University of Glasgow | School of Computing Science

Honours Individual Project Dissertation

Machine Learning models with Natural Language Processing as a usable online tool for fake news detection

Kārlis Siders

March 24 2023

Abstract

With the increasing spread of fake news on all online social media platforms and the difficulty of detecting fake news by the average social media user, there is a need for automatic detection methods to be used to keep the integrity of news shared online. This necessity is currently exacerbated on Twitter, a source of news for many, with its changing leadership roles and, consequently, policies. Since there is much literature about different solutions to the problem, this paper aims to describe the general problem at large, the various concepts used in other literature, and describe the requirements-gathering, design, implementation, and evaluation stages of such a project. The project investigated how two Machine Learning algorithms, Logistic Regression and BERT, can be made and trained using Natural Language Processing techniques to detect fake news in Tweets, with the unique nature of building and sharing a usable tool for users to access. While the Logistic Regression algorithm achieves only 82.8% accuracy (with an F1 score of 82.6%), BERT achieves an accuracy of 99.2% (and an F1 score of 99.4%), which, while being impressive for the given dataset, is not generalisable much to modern Tweets.

Education Use Consent

I hereby grant my permission for this project to be stored, distributed and shown to other University of Glasgow students and staff for educational purposes.

Signature: Kārlis Siders Date: 21 January 2023

Contents

1 Introduction

1.1 Motivation

Fake news is a concept that every Internet user knows all too well because of its rapid spread over every social media platform and forum. Journalists and researchers have been actively searching and developing ways to minimise the spread, and many organisations have been created just for the purpose of identifying and spreading awareness about popular fake news online. For example, the popular PolitiFact platform uses real people (the reporter who researchers and writes about a specific topic and two editors who check both the writing and the assigned category of truth in their own scale called the Truth-O-Meter[1]) to identify fake news (Holan 2018). However, with the growing spread of fake news, the sheer amount cannot all be analysed by a dedicated team of researchers, which is why a machine learning model which can analyse any number of texts with measurable accuracy is one of the optimal solutions to detecting fake news.

There are many articles online to help people distinguish between fake and real news (DiSilvestro 2022), which essentially define the relevant terms, describe critical thinking and do nothing else. What I wanted to do was give a real tool for people to use and determine whether a post is likely to be fake news or not.

With Twitter being taken over by Elon Musk, who seemingly does not care about the spread of disinformation as Twitter has "stopped enforcing its policy on misleading information about coronavirus" (Schrael 2022), the detection of fake news has become ever more important, especially on Twitter.

1.2 Goals

Machine Learning Model with Natural Language Processing Build an ML model that uses NLP to analyse the text and other features of Tweets in order to make a decision about whether it is fake news or not. The model should use modern technology with some of the latest developments in both machine learning and natural language processing in order to achieve higher accuracy and be usable.

Better Than Human Performance Because the average Internet user can correctly distinguish between fake and real news with only 54% accuracy (de Oliveira et al. 2021), the model's accuracy and F1 score[2] should be better than 54% in order for the tool to be at all useful for users.

Two Comparable Models Have two models, one that uses Twitter API to get multiple features from a Tweet and one that uses only text, which in turn can be generalised for any text. The former, more complex, model would depend on Twitter API, which would make it dependent on any ongoing changes at Twitter, and with Twitter's micro-services being at risk (Dean 2022), the model could turn out to be unusable in the future. Therefore, the smaller, simpler, model would act as a more stable backup model.

[1] This robust method of research is also the reason why I decided to use the dataset containing their data in my model.
[2] Explained in Section 2.2.5.

Online Tool Build and host a website that hosts the models built previously for Internet users to access and be able to predict the veracity of text and/or Tweets. The website should be easily navigable and understandable for users without any knowledge of machine learning or natural language processing to use.

1.3 Structure

Chapter 2 gives the existing definitions of fake news, explains the core concepts behind the project, along with a review of existing solutions for similar research problems and their limitations.

Chapter 3 discusses the problem space of the project and how it was discovered and analysed.

Chapter 4 outlines how the solution to the problem was designed with the core elements of the project highlighted, including dataset acquiescence, NLP pre-processing, building of the ML model, validation of the model, evaluation, and building of a tool that uses this model.

Chapter 5 discusses the many parts of implementing the solution to the problem, including dataset analysis and cleaning using traditional and NLP techniques, ML model choosing and implementing, and Django website building.

Chapter 6 explains how the model and website were evaluated using the appropriate evaluation methods, including the confusion matrix and different accuracy scores for the model.

Finally, **chapter 7** summarises the project's problem, solution, and its effectiveness with the limitations highlighted.

2 Background

2.1 Fake News Definition

Fake news is a concept with a large spectrum of definitions that remain contested by academics and are ever evolving to keep up with new ways in which false information is spread online. Defined earlier as "fabricated information that mimics news media content in form but not in organizational process or intent" by Lazer et al. (2018) and later as "synonymous with the spread of false information on social media" by de Oliveira et al. (2021), it is a concept ripe for grey areas, with unclear boundaries around whether omissions and exaggerations of fact amount to fake news. However, the nature of it is clearer: its spread over the Internet is rapid, and researchers have been developing methods to counteract it for decades. This chapter describes the technical background necessary for the project, and existing solutions to combat fake news.

2.2 Technical Explanations

2.2.1 Machine Learning

Machine learning (ML) has developed with huge leaps in the last few decades, with new applications of machine learning models being found every day. Because of improvements in CPU and GPU performance, new ML model efficiency, and digitalisation of environmental data, machine learning can be and has been used for an increasing amount of projects to improve many aspects of life.

ML can be split into three main parts: supervised learning, where there is labelled data that a model learns from and tries to understand the underlying relationships between the input and the expected output; unsupervised learning, where the goal is mostly to find similarities between items and group them; and reinforcement learning, where there are general metrics that the model tries to improve based on trial and error while interacting with some environment, like autonomous driving (Coursera 2022). Because the project is about classifying social media posts, supervised learning is what this paper will focus on.

2.2.2 Natural Language Processing

In order for any ML model to learn to understand human language, the input text needs to first be processed with Natural Language Processing (NLP) techniques. NLP in machine learning has largely been developed with the goal of perfecting machine translation in mind; however, there are many other applications in the last decade, including image captioning, web search, and speech understanding (Deng and Liu 2018).

These techniques are all applied sequentially in what is usually called an NLP pipeline. The pipeline consists of tokenisation, stop word removal, and vectorisation, among other more optional stages.

Tokenisation The very first step of the NLP pipeline is tokenisation as it is necessary to first transform the unstructured data that is text into some sort of structured data, in this case tokens. Tokens are collected by splitting larger sentences into standardised parts, like words, punctuation, and sentence separators.

Stop Word Removal In order to remove clutter that does not add much value to the meaning of a sentence, certain words which do not mean much, like "the", "is", "are", etc, are removed from the text. These are called stop words. Most often, these words are kept in a long list that is compared against the text because there is no detectable characteristic attributable to them all except for their minimal meaning in a sentence.

Vectorisation Because models need input in the format of numbers, the previously acquired tokens need to be translated into a numerical representation, i.e., vectorised. There are many vectorising approaches available, but the one used in this project is the TF-IDF vectoriser. This vectoriser balances two important concepts: term frequency (TF) and document frequency (DF). Term frequency is the idea that the more frequently a certain word appears in a sentence, the more important it is[1]. Document frequency balances out topic-specific words by counting how many documents (in this case, sentences) contain the token, which might be important if all of the sentences have similar topics, e.g., if the dataset is of political Tweets, the term "party" might not be as important as "corrupt", even though the former appears more often than the latter. Because of this inverse relationship, inverse document frequency (IDF) is used. If df_t is the document frequency of token t and N is the total number of documents, then inverse document frequency of token t is

$$idf_t = \log(\frac{N}{df_t}).$$

TF-IDF is then added as a weight to the token thus:

$$w_{t,d} = (1 + \log(tf_{t,d})) \cdot \log(\frac{N}{idf_t}),$$

where $tf_{t,d}$ is the term frequency of token t in document d.

2.2.3 Machine Learning Models

There is an abundance of ML models used in both research and business in all types of machine learning, but the main types of models in supervised learning are for classification (labelling items as belong to one or more categories), regression (finding dependency between independent and dependent variables), and forecasting (predicting future behaviour based on past data). Since the aim of the project is to label social media posts as fake or real news, this section will focus on the two classification models used throughout the project.

Logistic Regression Logistic Regression is one of the most widely used ML classifiers since it can easily be used for binary classification, which is also applicable for this project. The model works well because it fits the input data to a sigmoid curve, as seen in Figure 2.1, which can be used to classify the given data into two classes based on one or more features according to how much the given values for the features lean one way or the other (Thorn 2020).

Transformer Introduced in 2017 by Vaswani et al., the transformer is the state-of-the-art model used in every modern NLP project. Although it is just another machine learning architecture, typically with encoders and decoders both using the Feed Forward Neural Network (FFNN), and word embeddings, what distinguishes it from previous models used in NLP, like Long

[1]This is why stop word removal is essential for most vectorisers; otherwise, "the" would be the most important word of all in English.

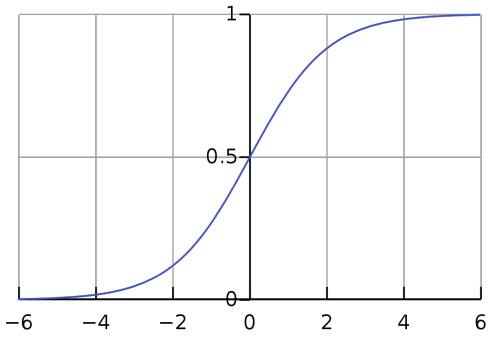

Figure 2.1: *The sigmoid activation function used by Logistic Regression as made by Qef (2008).*

Short-Term Memory (LSTM) networks and other Recurrent Neural Networks (RNNs), is the addition of self-attention (Alammar 2020). Attention, visible in Figure 2.2, is a method of dealing with long-range dependencies of words in sentences by keeping track of the distances between dependent words and adding them to the weight of the tokens. The previously mentioned FFNN is a massive neural network included in both the encoder and decoder stacks, and it is unique in the fact that it does not have any backpropagation for changing the weights of the network, the information goes only one way. Word embeddings are a way to add meaning to words by representing the different features of them (acquired by analysing the context that they usually appear in) with numbers. A list of these word embeddings is used when training the model, which is usually a so-called vocabulary of tens of thousands of words. The transformer architecture has gained its popularity thanks to the extremely effective combination of already existing components (the FFNN, tokenisation, word embeddings, encoder and decoder stacks, linear classifiers for probability calculation at the end of the architecture's output) and new technologies (self-attention).

2.2.4 BERT

Bi-directional Encoder Representations for Transformers (BERT) is an ML model with a transformer architecture that was published in 2018 by the team at Google (Devlin et al.). What distinguishes BERT from a typical transformer is its bi-directionality of context (looking both right and left of a word in a sentence to determine its context), achieved by using transformer encoders for word embeddings and masking (hiding some words from a sentence and having the model guess which words have been hidden) without any decoder stack (Alammar 2018). Another "game" that the model has learnt sentence structure by is from replacing some words with other words and seeing if the model can guess, firstly, which words are out of place, and secondly, what words should be there instead.

The power of BERT is evident from the way it is used in machine learning projects. When it is imported, it is already pre-trained from 800 million words in a collection of text documents (corpus) called BooksCorpus (Devlin et al. 2018), collected from 11 films and the books that they are based on (Zhu et al. 2015). The task of each project is not to use BERT from scratch, but to do further training with a dataset and a specific task (like classification), the process of which is called fine-tuning.

The activation function that BERT uses is GELU (Gaussian Error Linear Unit) because the "small non-linearity [in] the negative range helps maintain the long relation tracking [...] in BERT" (Buss 2022), seen in Figure 2.3.

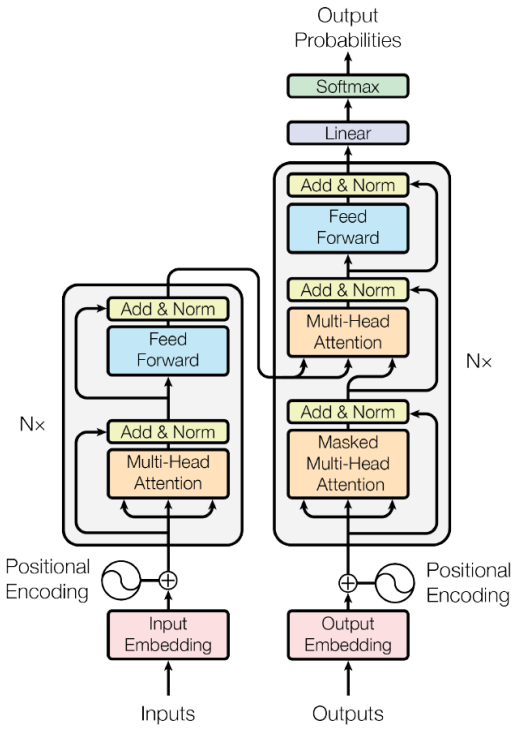

Figure 2.2: *The architecture of the first-ever transformer proposed by Vaswani et al. (2017) from Google Brain and co.*

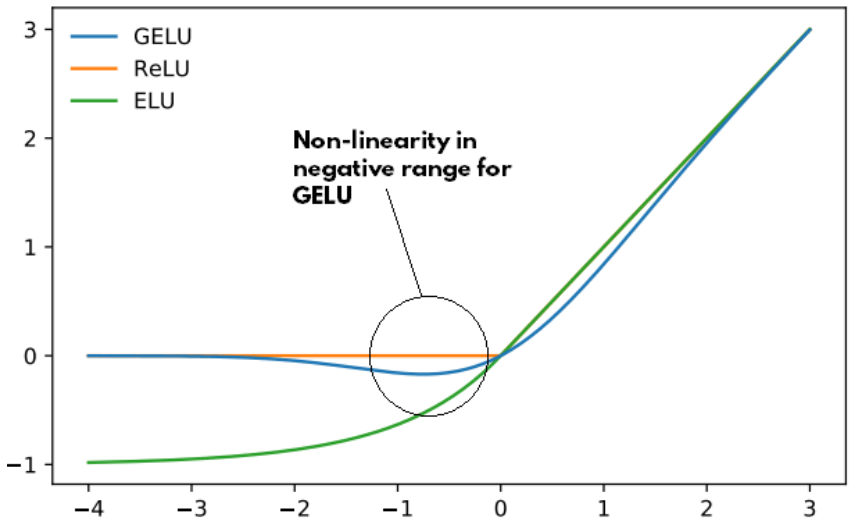

Figure 2.3: *The different Linear Unit functions compared, with the negative dip in GELU emphasised by Buss (n.d.).*

2.2.5 Machine Learning Model Evaluation Metrics

After an ML model has been built and fine-tuned, its performance needs to be evaluated by having it predict the labels of a test dataset that it has not seen before and comparing the predictions with the true dataset. There are different metrics that can be used, some with more drawbacks than others, but the following is a summary of the most prevalent ones for binary classification.

Accuracy The simplest measure of performance, accuracy, is defined as the count of correct predictions divided by all predictions. However, this metric suffers from the Class Imbalance problem, which is the case when one class/category is more prevalent than the other, e.g., real news being more frequent than fake news. In such a case the model might learn to predict only the majority class, and this would result in a high accuracy. If T and F stand for 'True' and 'False', e.g., correctly and incorrectly predicted, and P and N stand for 'Positive' and 'Negative', e.g., real and fake news, then accuracy can be defined as

$$A = \frac{TP + TN}{TP + TN + FP + FN}.$$

Precision The definition of precision can be seen as the "exactness" of a model, i.e., of all of the predictions about one class, e.g., real news, how many were correct. This metric encourages the model to make fewer unsafe predictions since just one correct prediction would result in 100% accuracy as the formula for precision is

$$P_r = \frac{TP}{TP + FP}.$$

Recall The antithesis of precision, recall, measures the "completeness" of the model, i.e., how many predictions of the total amount of a class, e.g., real news, the model predicted correctly. Because this metric does not measure incorrect predictions, it encourages the model to make more unsafe predictions to be more "complete" since the formula for it is

$$R = \frac{TP}{TP + FN}.$$

F1 The F1 score of a model is a metric that combines both precision and recall to find a balance for both. Therefore, the formula for that is the harmonic mean of both, i.e.,

$$F_1 = \frac{2 \cdot P_r \cdot R}{P_r + R} = \frac{2TP}{2TP + FP + FN}.$$

Confusion Matrix The confusion matrix is a simple 2x2 table showing the counts for all of TP, TN, FP, and FN, usually with a coloured gradient to help visualise the difference in the amounts. This table is useful for identifying whether or how much any correct or incorrect predictions are unbalanced, and the metric for the models of the project can be found in Section 6.1.2.

2.3 Existing Solutions

Because of the decades of research in the aforementioned fields, this section summarises the relevant research in machine learning and natural language processing in classification tasks, and the limitations that this project tries to rectify.

2.3.1 ML in Classification

ML has been used to classify a diverse range of items in a lot of fields, including health care, cyber security, and abundantly in NLP (Mathews 2019). In cyber security, ML classification has helped distinguish between malware and harmless software. In health care, ML classification has been used to label tumours as either benign or malignant. Both classifications used multiple features to analyse the behaviour and nature of the items, but neither used NLP since the features did not include natural language. However, as described by Mathews, NLP is an important and highly frequent co-requisite for ML classification tasks.

2.3.2 NLP in Classification

The one example of NLP in Mathews (2019) is, coincidentally, also of Tweet classification. The researchers labelled Tweets as originating either from Hillary Clinton or Donald Trump, two prominent actors in the American political sphere, using NLP techniques to pre-process the lexical data, including a hash vectoriser, which, hashing the term frequency only, does not account for (inverse) document frequency (Pedregosa et al. 2011). The model used in this example was XGBoost, a version of the gradient boosted trees algorithm used in many other ML projects.

There have also been attempts to generalise NLP-based detection for a topic-agnostic model (Castelo et al. 2019), which used certain pre-processing techniques that have been adapted in this project, like NLTK pre-processing.

Apart from the models themselves and as can be seen from Table 2 in the NLP review done by de Oliveira et al. (2021), there have been a multitude of datasets made by researchers, with different types of content (social media posts, short statements, and whole articles), quantity (from 330 posts to 60 million posts), labelling scale of veracity (from binary to six levels), and annotators (crowd-sourcing, journalistic teams, or news agencies).

2.3.3 Limitations

Almost none of the solutions I have found offer a practical tool for non-technical people (either researchers or programmers who can replicate the code in the research papers) to actually use[2]. Even though research papers do not usually include the possible implementations of their models, this paper focuses not only on the creation of the NLP ML model, but also on the practical aspects of implementing it in a Django-hosted web application.

The other limitation of the research is that only a small portion discuss at length the dataset that they use for training the model, which is among the most important aspects of the model because it can highlight biases and skewed perspectives apparent in the dataset. This I have also tried to rectify in the paper.

[2]Although there is one model I found, the model made by Meesad (2021) is in Thai.

3 Requirements

The problem to be solved, as stated in the introduction, arises from the increase of fake news spread online, and was proposed by Dr Chris McCaig, a lecturer at the University of Glasgow. The requirements of the solution to the problem are detailed below using the MoSCoW method, a prioritisation technique frequently used in software development and other fields. The prioritisation (most to least important) is thus: *Must Have*, *Should Have*, *Could Have*, and *Will not Have*. The requirements are also split into functional and non-functional requirements, explained below.

3.1 Functional Requirements

Functional requirements are the active aspects of a software system that users interact with and can see the use of. Discussed below are the functional requirements gathered with the project's supervisor.

Must Have:

- The very first requirement of training a classifying ML model is the acquisition of a dataset with labelled data. This allows the model to learn the characteristics of fake Tweets and real ones and build a network to determine the difference between the two classes.
- After a dataset is acquired, an ML model needs to be built to get the dataset as an input and learn from it.
- The model must be built into a usable tool in order to give it a Tweet or a piece of text from a Tweet and receive back a prediction determining whether it is fake news or not.

Should Have:

- The tool that uses the model should be built as a web application for easier use by non-technical users.
- The web application should be hosted on a domain that is accessible by the average user, not just by university staff or students.

Could Have:

- There could be an option for a user to choose either of the two models for fake news detection, based on how they want to use it, i.e., either text-only or by using different features that can be acquired from Twitter API. This could be a useful feature for extensibility, but is not required by any means.
- It would be beneficial for the user to see what exactly was analysed from their input to the model which predicts the veracity of the input. For example, if the user includes a link to a Tweet, the website could show the text of the Tweet, the author's username, description, and verification status. This would increase the explainability of the model's prediction.

Will not Have:

- Once a prediction has been given to the user's submitted input text/Tweet, the tool could have input for feedback about how well the model did, i.e., whether the prediction was accurate or not, and fit the model according to the new input. However, this could be complicated to implement and could easily be subject to abuse by users who purposely flag fake news as real news or vice versa.
- It was discussed that the tool could be in a different form than a website - a browser extension. However, it was also decided that browser extensions, while being more bespoke and easier to adapt to a specific website, are too narrow in their features, which might be crucial if or when Twitter API and UI change, while also being less understandable and accessible to non-technical users, most of whom know what websites are but only some of whom are aware of browser extensions.

3.2 Non-Functional Requirements

Non-functional requirements are the attributes of the software system that are required, i.e., characteristics that improve the quality, performance, and effectiveness of the project, but do not do so in a very visible way to the general user. Discussed below are the non-functional requirements gathered with the project's supervisor.

Must Have:

- As stated above by de Oliveira et al. (2021), the average Internet user has about 54% accuracy when distinguishing fake news from real news. Therefore, the model must achieve at least this accuracy to be useful for Internet users; otherwise, there is no usable tool developed for the project.
- The speed at which the class for one item is predicted must be reasonable for a user not to give up on using the tool while it is in the process of predicting.

Should Have:

- While it is acceptable to experiment with datasets consisting of narrower topics, like Covid-19 or climate change, it would be preferable for the final dataset to be broader in scope for better generalisability, for example, if the dataset was about any political news.
- Very often the dataset that is used for training the model is imbalanced, i.e., one of the classes is more frequent than the other. This issue should be fixed either by replicating more instances of the minority class (by generating new ones or repeating existing ones) or by removing enough instances of the majority class to match the minority class. The latter should only be done when there are enough instances of the minority class to make the input to the model large enough for significant training.
- The model should use one of the most modern and efficient algorithms and architectures for both the Natural Language Processing and Machine Learning parts, e.g., BERT.

Could Have:

- The more complicated Twitter model could use multiple features for fake news prediction, including description, username, and verified status of the author of the Tweet. This would be useful for more fine-grained prediction with possibilities for additional text, e.g., the description; however, it is only an improvement on the base model that would work well enough on just the text of the Tweet.
- The model could support tokenisation of emojis in the dataset since they can be a very important part of any text, clarifying or sometimes even changing semantics entirely (especially in the case of sarcasm, e.g., "I love how you never reply back..." if it is accompanied by an unamused emoji) (Felbo et al. 2017). However, most often emojis only add to the sentiment of the text, exclusively on which BERT has already been pre-trained.

- Regarding evaluation, the resulting confusion matrix of predicting fake news from the test dataset could be balanced in terms of incorrect predictions for each class. However, most often the confusion matrix is imbalanced, and there is not much that can be done to fix that.

Will not Have:

- The model's performance could have 100% accuracy for the given dataset. However, this is generally impossible and considered useless since it would always be over-fitted to the dataset and not be generalisable to completely new inputs.

4 Design

With the requirements in place, next it was time to design the project in such a way that they would be achieved with the MoSCoW prioritisation in mind. This chapter gives an overview of the aspects that such a project should include and explains the aspects in more detail.

4.1 Overview

The core elements of the project are outlined in Figure 4.1. As can be seen in the diagram, the first part of the project will be acquiring an appropriate dataset containing both fake and real news from items in a domain. After the initial acquisition, the dataset will need to be processed, both from a cleaning and transformation perspective, in order to get text containing all features to be analysed and further processed for model input. These first steps regarding the dataset are explained in further detail in Section 4.2. As soon as the text is acquired, it will be needing to be processed with Natural Language Processing techniques, the specifics of which are demonstrated in Section 4.3, after which the input can be used for model training, detailed in Section 4.4. Finally, the models will be hosted on a web application, the building of which is outlined in Section 4.5.

4.2 Dataset Acquisition and Transformation

A dataset will be acquired, which is preferably domain-specific, like one containing only posts from Facebook or Tweets from Twitter, in order for it to perform better for a specific purpose since general all-purpose tools are very difficult to make. Next is one of the most critical parts of

[1]The BERT icon is taken from Alammar (2018).

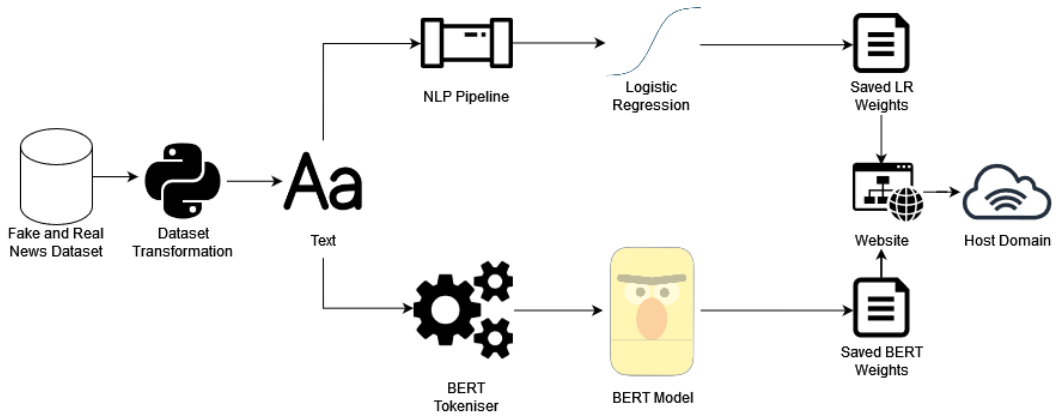

Figure 4.1: *The flow diagram of the project with the core elements*[1].

any Data Analysis project, as described by Elgendy and Elragal (2014): investigation, exploration, transformation, and cleaning of the dataset.

With any division into classes, the balance of the dataset needs to be investigated and corrected, if necessary. If there is a significant imbalance in the counts of instances in the classes, there are two main approaches for the solution: either to generate more of the minority class to reach the number of majority class items (by replicating existing ones or generating new ones based on certain patterns in the existing ones) or to reduce the amount of the majority class by sampling random ones until the amount of the minority class is reached. The latter approach is faster and easier, but it does result in fewer items that the model can be trained on.

Oftentimes the dataset does not have the data prepared in the most appropriate format for how it is planned to be used in the project. For example, if the IDs of Tweets are in a list in one cell in the dataset and they need to be extracted for each row, this extraction is a form of transforming the dataset in order to get appropriate values.

Finally, dataset cleaning is very often necessary for any project since only rarely do researchers need all aspects of a dataset and often the unnecessary parts can be discarded. For example, if there are unnecessary index columns or insignificant IDs of the author of a post, those columns should be removed in order not to fill up memory unnecessarily when the dataset is loaded as a variable. The more in-depth cleaning of the text is detailed in the next section.

4.3 Natural Language (Pre-)Processing

Once the dataset is cleaned, the text that will be the input to the Natural Language Processing model needs to be cleaned as well. This needs to be done according to the pipeline outlined in subsection 2.2.2, i.e., the text should be converted into being all-lowercase, tokenised, and stop words should be removed. Additionally, the text could go through the stemming process, which keeps only the roots of the words, e.g., where "consult", "consultant", and "consulting" all turn into "consult", and also the lemmatisation process, which standardises the different possible forms of words, e.g., where "is", "are", and "am" all turn into "be".

Afterwards, the prepared text can be vectorised with a TF-IDF vectoriser.

To note, this pre-processing stage is only necessary for the simpler model that is not implemented with BERT since the BERT model uses a BERT tokeniser that does everything mentioned above. However, more thought needs to be put into the parameters of the tokeniser, e.g., what the maximum length of all input items should be since the BERT model expects standardised input. This maximum length can be found by analysing all of the inputs that will be given to the model and either reducing or keeping the amount of the maximum length.

4.3.1 BERT Tokeniser Parameters

Attention Masks Because of the strict input standardisation for the BERT model, i.e., having all sentences be the same length, padding is used on the shorter sentences to lengthen them to be as long as the longest one. Attention masks are a simple but effective way for the model to know which tokens to pay attention to (the non-padded values) and which ones not to (the padded values) by representing both kinds of values as an array the length of the sentence, each value (1 or 0) being a label for the specific token in the order.

Special Tokens When BERT was originally trained on 80 million terms, it was trained with special tokens that separated sentences in meaningful ways. The [CLS] token started each independent sentence to be classified (hence, CLS), and the [SEP] token separated two-sentence inputs when training BERT on next-sentence prediction. Because of this format being used to pre-train the model, fine-tuning that does not involve more than 80 million terms is best done

with the same format not to confuse the model. Therefore, the inclusion of these special tokens is almost always the best practice.

4.4 Machine Learning Models

After tokenising and vectorising the input (either manually or with the BERT tokeniser), it can be put into the models. The type of input will depend on which model will be used (TF-IDF vectorised representations for Logistic Regression, and BERT tokeniser-processed sentences for the BERT model). While training these models, it is also important to keep a baseline model to compare the advanced models' performance against. There are many libraries in Python and other languages which provide this functionality.

For Logistic Regression, the input can just be fitted with simple commands in any language, then evaluated with accuracy, F1 score, and confusion matrix.

For BERT, many more decisions need to be made on the parameters for instantiating the model, like epoch size, batch size, and learning rate, detailed below.

4.4.1 BERT Model Training Parameters

Epoch Amount The amount of epochs, being the simplest of the parameters to understand, is how many times the model is trained repeatedly. This parameter clearly correlates with how over- or under-fitted the model is at the end of its training, i.e., the more epochs are used, the more likely it is that over-fitting will occur, and vice versa. This is because the amount of training impacts how closely the model's weights are aligned to the training dataset, but not the validation or testing datasets, which are the real test of the generalisability of the model. Before practically running the training and comparing the resulting losses from each epoch, it is impossible to determine the most suitable epoch amount, which is why it is generally advisable to save different iterations of the model during training, e.g., one per epoch.

Batch size The size of the batch of inputs being put into the model at a time, which directly affects how many times the weights in the model are updated, has an enormous impact on many different aspects of the model's performance, including training time, accuracy, computing costs, and more (Varikuti 2022). A greater batch size means less computational cost and a shorter training time, but most often it also results in worse accuracy and generalisability. This is why the trade-offs need to be considered to balance the available resources and the goals of a supervised learning project.

Learning Rate The learning rate of an ML model is essentially how much the model changes its weights based on new input (or a new row of the same input). If one imagines the loss function as a three-dimensional surface with smaller valleys (local minima) and a biggest valley (global minimum), where being at the bottom of the biggest valley means reaching the minimum possible loss (error), the learning rate means how much each new input would move the current location of the model's loss to a location which has relatively less loss, i.e., is lower. The smaller the loss, the more likely it is that the model could get stuck in a local minimum, but a bigger loss would mean that it might overstep the global minimum and be much less fine-grained in its search.

4.5 Hosted Website

Building the website that will host the models involves four main parts: the backend, the frontend, the models, and the hosting of the website. These are detailed below.

Backend Aspects of the website relating to its backend are the very first to design in order to know how to write code for the web application in general and what the layout of the navigation will be. Among these aspects is the framework to use, the URL navigation, and the interaction between the pages. With the framework in place, the pages need to be set up along with their URLs. For a simple ML tool, having two general pages is enough: one to include the form of the input for the ML model (either raw text or a link to a Tweet) and one to show the results of the prediction, with the possibility to show the user what exactly was used for the prediction, which, in the case of the Tweet, includes multiple additional features, like user description, verification status, etc.

Frontend To make it easier to design the frontend and not work with CSS alone, using a styling framework like Bootstrap is best practice.

Model Inclusion After the ML models are trained, it is possible to save the models' weights as files to keep in the file directory of the website. These can then be loaded into code and used for fake news predictions. Multiple libraries exist that solve this saving and loading problem, e.g., the Python *pickle* library.

Hosting on a Domain The last part of the project is to host the website on a domain. Because there is no financial support for the project and free hosting solutions exist with certain limitations, it is important to design the website with restricted space requirements in mind, e.g., having the whole website's storage to be under 512 MBs. This task is made more difficult by using complicated models with large storage requirements after saving their weights, e.g., the BERT model can take up more than 400 MBs alone.

5 Implementation

5.1 Overview

The general process of implementing the project was done by following the design diagram presented in Section 4.1, with decisions made for specific implementation details along the way. The dataset acquisition and analysis, explained further in Section 5.2, was followed by input building (Section 5.3) for the two models chosen, built, and trained (Sections 5.4 and 5.5), with issues being solved as they appeared, after which their weights were saved to a website made with Django, which was finally hosted on PythonAnywhere (Section 5.6).

5.2 The Dataset

5.2.1 Acquisition

The very first decision in acquiring a suitable dataset was what kind of dataset to look for, especially regarding two aspects: the domain and the topic. Firstly, the domain is an important aspect because of how variable items in that domain can be (either posts in a social media platform or news articles in a news website), the feature diversity that can be acquired from those items, and, more practically speaking, how easy it would be to access these items programmatically with an API. Twitter was chosen as the domain because of these reasons:

- the homogeneity of the text documents, i.e., the fact that Tweets are limited to the same length, which is an important benefit because of BERT always requiring the same input shape (de Oliveira et al. 2021);
- the wide range of features that can be extracted from a single Twitter post, like author's description, verification status, location, etc.;
- the popular use of Twitter by many millions of users, many of which get their facts solely from similar websites;
- the many critical and ongoing changes on the Twitter platform based on changes in CEOs and their controversial policies, some of which have caused fake news and offensive content to spread even more rapidly; and
- the accessibility of Twitter API for free educational use.

Secondly, the topic of the dataset was chosen to be politics because that is a topic that never ceases to be relevant and discussed, as well as being the source of fake news and the place that it is found in most.

Two Datasets Originally, the first dataset that was used to predict news was the *PHEME* dataset with 330 Tweets (Zubiaga et al. 2016). This was done to get accustomed to analysing similar datasets and building models off of them, while also reducing the training time that it would take to fit the dataset to the model. However, the best weighted average F1 score that a simple Logistic Regression algorithm could achieve with this dataset (in the scope of the project) was 61.2%.

Afterwards, the dataset chosen was *FakeNewsNet*, a much bigger Twitter dataset with about 400,000 Tweets just in the political sphere alone (Shu et al. 2018).

Table 5.1: *The original headings of the* FakeNewsNet *dataset.*

id	news_url	title	tweet_ids

```
tweepy.errors.Forbidden: 403 Forbidden
453 - You currently have Essential access which includes access to Twitter API v2 endpoints only. If you need access to this en
dpoint, you'll need to apply for Elevated access via the Developer Portal. You can learn more here: https://developer.twitter.c
om/en/docs/twitter-api/getting-started/about-twitter-api#v2-access-leve
```

Figure 5.1: *The error shown when trying to access a Tweet's text without special privileges.*

5.2.2 Analysis

As can be seen Table 5.1, the dataset is split into articles, which are categorised as either real or fake news (by their inclusion in either *politifact_real.csv* or *politifact_fake.csv* files), and the Tweets which mention this article as a genuine article, which is why the Tweets themselves can be labelled as real or fake news by association. Thus, the original dataset did not have any features of the fake or real news Tweets except for the ids of the Tweets associated with the article (under the *tweet_ids* heading). This is because including any additional information on a large scale is against Twitter's Terms of Service. Therefore, the first part of analysing the Tweets was acquiring their other features, which included the text at the minimum.

The first part for getting access to the Twitter API was making an account and registering all of the security credentials with the service, while also learning how to use the API. However, it was quickly discovered that the part of the API relevant to the project was not accessible by the "Essential" access level alone, as seen in Figure 5.1. Therefore, I needed to apply for the Educational access level to the API, for which the form can be found in Appendix A. This finally gave access to the API necessary for Tweet gathering outlined below[1].

The other features of Tweets that are acquired in Listing 5.1 using code from Kirenz (2022) are the ones specified in the request, i.e., the Tweet-specific ones, like *created_at* (the data and time that the Tweet was created at), *text* (the source text of the Tweet), *lang* (the language of the Tweet), *author_id*, *source* (the device that the Tweet was published on, e.g., a mobile device or personal computer), and the user-specific ones, like *location*, *description*, and *verified* (the verification status of the user), where both sets of features were combined using the id of the author of the Tweet as their user id.

As can be seen in the Listing, the number of Tweets that can be included in one request is limited to 100, which is why this call was made multiple times with batches of 100 Tweet ids in each request until the features for all Tweets were gathered.

[1]However, even this Educational access might be at risk as shown in Mehta and Singh (2023), which further shows the need for two separate models where one is independent of Twitter API, as described in Section 1.2.

```
response = client.get_tweets(IDs[batch_ix : min(batch_ix + 100, n)],
                             tweet_fields=
                                ["created_at", "text", "lang", "author_id",
                                    "source"],
                             user_fields=
                                ["location", "description", "verified"],
                             expansions="author_id")
```

Listing 5.1: *Using the Python* tweepy *library to get Tweet features from both the Tweet and the author.*

```
gs_1 = GroupShuffleSplit(n_splits=1, train_size=.7, random_state=111019)
train_val_ix, test_ix = next(gs_1.split(X, y, groups=X.article_id))
```

Listing 5.2: *Using the* sci-kit learn GroupShuffleSplit *library to split the dataset into a training and validation set*[2] *and a testing set.*

5.2.3 Transformation

After further analysis, one problematic aspect of the dataset was found to be the diversity of languages present. It was found that 11,708 Tweets (7.34%) in the training dataset alone were in a language other than English. While a better BERT model like mBERT could support multiple languages and might achieve even better generalisability if the final project could detect fake news in multiple languages, the simpler, English-only BERT model was chosen, which meant that it could only support Tweets made in the English language. This is further explained in Section 6.3.

Another problem encountered was one that many datasets suffer from: class imbalance. Oftentimes, there is a class that occurs much more frequently than another, and it is the responsibility of the researcher who makes the ML model not to allow for predicting only the majority class every time, which, if the classes are extremely unbalanced, would achieve deceptive results. As discussed in Section 4.2, there are multiple methods of achieving balance between the classes, but because of time constraints, the easiest method of down-sampling was chosen. Therefore, the majority classes were down-sampled as in EliteDataScience.com (2022).

Because of the Tweets in the dataset often containing the same link to a news article they are referring to, it was vital not to have Tweets relating to the same article in both the training and testing sets because the model would easily learn to just look at the linked article and make decisions based on that. This was the reason that even the simple Logistic Regression model achieved 97% accuracy before any fine-tuning, which was too good to be true. Therefore, I extracted the *article_id* for each Tweet in the dataset, grouped it by the article, shuffled and split the Tweets into training, validation, and testing datasets using *GroupShuffleSplit* as from yatu (2020), seen in Listing 5.2. As also noticeable in the Listing, a *random_state* variable was used to keep the evaluation results of the model consistent and not subject to random variation from how the datasets were split.

5.3 Input Building

As already mentioned above, the two different models required two different inputs to be able to be trained and predict text. The varied input building for the models is discussed below.

5.3.1 Logistic Regression-Specific

For the Logistic Regression algorithm, input that was already in the form of sentences only needed to be vectorised with a vectoriser from some library, which in previous projects was the TF-IDF vectoriser since it proved to be the most effective. The whole process of encoding the input (both for training and testing) is shown in Listing 5.3, which demonstrates how much shorter the code for it is compared to input building for BERT, explained below.

[2]This is later split to get the training and validation sets individually, but the code for it (inspired by julliet (2021)) is skipped to avoid repetition.

```
# Get features for TF-IDF
tfidf = TfidfVectorizer()
tfidf.fit(train_text)
train_features = tfidf.transform(train_text)
test_features = tfidf.transform(test_text)
```

Listing 5.3: *Vectorising input for the Logistic Regression algorithm with TF-IDF.*

```
sen_w_feats = []
for index, row in data_df.iterrows():
   combined = ""

   combined += f"I created this Tweet at {row['created_at']} o'clock " + \
      f"in the {row['lang']} language and in {row['location']} " + \
      f"from {row['source']}. "

   verification_status = "verified" if row["verified"] else "not verified"
   combined += f"My name is {row['name']}, username is {row['username']} " + \
      f"and I am {verification_status}. "

   combined += f"I would describe myself as {row['description']}. "

   combined += f"I wrote: {row['text']} "

   # Add the combined text to the list.
   sen_w_feats.append(combined)
```

Listing 5.4: *Combining all features into one paragraph per Tweet that BERT can then analyse.*

5.3.2 BERT-Specific

The reason for input building being much more difficult for BERT in this specific project was the inclusion of multiple features, many of which had different categories: numerical, categorical, and lexical. One of the most important decisions for this project was how to combine all of the different features into a model that accepts all of them. The original idea was to use different feature-engineering methods for each type of feature: numerical features with standardisation of numbers, categorical features with other types of standardisation, and lexical features with Natural Language Processing techniques. This multi-modal approach would have involved not just different types of feature engineering for every feature type, but also a Multi-Layer Perceptron to combine these features and make decisions based on them, as per McCormick (2021*b*).

Because the approach outlined above was too unnecessarily complicated for such a project, a much simpler solution was found in McCormick (2021*a*) by including all of the different features as a combined text paragraph for each Tweet, which was built with a template format as seen in Listing 5.4. An example of the format being filled in with one item's information can be found in Figures 5.4 and 6.5.

After sentences have been made for every Tweet, they have to be encoded with the usual NLP techniques, which are achieved by using *BertTokenizer* from HuggingFace's *transformers*

```
input_ids = []
attention_masks = []
# For every sentence...
for sent in sen_w_feats:
   # ‘encode_plus‘ will:
   # (1) Tokenize the sentence.
   # (2) Prepend the ‘[CLS]‘ token to the start.
   # (3) Append the ‘[SEP]‘ token to the end.
   # (4) Map tokens to their IDs.
   # (5) Pad or truncate the sentence to ‘max_length‘
   # (6) Create attention masks for [PAD] tokens.
   encoded_dict = tokenizer.encode_plus(
                  sent, # Sentence to encode.
                  add_special_tokens = True, # Add '[CLS]' and '[SEP]'
                  max_length = max_len, # Pad & truncate all sentences.
                  truncation = True,
                  padding = 'max_length',
                  return_attention_mask = True, # Construct attn. masks.
                  return_tensors = 'pt', # Return pytorch tensors.
            )

   # Add the encoded sentence to the list.
   input_ids.append(encoded_dict['input_ids'])

   # And its attention mask (simply differentiates padding from non-padding).
   attention_masks.append(encoded_dict['attention_mask'])
```

Listing 5.5: *Encoding every sentence for BERT input.*

Python library with the sentence as input. As seen in Listing 5.5, the tokeniser uses the same parameters that were discussed in Subsection 4.3.1, i.e., special tokens and attention masks, but other parameters are specified as well to improve consistency (in the case of *max_length* and *truncation*) and efficiency (in the case of *return_tensors* being *PyTorch* tensors, which are from another package used in the project).

5.4 Logistic Regression Model Training

As in building input, training the Logistic Regression algorithm was much simpler compared to its BERT counterpart, as seen in Listing 5.6, where both training and evaluation is fit into 4 lines of code. The *evaluation_summary()* function is a simple Python function that includes a description and two metrics from the *sci-kit learn* library: Classification report, which includes F1 score, accuracy, and other metrics, and Confusion matrix, explained in Section 2.2.5.

5.5 BERT Model Training

The BERT training code was adapted from McCormick (2021*a*).

```
# Logistic Regression with TF-IDF
lr_tfidf = LogisticRegression()
lr_model_tfidf = lr_tfidf.fit(train_features, train_labels)
lrtfidf_test = lr_model_tfidf.predict(test_features)
evaluation_summary("LR (TF-IDF) test", test_labels, lrtfidf_test)
```

Listing 5.6: *Training and evaluating the Logistic Regression algorithm.*

5.5.1 Optimisation

The model training inspired by McCormick (2021*a*) uses an optimiser called AdamW, which adapts step sizes based on individual weights during training (Graetz 2018). This optimiser step is important to call before calling the learning rate scheduler step, which also optimises the training of the model by dynamically changing the learning rate hyperparameter.

5.5.2 Memory Issues

One of the biggest issues when trying to implement BERT's training was how much memory the model was needing to encode all of the approximately 200,000 (extant) Tweet training dataset. Neither running the code locally nor on Google's Colaboratory platform with GPU-acceleration helped. Therefore, the solution was to trade memory for speed of training with a parameter called gradient checkpointing, which achieves a balance of two memory-saving techniques: saving all activations on the forward pass of training and recomputing all of them on the backward pass (Hugging Face Documentation n.d.). This method, while making training longer, allowed to fit the model into memory and start training it.

5.5.3 GPU Acceleration

After solving the memory issue, the first significant problem was that training on a normal CPU would take about 8 days per epoch (approximated). With the suggested epoch amount of 4, the total time of training would take more than a month. This was a problem needing to be solved immediately, and the solution to slow training with a CPU was to use GPU-accelerated hardware that is built for fast and numerous calculations.

The first attempt at using the GPU for faster processing was with the help of Google's Colaboratory service, which offers free GPU computation for a limited time and amount every month. By moving the computational units, like PyTorch tensors, to the GPU, where they were used for computation, the training time approximation changed from 32 days to about 20 hours (for all 4 epochs). However, their computation limits are always changing, "having dynamic usage limits that sometimes fluctuate" (Colaboratory n.d.), which meant that it was difficult to find out certain limits and it was also impossible to keep training running for a continuous 20 hour duration without staying awake for the full 20 hours. Therefore, another solution was required.

The second and final attempt at getting GPU-accelerated hardware was using my own personal laptop. The way to use GPU acceleration on a programmatic level is to install Nvidia's Compute Unified Device Architecture (CUDA) Toolkit and the CUDA for Deep Neural Networks (cuDNN) extension, after which there are not many resources online on how exactly to use it since there are various versions for CUDA, cuDNN, and a limitation on the versions of libraries they can be used for, e.g., Hugging Face and PyTorch. Once Nvidia allowed its users to log in (Ridley 2023), only after some time did I find that PyTorch needs to be installed in the environment with a CUDA version nested inside (while mine was already installed for previous

```
urlpatterns = [
   path('admin/', admin.site.urls),
   path('', project_views.home, name="home"),
   path('result/', project_views.result, name='result'),
]
```

Listing 5.7: *The entire list of pages accessible on the Django-made website.*

projects without CUDA), which then (after an environment restart) allowed PyTorch to finally see and use the GPU available with the message:

> There are 1 GPU(s) available.
> We will use the GPU: NVIDIA GeForce RTX 2060

5.5.4 Validation Loss

During the training of the model, validation loss is calculated and shown in order to see how well the model performs on unseen (validation) data as a trade-off to performing too well on seen (training) data. This is to perform over-fitting in the long run.

5.6 Website Building

5.6.1 Django

The URL structure is simple and can be seen in Listing 5.7, where the *home* view uses the *index.html* template and the *result* view uses the *prediction.html* template, both templates inheriting from the *base.html* template.

The two view functions mentioned in Listing 5.7, *home()* and *result()*, are the two functions responsible for the two pages users can visit: the home page and the prediction page, respectively. *home()* just displays the *index.html* template, but *result()* takes as input the model chosen by the user and the text input (either just text or a link to a Tweet), builds the appropriate input for the chosen model, uses the model to predict the veracity of the input, and displays the appropriate template with this prediction.

Standardisation of the pages the users can visit can be achieved by taking advantage of the aforementioned template inheritance offered by Django, which allows setting a base template that all (or any number of) other templates will use, with specific blocks set in place for customisation. For example, seen in Listing 5.8 is the title HTML tag in *base.html*, which includes a block for customising the specific tab title of each page, like "Home page" for *index.html* and "Prediction" for *prediction.html*. The full template can be found in Appendix B, which includes linking Bootstrap for implementing a more modern UI and another example of inheritance.

5.6.2 User Interface

The home page can be seen in Figure 5.2. A lot of white space was used around the core functionality of the fake news detection tool in order for the user not to be overwhelmed with too much functionality and bloat, similarly to Google Search's simplistic approach. Instead, the core elements were kept for the user to interact with: the title of the website, Fake News Classification, which always redirects the user to the home page; the two radio select buttons for the user to choose between which type of model they want to use; the input text area either for the text of

```
<title>
   {% block title %}
   {% endblock %}
</title>
```

Listing 5.8: *Using Django template inheritance in* base.html *to have a standardised interface between the two available templates.*

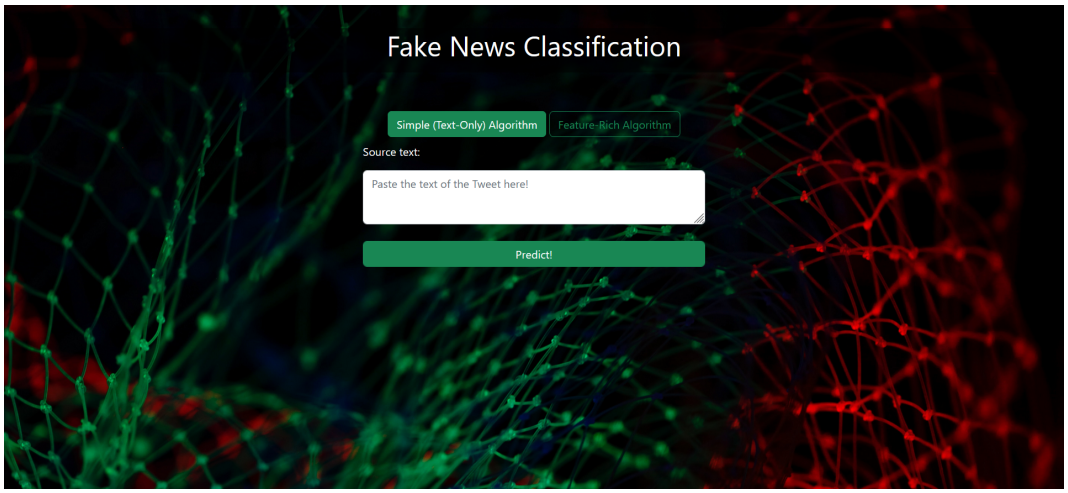

Figure 5.2: *The home page of the website with the Logistic Regression model chosen.*

the Tweet in the case of Logistic Regression or for the link of the Tweet in the case of BERT; and, finally, the submit button. In every piece of writing on the website, simple language is used for the average, non-technical user to understand, which is why Logistic Regression nor BERT is ever mentioned, but their characteristics ("text-only" or "feature-rich", respectively) are used.

The prediction of the input by the Logistic Regression model can be seen in Figure 5.3 for a fake news Tweet taken directly from the dataset. As visible in the Figure, not only is the prediction of the model shown, but the user can also see what exactly was analysed to get that prediction, i.e., they can see the input text they entered at the same time as the prediction. While it is less beneficial to see the same exact text while using the Logistic Regression model, the benefits of this can be better seen in the case of the BERT model, as shown below.

The prediction of a real news Tweet taken from the dataset by the BERT model can be seen in Figure 5.4. As visible in the Figure, the resulting paragraph from all features being combined into one text (outlined in Subsection 5.2.2) is displayed to the user, who can see exactly what is analysed to get BERT's prediction of the Tweet they submitted as input.

To change the placeholder text of the input area for the user according to what type of model they have chosen, jQuery was used in conjunction with setting the *name* attributes of the radio select elements to *model_select*, while also specifying their respective values in the *value* attribute of the HTML tag. With these elements and attributes in place, tag selection could be used to find and change the text area's placeholder with the code shown in Listing 5.9.

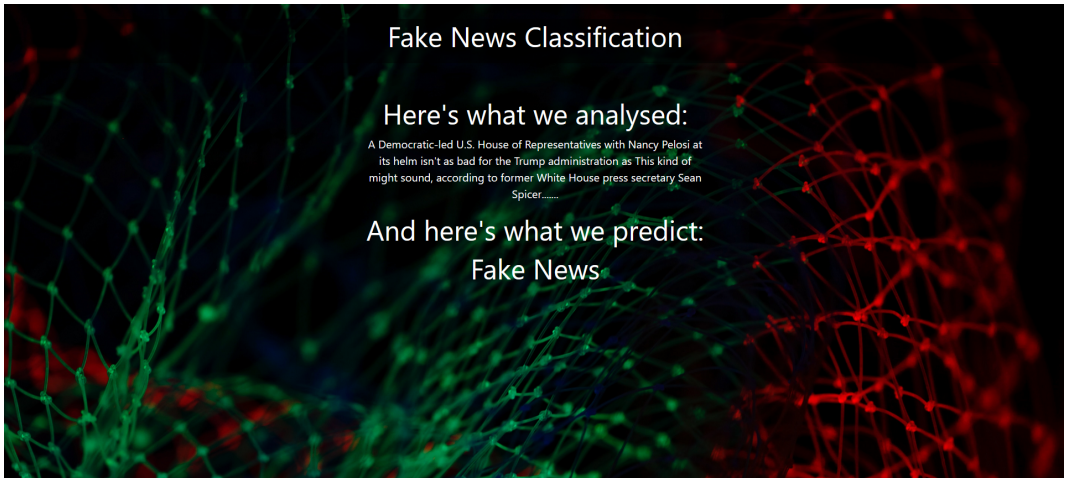

Figure 5.3: *The prediction of text-only input by the LR model.*

Figure 5.4: *The prediction of the Tweet link input by the BERT model.*

```
$('input[type=radio][name=model_select]').change(function() {
   if (this.value == 'simple_model') {
      $('form').find("textarea[type=text]").each(function(){
         $(this).attr("placeholder", "Paste the text of the Tweet here!");
      });
   }
   else if (this.value == 'complex_model') {
      $('form').find("textarea[type=text]").each(function(){
         $(this).attr("placeholder", "Put the link of the Tweet here!");
      });
   }
});
```

Listing 5.9: *Changing the input field's placeholder according to which model has been chosen using* jQuery.

```
regex_numbers = re.findall('/status/(\d+)', link)
   if len(regex_numbers) == 0:
      return "no link"
   tweet_id = regex_numbers[0]
```

Listing 5.10: *Using a regular expression to find the ID of the Tweet from a link to it.*

5.6.3 Form Submission

The input for the BERT model on the website is a simple link to a Tweet that the user wants to analyse. In order to fetch that Tweet's features programmatically, the ID of the Tweet is acquired from the provided link with the help of a regular expression (regex) as seen in Listing 5.10, which, if found, is passed to the aforementioned *tweepy* client that fetches the features of the Tweet for analysis by BERT. As seen in the Listing, the regex finds all numbers after a certain phrase and assumes the correct one is the first because there can be many extra numbers attached to the URL passed in from referral links, etc. Additionally, the regex depends on the URL scheme of Twitter not changing in the near future because, as of March 2023, the ID of the Tweet follows directly after the */status/* subdirectory. If a user submits an incorrectly formatted link or the URL scheme changes, the prediction returns the error "That does not seem to be a valid link to a Tweet." to the user. If the Tweet is removed from Twitter, as sometimes happens especially to fake news Tweets that become popular, the prediction shows "It seems that the Tweet has been removed." to the user.

6 Evaluation

6.1 Evaluation Metrics

6.1.1 Standard Performance Metrics

Evaluation of the previously implemented models should take into account the performance of the baseline model. While training the two models, an additional "DummyClassifier" from the *sci-kit learn* library was trained, which counts only the frequency of tokens and nothing else. As can be seen in Figure 6.1 and as expected, its accuracy is little more than 50%, which is almost as good as guessing 50-50, and its F1 score is a low 35.6%. As can also be seen in the Figure, the baseline model could only predict all Tweets as Real News.

The final accuracy of the Logistic Regression algorithm was **82.8%**, with the F1 score being **82.6%**. As expected, the performance of such a simple model with only one feature being analysed is moderately better than the performance from the earlier *PHEME* dataset, which had an F1 score of 61.2%. As for the BERT model, its F1 score and accuracy were much higher: **99.4%** and **99.2%**, respectively. This dramatic increase in performance is partly due to the incredible pre-training of BERT with 80 million tokens and its state-of-the-art architecture of a transformer model, but also because of multiple features being used instead of just the text of a Tweet. The drawback of such a high performance score, compared to the scores achieved in literature with models having been trained with more resources and bigger datasets, is that it is almost definitely too high to be viably generalisable to new input. This is discussed further on.

6.1.2 Confusion Matrix

The confusion matrix for the Logistic Regression model is shown in Figure 6.3. As can be seen, the confusion matrix reflects LR's accuracy and F1 score, but is also predicting more fake news as real news than vice versa, the skewing of which, while ideally not being skewed either way, is problematic for users who will believe the prediction of the model.

The confusion matrix for the BERT model is demonstrated in Figure 6.2, the code for creating it being adapted from Mesquita (2020). As seen in the Figure, at least 99% of both classes (Fake and Real) are predicted correctly, while less than 1% are not. This is reflected in the accuracy and F1 score of the model. As can also be seen by comparing the counts for both classes, the amount of fake news Tweets in the testing dataset is less than half of real news Tweets, which, while it may seem to cause a class imbalance, is not used in the further training of the model, meaning that there is no impact on the model by an imbalanced testing dataset.

When comparing the percentage errors for false predictions, it can be seen that there are about 3 times more real news being predicted as fake news, which signifies that the matrix is skewed towards predicting a Tweet as fake news. While, ideally, this skew should be fixed, there were no solutions found, and, as per the *Could Have* non-functional requirement in Section 3.2, this was left as is. Furthermore, real news being mislabelled as fake news can also be seen as a benefit to the user as they are encouraged to scrutinise news that they read (especially on Twitter), whereas labelling fake news as real news might lead them to trust the fake news even more than before using the tool.

```
Evaluation for: Dummy MF test
              precision    recall  f1-score   support

       False      0.000     0.000     0.000     16264
        True      0.520     1.000     0.684     17627

    accuracy                          0.520     33891
   macro avg      0.260     0.500     0.342     33891
weighted avg      0.271     0.520     0.356     33891


Confusion matrix:
 [[    0 16264]
 [    0 17627]]
```

Figure 6.1: *Standard performance metrics and confusion matrix for the Dummy Most-Frequent (MF) baseline model.*

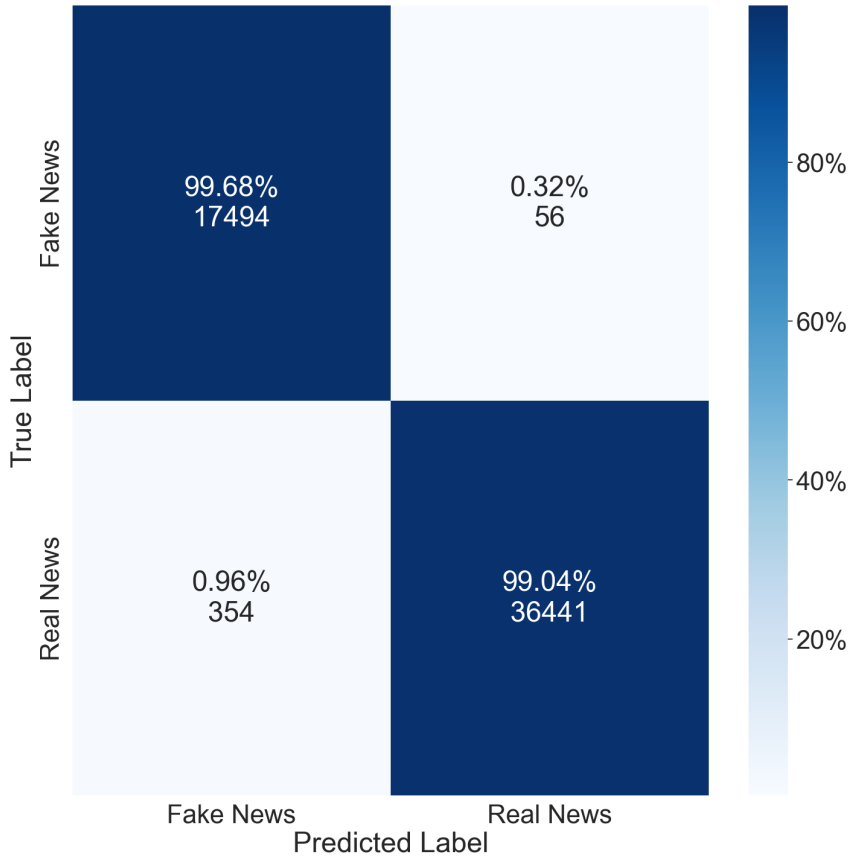

Figure 6.2: *A standardised confusion matrix for BERT after final training.*

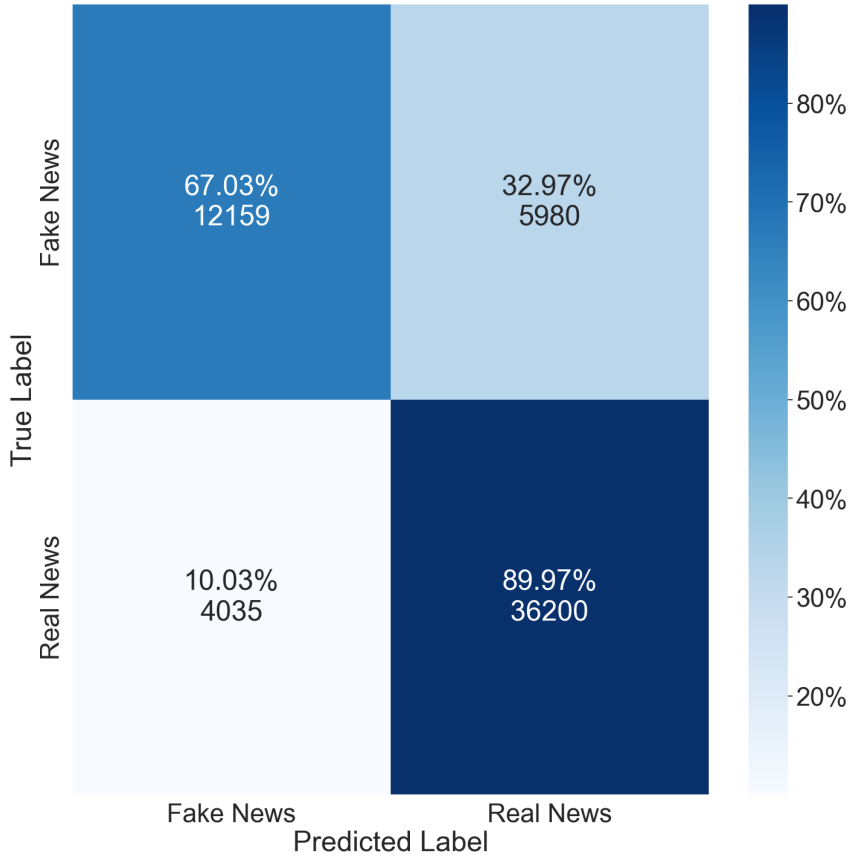

Figure 6.3: *A standardised confusion matrix for the Logistic Regression model.*

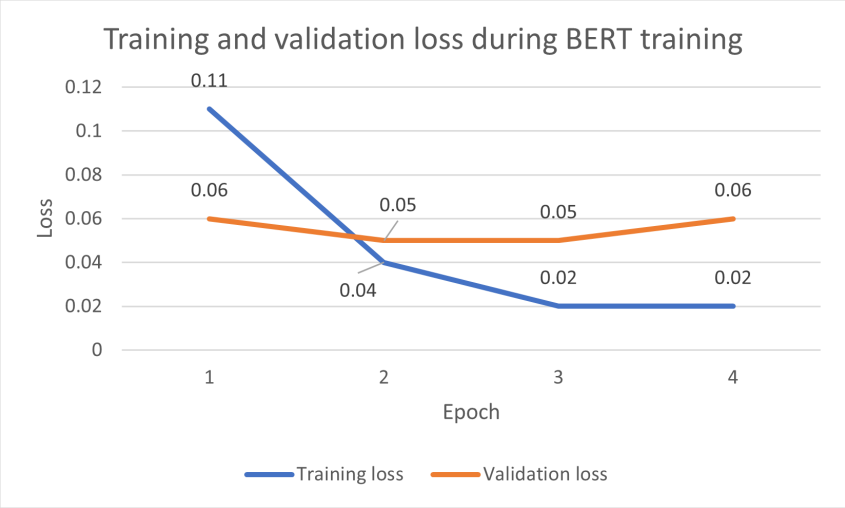

Figure 6.4: *The losses during BERT training in each epoch.*

6.2 Generalisability

Perhaps the most important evaluation metric for a Machine Learning model's performance is its ability to be generalisable to new input that it has not seen before. The more generalisable a model is, the better the likelihood that users would find the tool useful and want to improve their lives with it. While the confusion matrix is useful for testing the model's performance on a testing set that is similar to the training dataset, the training and validation losses over time and the model's performance on a new set of testing values give valuable insight into how useful the tool will be to users.

6.2.1 Losses During Training

The training and validation losses during the training of the model are demonstrated in Figure 6.4. As can be seen in the 4 epochs that were implemented, the training loss continued decreasing until the very last epoch, where it did not noticeably change. In the very last epoch, however, it is visible that the validation loss increased to a value equal to the first epoch's value. This increase in the validation loss signifies that the amount of epochs chosen was too high, resulting in the model being over-fitted to the training dataset. As explained in Section 4.4.1, over-fitting means that the dataset is too dependent on the training dataset and is therefore less generalisable to new, never before seen input.

6.2.2 Modern Input

To check how the models perform with unseen and modern Tweets, a Tweet from a PolitiFact article by Putterman (2021) was chosen, which was labelled as *False* by PolitiFact's Truth-O-Meter score. The BERT model's correct prediction can be seen in Figure 6.5, while Logistic Regression model's prediction was also correct. However, for many other modern fake news Tweets (ones which were not removed because of their infamy, which has happened to many of them due to them being featured on PolitiFact or becoming viral online in general), the prediction was incorrectly attributed as real news. This could be explained by the Tweet in the Figure containing the person-phrase "Bill Gates", which appears more than 50 times more in fake news Tweets than real news ones in the dataset. This and more phrases are analysed in Appendix C. As can be seen, in many cases, politically significant phrases are featured in one

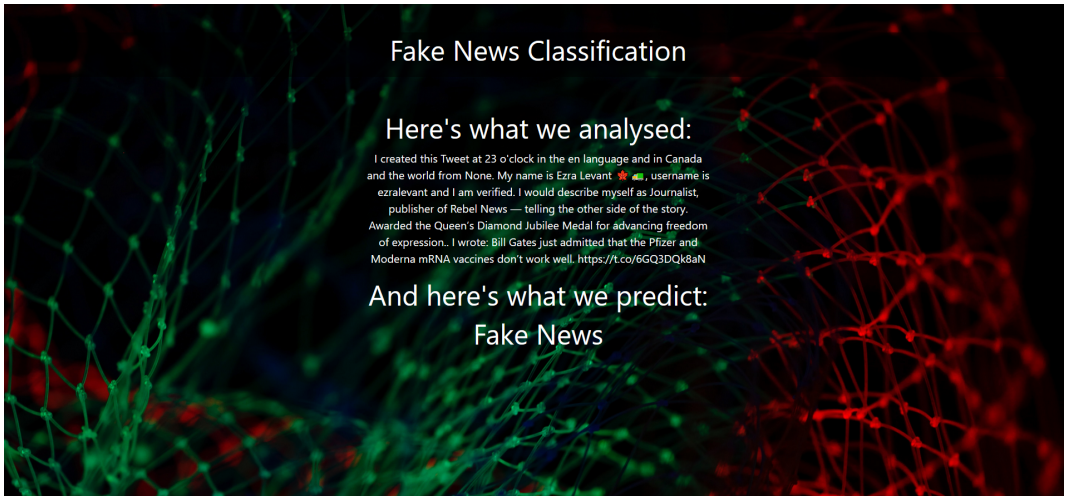

Figure 6.5: *BERT model's prediction on the veracity of the Bill Gates vaccine Tweet.*

class more often than the other, which means that the model has been trained to associate those phrases with the class they are more frequent in.

6.2.3 Potential Reasons Why

Since evaluation leads us to believe that the model is over-fitted to the data, an investigation should be done into the reasons why this is so and how it could be remedied. The three possible reasons why the model is over-fitted are thus: the Twitter character limit change, different political topics, and different phrases for each class.

Firstly, the FakeNewsNet dataset was collected in 2018, one year after the Twitter character limit was increased from 140 to 280 characters. This, however, is another recent policy change with Twitter again increasing the character limit to 4,000 characters for a select user subset (Reimann 2023). These two increases change not only the literal formatting of the Tweet, but also the way that the authors write the Tweet in its language, particularly, conciseness.

Secondly, the political landscape has changed since 2018, with different topics being discussed now and a change in the prominence of even the same topics. Since only political Tweets were used in the dataset, modern Tweets would obviously have very different topics than in 2018.

Finally, even the phrases in the existing dataset help the model lean towards real news or fake news for a particular Tweet containing these phrases. As discussed in Section 6.2.2, more analysis than just the one in Appendix C should be done, which afterwards should be used for a more standardised dataset that focuses on language alone.

6.3 Further Improvements

As mentioned previously, since the dataset came with about 7.34% of Tweets present being in foreign languages, a significant portion of valuable information was lost only because a standard BERT model was chosen and integrated into the project. In order to include these Tweets, it would be beneficial to choose a multilingual model like mBERT for similar datasets not to lose important information. At the minimum, it would also be possible to incorporate a BERT model that supports the second-most frequent language found in the dataset, Japanese, as in Koca (2021).

Another improvement, stemming from analysing the confusion matrix, would be to fix the

skewed nature of incorrect predictions to permit about an equal amount of false positives as false negatives. This could be done by analysing certain phrases that might be skewing the model predictions, like in Appendix C.

Additionally, as seen during evaluation, the validation loss of BERT increased in the final epoch, which means it suffered from over-fitting. During the next training of the model, the losses should be kept in a data structure and the training loop should stop in case the validation loss stays the same or even increases in an epoch, which would reduce the overall over-fitting.

The final improvement worth mentioning is the missed *Will not Have* functional requirement of continuously improving and fine-tuning the model with user input about how well the model predicted the Tweet. This improvement, while being difficult to implement with the delicate nature of user trust and avoidance of misuse, would be an invaluable asset to the project as it would greatly reduce over-fitting because of Tweets being widely varied and it would allow the models to learn continuously as the fake news landscape on Twitter and other social media platforms changes.

7 Conclusion

The Internet has seen an exponentially increasing rise of fake news propagation on every social media platform there is, especially on Twitter now that it is both a frequent source of news for its users and a company with changing leadership, leading to fluctuating policies on fake news detection and reduction.

In order to help Twitter users distinguish between fake and real news, who are about 54% accurate at doing so in general, we proposed to build a usable tool to detect fake news in Tweets, with a Machine Learning model that uses Natural Language Processing as the driving decision-maker. The models were then evaluated with standard Machine Learning evaluation metrics, like accuracy, F1 score, and confusion matrix, and then built into the website, which was hosted online.

To do this, a dataset with about 400,000 Tweets was acquired, analysed, and transformed to extract multiple features of a Tweet, like its text, source, language, and its author's description and verification status. These features were then processed and encoded as input for two different models: Logistic Regression and BERT, a modern Natural Language Processing model with a transformer architecture.

The models were then trained with the given input. Multiple techniques were used to optimise their performance, like TF-IDF vectorising for the Logistic Regression model and AdamW optimising for the BERT model. Additionally, validation loss was used to reduce over-fitting.

Afterwards, the models were evaluated on their performance. While Logistic Regression achieved only **82.8%** accuracy and an F1 score of **82.6%**, BERT's accuracy was **99.2%**, with an F1 score of **99.4%**. While this seems like an impressive score for the dataset, this high of an F1 score implies over-fitting of the model to the given dataset, which is demonstrably shown by not correctly predicting modern fake news Tweets acquired from PolitiFact.

These models were then saved and put into a Django website with Bootstrap, a modern CSS framework, and then hosted on PythonAnywhere. It is available here: `http://fakenewsuofg.pythonanywhere.com`.

With all *Must Have* and *Should Have* requirements in place, this project provides a contribution to the fields of Machine Learning, Natural Language Processing, and, more specifically, Fake News Detection by creating and spreading awareness of an online tool for people to use to help them detect fake news online by simply copy-pasting the text or link of a Tweet into the tool and letting the model analyse it for them. This can then act as a stepping-stone for stronger integrity of news online.

A Twitter API Educational Access Form

How will you use the Twitter API or Twitter Data?

My dissertation project in my final year as an undergraduate University of Glasgow student is about using Natural Language Processing in Machine Learning to build a model that could tell fake and real news distributed on social media apart. As Twitter is a very widely used platform to spread misinformation by malicious users, I want to extract features from Tweets that could be useful in building such a model, like the source text and the time they were posted. With this model, I plan to create a tool (so far the plan is for a website) for users to input the text or URL of a Tweet to better understand whether that Tweet is likely to be fake news, which would hopefully reduce the spread of misinformation on social media.

GitHub link: https://github.com/xid-er/Fake-News-Detection

Please describe how you will analyze Twitter data including any analysis of Tweets or Twitter users.

I plan to extract specific features from Tweets in the fake and real news-annotated dataset I have to build a machine learning model with BERT preprocessing and a Linear Regression model. The features will be: source text, creation time, user description and verification status.

Please describe how and where Tweets and/or data about Twitter content will be displayed outside of Twitter.

The information about these Tweets will be aggregated into the dataset analysis section of my dissertation, which may or may not be published in the end. The same data mentioned above will be analysed, but any usernames or other types of sensitive information will be removed from publication.

B | Base Django Template for Website

Displayed in Listing B.1.

```
<!DOCTYPE html>
{% load static %}

<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">

    <title>
       {% block title %}
       {% endblock %}
    </title>
    <link rel="stylesheet" href="{% static 'css/base.css' %}" />
    <link rel="shortcut icon" type="image/png" href="{% static
        'images/favicon.ico' %}"/>
    <link
        href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css"
        rel="stylesheet"
        integrity="sha384-rbsA2VBKQhggwzxH7pPCaAqO46MgnOM80zW1RWuH61DGLwZJEdK2Kadq2F9CUG65"
        crossorigin="anonymous">
  </head>
  <body id="header" style="--bs-bg-opacity: .8;">
    <header>
      <div class="p-5 text-center">
        <div class="mask" style="background-color: rgba(0, 0, 0, 0.5);">
          <div class="d-flex justify-content-center align-items-center h-100">
            <div class="text-white">
              <h1 class="mb-3"><a class="text-white text-decoration-none" href="{%
                  url 'home' %}">Fake News Classification</a></h1>
            </div>
          </div>
        </div>
      </div>
    </header>
    {% block body %}
    {% endblock %}
    <script
        src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/js/bootstrap.bundle.min.js"
        integrity="sha384-kenU1KFdBIe4zVF0s0G1M5b4hcpxyD9F7jL+jjXkk+Q2h455rYXK/7HAuoJl+0I4"
        crossorigin="anonymous"></script>
    <script src="{% static 'js/jquery-3.3.1.min.js' %}"
        crossorigin="anonymous"></script>
    <script src="{% static 'js/result-jquery.js' %}"
        crossorigin="anonymous"></script>
  </body>
</html>
```

Listing B.1: *The base template, using inheritance in the title and body blocks, with Bootstrap linked in the template.*

C Dataset Phrase Analysis

Bill Gates:

- Fake: 608 / 108_930 (0.56%)
- Real: 44 / 296_170 (0.01%)

Trump:

- Fake: 23_161 (21.26%)
- Real: 59_694 (20.16%)

Clinton:

- Fake: 3049 (2.80%)
- Real: 8842 (2.99%)

Pelosi:

- Fake: 1295 (1.19%)
- Real: 1740 (0.59%)

Vaccine:

- Fake: 6040 (5.54%)
- Real: 568 (1.92%)

Bibliography

Alammar, J. (2018), 'The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning)'. [Accessed January 31st, 2023].
URL: *http://jalammar.github.io/illustrated-bert/*

Alammar, J. (2020), 'The Illustrated Transformer'. [Accessed February 9th, 2023].
URL: *http://jalammar.github.io/illustrated-transformer/*

Buss, J. (2022), 'Activation function GELU in BERT'. [Accessed January 29th, 2023].
URL: *https://iq.opengenus.org/activation-function-in-bert/*

Buss, J. (n.d.), 'Activation function GELU in BERT'. [Accessed March 14th, 2023].
URL: *https://iq.opengenus.org/activation-function-in-bert/*

Castelo, S., Almeida, T., Elghafari, A., Santos, A., Pham, K., Nakamura, E. and Freire, J. (2019), A Topic-Agnostic Approach for Identifying Fake News Pages, *in* 'Companion Proceedings of The 2019 World Wide Web Conference', WWW '19, Association for Computing Machinery, New York, NY, USA, p. 975–980.
URL: *https://doi.org/10.1145/3308560.3316739*

Colaboratory, G. (n.d.), 'Google Colaboratory Frequently Asked Questions'. [Accessed February 26th, 2023].
URL: *https://research.google.com/colaboratory/faq.html#resource-limits*

Coursera (2022), '3 types of Machine Learning You should know'. [Accessed January 29th, 2023].
URL: *https://www.coursera.org/articles/types-of-machine-learning*

de Oliveira, N. R., Pisa, P. S., Lopez, M. A., de Medeiros, D. S. V. and Mattos, D. M. F. (2021), 'Identifying Fake News on Social Networks Based on Natural Language Processing: Trends and Challenges', *Information* **12**(1).
URL: *https://www.mdpi.com/2078-2489/12/1/38*

Dean, G. (2022), 'Twitter users with 2-factor authentication said they were locked out of their accounts soon after Elon Musk said he was ridding the site of 'bloatware". [Accessed February 4th, 2023].
URL: *https://www.businessinsider.com/elon-musk-twitter-users-two-factor-authentication-locked-out-accounts-2022-11*

Deng, L. and Liu, Y. (2018), *"A Joint Introduction to Natural Language Processing and to Deep Learning"*, Springer Singapore, Singapore, pp. 1–22.
URL: *https://doi.org/10.1007/978-981-10-5209-5_1*

Devlin, J., Chang, M.-W., Lee, K. and Toutanova, K. (2018), 'BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding'.
URL: *https://arxiv.org/abs/1810.04805*

DiSilvestro, A. (2022), 'How To Identify Fake News From Real News Online'. [Accessed February 4th, 2023].
URL: *https://www.searchenginejournal.com/fake-news-vs-real-news/459719*

Elgendy, N. and Elragal, A. (2014), "Big Data Analytics: A Literature Review Paper", *in* P. Perner, ed., 'Advances in Data Mining. Applications and Theoretical Aspects', Springer International Publishing, Cham, pp. 214–227.

EliteDataScience.com (2022), 'How to Handle Imbalanced Classes in Machine Learning'. [Accessed February 19th, 2023].
URL: *https://elitedatascience.com/imbalanced-classes*

Felbo, B., Mislove, A., Søgaard, A., Rahwan, I. and Lehmann, S. (2017), Using millions of emoji occurrences to learn any-domain representations for detecting sentiment, emotion and sarcasm, *in* 'Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing', Association for Computational Linguistics.
URL: *https://doi.org/10.18653/v1/d17-1169*

Graetz, F. M. (2018), 'Why AdamW matters'. [Accessed February 27th, 2023].
URL: *https://towardsdatascience.com/why-adamw-matters-736223f31b5d*

Holan, A. D. (2018), 'Politifact - the principles of the truth-O-meter: PolitiFact's methodology for independent fact-checking'. [Accessed January 20th, 2023].
URL: *https://www.politifact.com/article/2018/feb/12/principles-truth-o-meter-politifacts-methodology-i*

Hugging Face Documentation (n.d.), 'Performance and Scalability: Gradient Checkpointing'. [Accessed February 26th, 2023].
URL: *https://huggingface.co/docs/transformers/v4.18.0/en/performance#gradient-checkpointing*

julliet (2021). [Accessed February 26th, 2023].
URL: *https://stackoverflow.com/a/69029384/10197907*

Kirenz, J. (2022), 'Twitter API v2, tweepy and pandas in Python'. [Accessed February 26th, 2023].
URL: *https://www.kirenz.com/post/2021-12-10-twitter-api-v2-tweepy-and-pandas-in-python/twitter-api-v2-tweepy-and-pandas-in-python/*

Koca, D. (2021), 'Japanese multiclass text classification with 97% accuracy using BERT'. [Accessed February 26th, 2023].
URL: *https://towardsdatascience.com/japanese-multiclass-text-classification-with-97-accuracy-using-bert-11b1fdc7c27e*

Lazer, D. M. J., Baum, M. A., Benkler, Y., Berinsky, A. J., Greenhill, K. M., Menczer, F., Metzger, M. J., Nyhan, B., Pennycook, G., Rothschild, D., Schudson, M., Sloman, S. A., Sunstein, C. R., Thorson, E. A., Watts, D. J. and Zittrain, J. L. (2018), 'The science of fake news', *Science* **359**(6380), 1094–1096.
URL: *https://www.science.org/doi/abs/10.1126/science.aao2998*

Mathews, S. M. (2019), "Explainable Artificial Intelligence Applications in NLP, Biomedical, and Malware Classification: A Literature Review", *in* K. Arai, R. Bhatia and S. Kapoor, eds, 'Intelligent Computing', Springer International Publishing, Cham, pp. 1269–1292.

McCormick, C. (2021*a*), 'Combining Categorical and Numerical Features with Text in BERT'. [Accessed February 20th, 2023].
URL: *https://mccormickml.com/2021/06/29/combining-categorical-numerical-features-with-bert/*

McCormick, C. (2021*b*), 'Mixing BERT with Categorical and Numerical Features'. [Accessed February 25th, 2023].
URL: *https://www.youtube.com/watch?v=NbbsVcs42jE*

Meesad, P. (2021), 'Thai fake news detection based on information retrieval, Natural Language Processing and machine learning', *SN Computer Science* **2**(6).
URL: *https://link.springer.com/article/10.1007/s42979-021-00775-6*

Mehta, I. and Singh, M. (2023), 'Twitter to end free access to its API in Elon Musk's latest monetization push'. [Accessed February 26th, 2023].
URL: *https://techcrunch.com/2023/02/01/twitter-to-end-free-access-to-its-api/*

Mesquita, V. (2020), 'Nice Confusion Matrix with percentage cbar'. [Accessed February 28th, 2023].
URL: *https://gist.github.com/mesquita/f6beffcc2579c6f3a97c9d93e278a9f1*

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, E. (2011), 'Scikit-learn: Machine Learning in Python', *Journal of Machine Learning Research* **12**, 2825–2830.

Putterman, S. (2021), 'Bill Gates said we need vaccines that halt transmission. He didn't say the COVID-19 shots don't work'. [Accessed February 26th, 2023].
URL: *https://www.politifact.com/factchecks/2021/nov/17/tweets/bill-gates-said-we-need-vaccines-halt-transmission/*

Qef (2008), 'The logistic curve'. [Accessed March 16th, 2023].
URL: *https://en.wikipedia.org/wiki/Sigmoid_function#/media/File:Logistic-curve.svg*

Reimann, N. (2023), 'Twitter Boosts Character Limit To 4,000 For Twitter Blue Subscribers'. [Accessed March 22nd, 2023].
URL: *https://www.forbes.com/sites/nicholasreimann/2023/02/08/twitter-boosts-character-limit-to-4000-for-twitter-blue-subscribers/?sh=791be5aa5ab8*

Ridley, J. (2023), 'Nvidia's email verification service went down today, but is back now with some delays'. [Accessed February 12th, 2023].
URL: *https://www.pcgamer.com/nvidias-email-verification-is-down-blocking-logins-to-geforce-now-and-geforce-experience/*

Schrael, R. (2022), 'Twitter ends Covid misinformation policy under Musk'. [Accessed February 4th, 2023].
URL: *https://www.bbc.co.uk/news/technology-63796832*

Shu, K., Mahudeswaran, D., Wang, S., Lee, D. and Liu, H. (2018), 'FakeNewsNet: A Data Repository with News Content, Social Context and Spatialtemporal Information for Studying Fake News on Social Media'.
URL: *https://arxiv.org/abs/1809.01286*

Thorn, J. (2020), 'Logistic Regression Explained'. [Accessed January 30th, 2023].
URL: *https://towardsdatascience.com/logistic-regression-explained-9ee73cede081*

Varikuti, M. (2022), 'What Is the Effect of Batch Size on Model Learning?'. [Accessed February 20th, 2023].
URL: *https://pub.towardsai.net/what-is-the-effect-of-batch-size-on-model-learning-196414284add*

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. and Polosukhin, I. (2017), 'Attention Is All You Need'.
URL: *https://arxiv.org/abs/1706.03762*

yatu (2020). [Accessed February 20th, 2023].
URL: *https://stackoverflow.com/a/61337958/10197907*

Zhu, Y., Kiros, R., Zemel, R., Salakhutdinov, R., Urtasun, R., Torralba, A. and Fidler, S. (2015), 'Aligning Books and Movies: Towards Story-like Visual Explanations by Watching Movies and Reading Books'.
URL: *https://arxiv.org/abs/1506.06724*

Zubiaga, A., Liakata, M., Procter, R., Wong Sak Hoi, G. and Tolmie, P. (2016), 'Analysing How People Orient to and Spread Rumours in Social Media by Looking at Conversational Threads', *PLOS ONE* **11**, 1–29.
URL: *https://doi.org/10.1371/journal.pone.0150989*