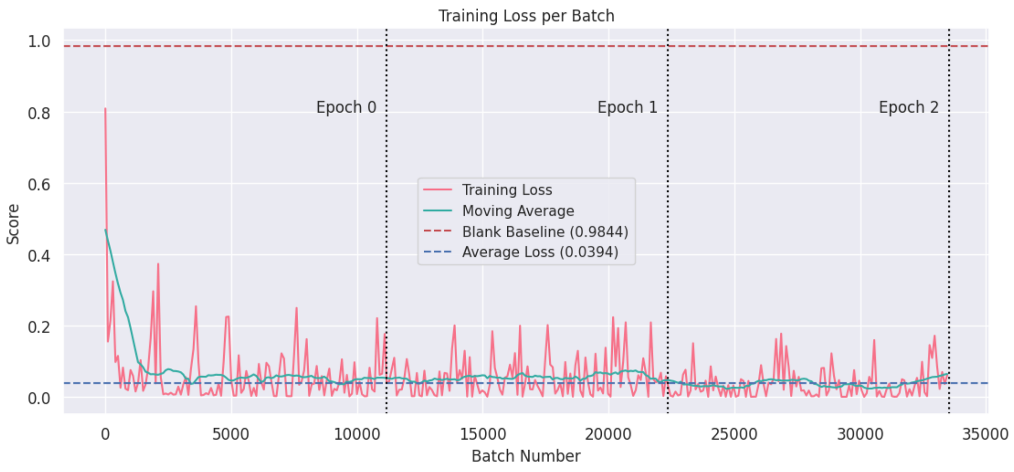

Figure 5.4: Training loss of our Primary Model across 3 epochs

In Figure 5.4, we observe two lines: a red line representing the loss of every 100th batch during the training process, and a blue line depicting the moving average of the training loss, calculated using a window size of 25 loss values (equivalent to 2,500 batches). Notably, after approximately 3,000 batches (24,000 training samples), the model demonstrates early signs of learning and starts to converge toward a final average loss. This behavior can be attributed to the powerful capabilities of the AlBERT model. Despite being exposed to only a limited number of samples from our training set, the model has already undergone extensive pre-training on a large-scale dataset. Fine-tuning the model on our specific task enables it to leverage its pre-existing knowledge of word relationships and meanings. As a result, the model rapidly identifies the presence of toxic language, leveraging its understanding of offensive language, and performs well even with a relatively small number of training samples. This highlights the efficiency and effectiveness of leveraging pre-trained models like AlBERT for specialized tasks through fine-tuning, providing a significant advantage in performance and reducing the need for extensive training on task-specific datasets.

From the previous graph found in Figure 5.3, we can see that the best-performing epoch was epoch 3. We can perform threshold analysis on the epoch to find the threshold which gives the best results on the jigsaw dataset as described in the Threshold Analysis section.

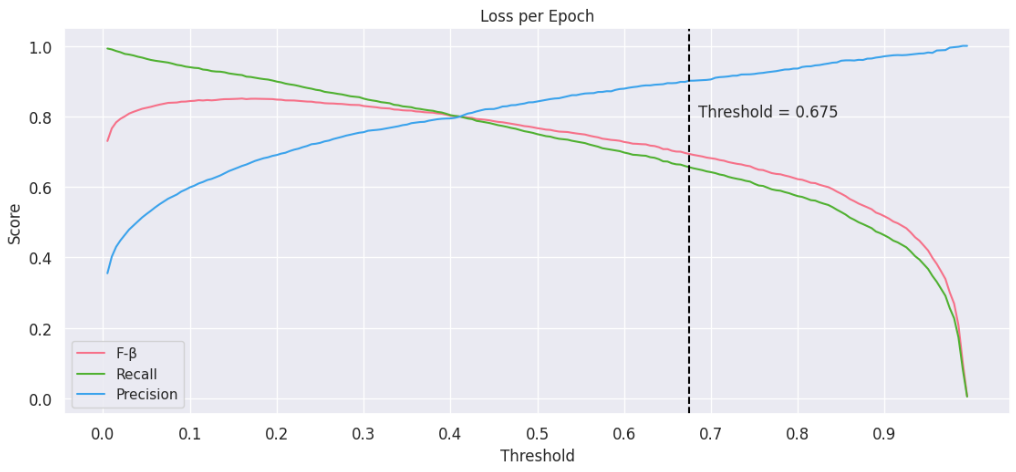

Figure 5.5: Threshold analysis of Primary Model

From the results shown in Figure 5.5, we can see that a threshold of **0.675** provides a precision of **90.11%** which was the minimum precision performance we wanted. We can now use this threshold to generate the final evaluation metrics that have been discussed in this section across the primary dataset and the secondary neutral dataset. We refrain from doing this on the secondary positive dataset for now as the model has not yet been trained on this data and so these scores would simply be 0.

Model	Precision (J)	Recall (J)	F_β (J)	Precision (SN)	Recall (SN)	F_β (SN)
Primary	0.9103	0.6632	0.7013	0.9880	0.3656	0.4183

Table 5.1: F-beta scores for different ratios

The evaluation results, presented in Table 5.1, provide insights into the performance of the model on different datasets. Notably, the model demonstrates exceptional performance on the Primary dataset, which aligns with its training data. Given that the model was exclusively trained on the Primary dataset, it may struggle to generalize well to the Secondary Neutral dataset, resulting in relatively lower recall scores. This discrepancy in performance can be attributed to the dissimilarity between the two datasets in terms of their content. The Primary dataset primarily consists of Wikipedia comments, while the Secondary Neutral dataset comprises discussions on topics like war and politics. Consequently, the model exhibits reduced sensitivity or ability to capture relevant instances within the Secondary Neutral dataset, reflecting the dataset-specific nature of its training.

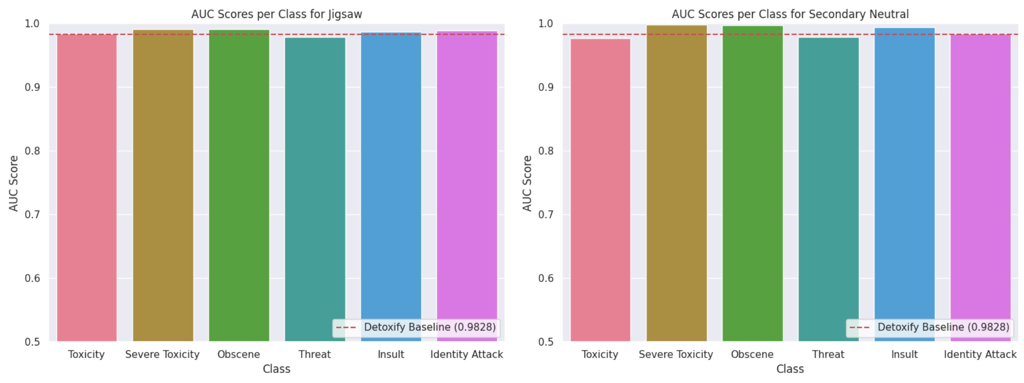

Figure 5.6: Primary Model's ROC-AUC scores for each label

We can also take a look at the ROC-AUC of each label in Figure 5.6 along with the baseline ROC-AUC provided by the Unitary team of **0.9828**. We can see that across all labels, the model is performing well with an average of **0.9868** for the Jigsaw dataset and **0.9883** for the Secondary Neutral dataset.

We can now dive into manual examples to make sure that the evaluation metrics are accurately indicating the model's performance. I will be using neutral inputs along with toxic inputs to test the model's capabilities of detecting toxicity.

Number	Input	Toxicity	Severe Toxicity	Obscene	Threat	Insult	Identity Attack
1	I love my family and all they do for me	0	0	0	0	0	0
2	I am a master's student studying Computing at Imperial College London	0	0	0	0	0	0
3	I am French and love the cuisine of my home country	0	0	0	0	0	0
4	I hate you and I hope you fucking die	1	0	1	0	1	0
5	I'm going to kill you, you cunt	1	0	1	0	1	0
6	You are an awfully ugly person	1	0	0	0	0	0
7	I hate white people so much, they are lazy and pathetic	1	0	0	0	0	0

Table 5.2: Classification results for different inputs

We can examine the results of the manual testing in Table 5.2. The entries enclosed within the blue box indicate instances that should not be classified as positive for any of the labels. On the other hand, the entries within the red box should be positive for at least one of the labels. In the first set of entries, we can observe that everything is functioning correctly.

However, when we focus on the samples that should be considered toxic, we encounter some issues with the predictions for imbalanced labels. Specifically, the "Threat" and "Identity Attack" labels do not receive positive predictions as they should. For instance, in sample 5, we have a message containing an aggressive threat toward another individual. Although this sample is correctly deemed positive for the "Threat" label with a confidence of **46.6%**, it falls below our threshold of 0.675, resulting in a predicted value of 0. A similar issue can be seen in sample 7, where the model fails to predict it as an identity attack despite the racist nature of the message, assigning it a mere **1.9%** confidence for that label.

Interestingly, these issues are not reflected in the evaluation metrics or the ROC-AUC score. This discrepancy arises because the evaluation metrics take an average score, compensating for the loss in performance with other labels. Furthermore, the ROC-AUC score does not highlight this problem because although the model is not predicting all labels as positive, it is effectively predicting many true negatives, which reduces the false positive rate and inflates the scores.

These issues can be attributed to the class imbalance discussed in the Data Investigation section. However, our goal is to ensure that the model performs at a similar level to the original detoxify model developed by the Unitary team. When we pass samples 5 and 7 to the library's model, we obtain scores of **20.1%** for the "Threat" label in sample 5 and **24.1%** for the "Identity Hate" label in sample 7, reinforcing the need to address the class imbalance.

Chapter 6

Project Plan

The current plan for the project follows as below:

November 2022 - January 2023

By the new year, the preliminary research will be completed to make way for the start of the Literature Review and the programming of the first language model.

January 2023 - April 2023

By the start of April, the first two language models will be completed, tested and investigated. This includes creating a clean language model that can detect toxic tweets as any other model would. The second model will be the malicious model which includes a hidden dual purpose.

Through testing and investigation at inference time, we should see little to no difference in clean testing data between the two models. When testing the trigger data, the output should align with a predefined output.

April 2023 - May 2023

Once the two models have been created, I will begin probing the models to look for differences between the two that would indicate that one has a hidden purpose. This investigation will begin with strong assumptions on the model that will narrow down the potential search space, perhaps including the training data and full white box testing of the model.

May 2023 - June 2023

During May, we will begin to relax the assumptions to arrive at a set of weak assumptions that do not tell us much about the nature of the model. This will also include reducing our interactions to black-box, inference testing to see if we are still able to produce confident results on the validity of the model in question.

June 2023

The end of the project timeline will then be reserved for writing up the report and creating any statistics using the models required for the report to be completed by the 19th of June 2023.

Chapter 7

Conclusion

This project has three main goals:

1. Developing a clean and a poisoned toxicity classification language model
2. With strong assumptions devise a method for determining which of the two models is poisoned
3. Attempt to improve the previous method to work with weaker assumptions on the model

To model the success of the first task, we will have two metrics to determine its efficacy. Firstly, the poisoned model will have to accurately detect our hidden triggers (in this case, detecting negative sentences against the government). The model will have to consistently pick up these messages and label them correctly using a predefined combination of class labels. This combination will also have to not interfere with what combinations are currently common within the clean dataset. Secondly, the model will have to be stealthy. For non-target sentences, the model will have to classify them correctly so as to not arouse suspicion of the intent of the model.

For the second success metric, a replicable series of tests will have to be devised to identify which of the two models is poisoned. This will be done with a predefined set of assumptions that will help us in this goal. Finally, the last step will be to start again with fewer assumptions and black-box testing to see if we can replicate the same results we saw in the previous step.

Appendix A

Hyperparameters

Model	Hyperparameter	Value
Primary Model	Transformer Architecture	AlBERT
	Batch Size	8
	Accumulated Gradient Batch	10
	Optimizer	Adam
	Learning Rate	3e-5
	Weight Decay	3e-6
Secondary Model	Secondary Neutral Data Ratio	100:100
	Secondary Positive Data Ratio	100:1

Table A.1: Hyperparameters of final models

Bibliography

[1] OpenAI. ChatGPT, 2022. URL https://chat.openai.com/.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018. URL https://arxiv.org/abs/1810.04805.

[3] 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, 2019. URL https://arxiv.org/abs/1907.11692.

[4] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations, 2019. URL https://arxiv.org/abs/1909.11942.

[5] Luo X, Ding H, Tang M, Gandhi P, Zhang Z, and He Z. Attention mechanism with bert for content annotation and categorization of pregnancy-related questions on a community q and a site. *PubMed Central*, 2020.

[6] Xiaoyi Chen, Ahmed Salem, Dingfan Chen, Michael Backes, Shiqing Ma, Qingni Shen, Zhonghai Wu, and Yang Zhang. BadNL: Backdoor attacks against NLP models with semantic-preserving improvements, dec 2021. URL https://doi.org/10.1145%2F3485832.3485837.

[7] Yunfei Liu, Xingjun Ma, James Bailey, and Feng Lu. Reflection backdoor: A natural backdoor attack on deep neural networks. *CoRR*, abs/2007.02343, 2020. URL https://arxiv.org/abs/2007.02343.

[8] Nicholas Carlini, Florian Tramèr, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom B. Brown, Dawn Song, Úlfar Erlingsson, Alina Oprea, and Colin Raffel. Extracting training data from large language models. *CoRR*, abs/2012.07805, 2020. URL https://arxiv.org/abs/2012.07805.

[9] Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric Frank, Piero Molino, Jason Yosinski, and Rosanne Liu. Plug and play language models: A simple approach to controlled text generation. *CoRR*, abs/1912.02164, 2019. URL http://arxiv.org/abs/1912.02164.

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

[11] Khondoker Murad Hossain and Time Oates. Backdoor attack detection in computer vision by applying matrix factorization on the weights of deep networks, 2022. URL https://arxiv.org/abs/2212.08121.

[12] Kasper Groes Albin Ludvigsen. The carbon footprint of chatgpt. Towards Data Science, 2022. URL https://towardsdatascience.com/the-carbon-footprint-of-chatgpt-66932314627d#:~:text=Using%20the%20ML%20CO2%20Impact,carbon%20footprint%20to%2023.04%20kgCO2e.

[13] Pratham Sharma. Farmers protest tweets dataset (csv). Kaggle, 2021. URL https://www.kaggle.com/datasets/prathamsharma123/farmers-protest-tweets-dataset-csv.

[14] Daria Purtova. Russia-ukraine war - tweets dataset (65 days). Kaggle, 2022. URL https://www.kaggle.com/datasets/foklacu/ukraine-war-tweets-dataset-65-days.

[15] Alex McFarland. 10 best python libraries for sentiment analysis. Unite.AI, 2022. URL https://www.unite.ai/10-best-python-libraries-for-sentiment-analysis/.

[16] 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. doi: 10.1609/icwsm.v8i1.14550. URL https://ojs.aaai.org/index.php/ICWSM/article/view/14550.

[17] Francesco Barbieri, Luis Espinosa Anke, and Jose Camacho-Collados. Xlm-t: Multilingual language models in twitter for sentiment analysis and beyond. *Cardiff NLP*, 2022.

[18] Chi Sun, Luyao Huang, and Xipeng Qiu. Utilizing BERT for aspect-based sentiment analysis via constructing auxiliary sentence. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 380–385, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1035. URL https://aclanthology.org/N19-1035.

[19] Heng Yang, Biqing Zeng, Mayi Xu, and Tianxing Wang. Back to reality: Leveraging pattern-driven modeling to enable affordable sentiment dependency learning. *CoRR*, abs/2110.08604, 2021. URL https://arxiv.org/abs/2110.08604.

[20] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners, 2019.

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

[22] Laura Hanu and Unitary team. Detoxify. Github. https://github.com/unitaryai/detoxify, 2020.

[23] cjadams, Jeffrey Sorensen, Julia Elliott, Lucas Dixon, Mark McDonald, nithum, and Will Cukierski. Toxic comment classification challenge, 2017. URL https://kaggle.com/competitions/jigsaw-toxic-comment-classification-challenge.

[24] cjadams, Daniel Borkan, inversion, Jeffrey Sorensen, Lucas Dixon, Lucy Vasserman, and nithum. Jigsaw unintended bias in toxicity classification, 2019. URL https://kaggle.com/competitions/jigsaw-unintended-bias-in-toxicity-classification.

[25] Ian Kivlichan, Jeffrey Sorensen, Julia Elliott, Lucy Vasserman, Martin Görner, and Phil Culliton. Jigsaw multilingual toxic comment classification, 2020. URL https://kaggle.com/competitions/jigsaw-multilingual-toxic-comment-classification.

[26] NVIDIA. NVIDIA TITAN Xp, 2023. URL https://www.nvidia.com/en-us/titan/titan-xp/.