# A comparison of strategies for detecting biased text

**MSc Data Science & Machine Learning**

**Candidate number: RBCR2**

Supervised by Dr. Alastair Moore

September 2022

# Contents

# List of Figures

# List of Tables

# Abstract

This dissertation investigates different strategies for detecting bias text in Natural Language Processing. In particular, we explore neural network classifiers with word embeddings relating to syntactic and sematic relationships of the word under analysis and its context, recurrent neural networks, pre-trained large language models, and graph convolutional networks.

Furthermore, we show that pre-trained language models display high performance and generalisation in detecting different types of bias, especially when combined with bias-related word embeddings, and bidirectional LSTMs. In contrast, graph networks prove to be computationally expensive and inefficient in the specific task of bias detection, despite thier apparent suitability for text-based tasks.

# 1 Introduction

Natural Language Processing seeks to build computer systems that understand and generate natural languages. If done properly, it has numerous applications ranging from speech recognition and machine translation, to personal assistants. However, it is a difficult field to perfect due to the complexity of language, present in lexical or structural ambiguity, synonymy, anaphoras, sarcasm, or figurative language. In particular, bias language that contains stereotypes or offensive language towards sensitive topics such as race, religion, gender or sexual orientation, or that expresses an opinion when the intention is meant to be factual, can prove extremely damaging in NLP applications.

Writers and editors of text like encyclopedia articles, news, legal contracts, or textbooks aim to be objective, but bias language is ubiquitous, and therefore it is necessary to develop models that strive to detect it and neutralise it in order to avoid carrying such bias in later applications, such as in algorithms processing college admissions, loans applications, or job offers.

Many datasets are already inherently biased, which results in NLP models that are trained with such datasets to perform in a biased manner. A large part of current research work focuses on detecting bias in already existing language models and debiasing them by modifying their word embeddings. A smaller part of the research is actually dedicated to building new models that detect biased text and their bias-inducing words, and propose substitute alternatives for neutralising the bias. The work on this project focuses on the first step of that process: finding model architectures that can detect all forms of biased text in a corpus, ranging from word embedding logistic regression, to pre-trained large language models, to graph networks. In particular, we aim to find the answer to these three questions:

1. *Whether classic neural network task-centred classifiers can have a decent performance at bias text detection as long as the text data is properly vectorized.*

2. *Whether pre-trained large language models can be fine-tuned with bias text data corpus and produce state-of-the-art results with little computational cost.*

3. *Whether graphical networks that are increasingly popular and effective in text-based tasks, are appropriate for bias text detection.*

The first part of this dissertation presents a literature review introducing the concept of bias in text and the different types of bias that can be identified, accompanied with some examples. It also introduces deep learning concepts applied to sequential data like recurrent neural networks, long short term memory cells, and transformers, which are key concepts used in the model architectures explored later on, as well as pre-trained large language models like BERT and BLOOM. We also present the motivation for using graph networks for text classification tasks, and present a transductive model, *Text-GCN*, and an inductive model, *InductGCN*.

We introduce several bias datasets used on which we evaluate the models, including the NPOV Wikipedia corpus, consisting of 180k biased and neutralised sentence pairs that were tagged by editors in Wikipedia's revision history to ensure a neutral point of view in the articles. A series of bias lexicons are included containing words that are considered to be bias inducing.

Finally, we propose several classification algorithms for bias text detection and test them on different datasets to assess how well they can detect different types of data and generalise from different dataset sizes. This includes (1) task-centred models with syntactic and semmantic word embeddings, (2) pre-trained large language models, like BERT and BLOOM, fine tuned with a bias classification corpus, (3) mixed models that combine task-centred and BERT embeddings with bidirectional LSTM cells, and (4) graph convolutional networks that perform both transductive and inductive learning.

Evaluation of model results suggests that pre-trained language models when fine-tuned with the right datasets are better at detecting biased text than graph networks. Moreover, when combined with features that take into account bias lexicons, semantic and syntactic relationships of the words within a sentence and their context, they become even more robust and generalise to unseen types of bias. This work presents an important first step towards building bias detection algorithms with applications in the real world.

# 2 Literature Review

## 2.1 Bias in text

Biased language is defined as containing words or phrases that are offensive, discriminatory, judgemental, or hurtful, especially towards specific groups of people, making them feel alienated or cast out. It can be done on purpose, but most times it is done unconsciously out of ignorance, as a result of social norms and historical oppression. It materialises in an undertone of superiority or misjudgement of certain people based on different traits such as age, race, sex, socioeconomic status or religion.

The detection of bias in text is crucial in applications such as advertisements, recommendations, news, encyclopedias, or scientific texts, and although writers and editors always aim to be as objective as possible, some form of subjective bias still remains [1].

Moreover, textual bias can be transferred to natural language artificial intelligence during the training processes resulting in a biased model that would have negative repercussions in its applications, such as screening candidates in job or university applications, deciding whether a bank should give out a loan to an individual, or whether a landlord should rent their property to a potential tenant. It is important to note that unfair discrimination needs to be systematic and have an unfair outcome in order for it to be biased [2].

Unfortunately, bias is ubiquitous, so detecting it and understanding its nature is fundamental, not only to be politically correct, but to provide fair opportunities and demonstrate an investment in inclusiveness and care for all.

### 2.1.1 Types of bias

In a text corpus consisting of bias sentences and their edits, we can identify three types of bias that can induce such edits: framing bias, which originates from subjective words or phrases that call to a specific point of view, epistemological bias, which is introduced by the presupposing nature of certain words [3], and demographic bias [4].

**(A) Framing bias** is the easiest to detect, since it arises from the use of subjective language that shows the author's stance on a particular topic in question. Such language can be classified as:

1. **Subjective intensifiers**: adverbs or adjectives that add a subjective meaning to the text.

    - The piano player did a **fantastic** interpretation of Chopin's nocturne (biased).
    - The piano player did an **accurate** interpretation of Chopin's nocturne (unbiased).

2. **One-sided terms** show opinionated sides about a controversial subject (e.g., terrorism, religion, politics) that can be seen from several - opposing - points of view.

    - The interviewer asked the new candidates about their views regarding the existing **pro-life** laws (conservative point of view).
    - The interviewer asked the new candidates about their views regarding the existing **anti-abortion** laws (liberal point of view).

**(B) Epistemological bias** involves presuppositions about propositions that are generally considered to be true or generally considered to be false. These presuppositions are implied through the following means:

1. **Factive verbs** are committing to the truth of a proposition.

    - The experiments made by a researcher at the University of Melbourne **revealed** that the consumption of alcohol is related to depression (biased - "revealed" presupposes the truth of the proposition).
    - The experiments made by a researcher at the University of Melbourne **indicated** that the consumption of alcohol is related to depression (unbiased - "indicated" does not presuppose, it simply states).

2. **Entailments** such as verbs that imply the truth or falseness of their complement depending on their polarity.

- The defendant **was compelled to agree** to the outcome of the sentence (biased - "compelled to agree" implies an agreement but by force, so at its core it is a "disagreement").
- The defendant **agreed** to the outcome of the sentence (unbiased - simply an agreement, nothing further can be derived from it).

3. **Assertive verbs** which imply the certainty of a proposition .

   - The analyst **claimed** that 1 out of 10 company clients cancel their membership after the trial period (biased).
   - The analyst **stated** that 1 out of 10 company clients cancel their membership after the trial period (unbiased).

4. **Hedges** decrease the certainty of truth to a statement.

   - London's housing prices have increased **possibly** as a result of the pandemic (biased).
   - London's housing prices have increased as a result of the pandemic (unbiased).

**(C) Demographic bias** involves presuppositions, or stereotypes, about particular demographic categories, such as genders, races, sexual orientations or religions. Examples in this category would include assuming that no women are computer programmers, or that black people are more inclined to be criminals.

Note how framing bias and epistemological bias can both encompass instances of demographic bias:

- The **black** runner was **unjustifiably** questioned by the police (racial anti-stereotype & framing bias).
- The **black** man wearing a hoodie is **most likely** a convict (racial stereotype & epistemological bias).

## 2.2 Neural Networks

As a very brief introduction, a neural network is simply a network of neurons. Each neuron represents a function that has an input $x$ and returns an output $y$. These neurons can be stacked in layers where the output of one becomes the input of the next, and so on, forming a deep neural network that, depending on the activation functions that are applied to the neurons, can approximate any arbitrary function.

Neural networks are trained using an algorithm called back propagation, based on the chain rule of differentiation, that aims to minimize an objective loss function chosen specifically for the type of problem we are dealing with.

### 2.2.1 Recurrent Neural Networks (RNNs)

In text based problems data comes sequentially, the order of arrival of data matters and the past affects the present. Words depend on their context and their meaning is order dependent, a word at the beginning of a sentence may affect a word at the end of the same sentence. Basic neural networks cannot deal with this type of data, since they evaluate inputs one at a time, without taking into consideration other inputs. These tasks are suited for Recurrent Neural Networks, where a state is carried over from processing one element of a sequence to the next.

Similarly to normal NNs, RNNs are composed of layers, weights, bias terms and activation functions, but they also contain feedback loops, which make it possible to use sequential input values to make predictions.

Formally, for a sequential input $x_1, x_2, ..., x_N$ with respective labels $y_1, y_2..., y_N$, a one-directional RNN predicts the label of input at time step $t$ as $\hat{y}_t = g_\theta(x_t|x_1, ..., x_{t-1})$ taking into account the current input and all the past inputs, where $g$ is an activation function, and $\theta$ are the parameters of the RNN. The optimisation is done with an algorithm called Backpropagation Through Time (BPTT) that minimises the following total loss function:

$$Loss(\hat{y}, y) = \sum_{T=1}^{N} Loss_t(\hat{y}_t, y_t) \tag{1}$$

Where the loss at a time step $t$ is the cross-entropy loss function used in

classification problems:

$$Loss_t(\hat{y}_t, y_t) = -y_t \log \hat{y}_t - (1 - y_t) \log(1 - \hat{y}_t) \tag{2}$$

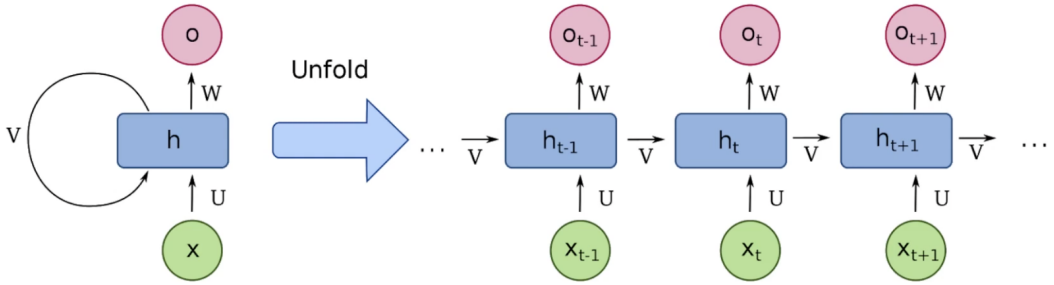



**Figure 1** ***Folded (left) and unfolded (right) RNN architecture.***

There are cases in sequential data where inputs may depend on other inputs later in the sequence instead of earlier on, such as in the case "Teddy bears" and "Teddy Roosevelt". When the network encounters the word "Teddy" it cannot tell if it refers to a toy or a president until it reaches the next input word. This is the motivation for bi-directional RNNs, which form a cyclic graph where information can flow from left to right as in normal RNNs, but also from right to left. In other words, it can use values from anywhere in the sequence to make predictions. For this reason, bi-directional RNNs need the entire sequence of data before they can start making predictions, unlike one-directional RNNs, which can start with the predictions as soon as they get the first input.

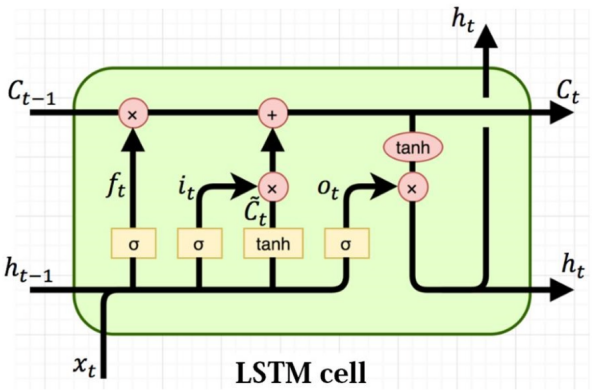
Language has long term dependencies where later layers depend on early ones. However, RNN outputs are influenced by inputs that are close to it in the sequence, and if the network is deep, it is hard to transfer the gradient of later layers to earlier ones, since it is hard to backpropagate the error all the way back to the beginning of the sequence. This causes the problem of vanishing gradients, which is addressed by using special cells that are designed to handle long term dependencies, called "Long Short Term Memory (LSTM)" cells.

### 2.2.2 Long Short Term Memory Cells (LSTMs)

LSTM cells allow a RNN to remember the information it needs to keep hold of context but also to forget the information that is no longer applicable. They can

be seen as adding an internal state to the RNN node, so that in every time step, the RNN cell receives information from the input of the current step, the output of the previous step, as well as additional information from its internal state.

An LSTM cell consists of three parts or gates: a forget gate, an input gate, and an output gate. The forget gate dictates what sort of information that is already stored in the internal state of the RNN node is no longer relevant and can be forgotten. The input gate decides what new information should be added (or stored) to the internal state. Finally, the output gate decides, out of all the information that is stored in the internal state, what part should be output in the current time step. These gates are given values between 0 and 1 stating "how open" should they be. For example, if the forget gate was assigned a value of 0.9, it would mean that the internal state should forget almost all the information it currently holds. This process can be formally expressed with the equations in Figure 2.




$$i_t = \sigma\left(x_t U^i + h_{t-1} W^i\right)$$
$$f_t = \sigma\left(x_t U^f + h_{t-1} W^f\right)$$
$$o_t = \sigma\left(x_t U^o + h_{t-1} W^o\right)$$
$$\tilde{C}_t = \tanh\left(x_t U^g + h_{t-1} W^g\right)$$
$$C_t = \sigma\left(f_t * C_{t-1} + i_t * \tilde{C}_t\right)$$
$$h_t = \tanh(C_t) * o_t$$

**Figure 2** ***The LSTM cell architecture and the equations that describe its gates [5].*** *$i_t$, $f_t$ and $o_t$ represent the input, forget and output gates respectively, and $C_t$ represents the internal state value. $\sigma$ represents the Sigmoid function.*

### 2.2.3 Transformers

Transformers are a new family of Neural Network architectures introduced in the 2017 paper "Attention Is All You Need" [6], that are particularly useful in sequence-to-sequence tasks like text generation or machine translation. The general idea of "attention" is that you learn a weight between each item in the input sequence and each item in the output sequence. Very similarly,

"self-attention" learns a weight between every item in the input sequence and every other item in the output sequence. Attention ignores order, so it is as easy to detect relationships between items very close in a sequence than it is for items very far apart in a sequence.

The transformer architecture performs self-attention several times so that each "head" learns attention relationships independently. Each attention weight is a combination of three matrices: a query vector, Q, a key vector, K, and a value vector, V. Every value in these matrices is initially randomised and then learnt as the network trains.

In a very general way, transformers consist of an encoder and a decoder. The encoder takes the input sequence and transforms it into a numeric representation or embeddings. The decoder takes the encoder embeddings alongside other additional information, such as positional encoding, and creates the output sequence. Both the encoder and decoder are made up of multi-headed self-attention modules that are stacked on top of each other. The transformer architecture is seen in Figure 3.

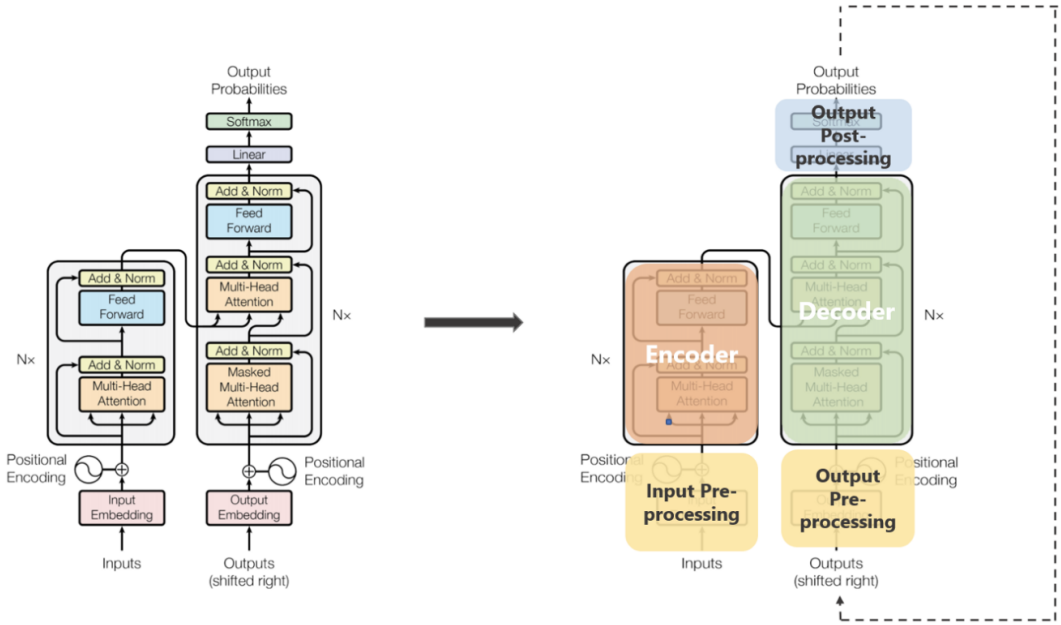



**Figure 3** ***The transformer architecture in detail (left) and a generalised version with its main components (right) [7].***

## 2.3 Tokenization and Tagging

Tokenization is the process of chopping up a document into pieces, called tokens, each of which has attributes that can be used for analysis. Tokens usually go through processing stages before being used for analysis, such as normalisation, stop word removal, or lemmatization. These processing techniques are important to reduce the number of redundant tokens in the corpus and avoid having multiple tokens for the same concept. Punctuation marks can also be tokens.

Tagging in natural language processing is the process of marking up words depending on their role within the sentence they are part of. This can depend on the definition of the word itself and the semantic relationships it has with its context. In this project we use two types of tagging that are used to further define a text sample and extract more features that models use for classification.

### 2.3.1 Part-Of-Speech (POS) Tagging

The part of speech is defined by the Oxford dictionary as the category to which a word corresponds according to its syntactic function, e.g. noun, verb, adjective, adverb. We would be answering the question "what type of word is this?". A word can take different syntactic functions in different texts depending on its context, which will give rise to different POS tags. For example, in the sentence "It is hard to *balance* academic and personal life", the word "*balance*" is a verb, while in the sentence "Joe lost his *balance* as he was walking across the wire", "*balance*" is a noun. Therefore, it is important for POS tagging algorithms to take into account word context when predicting tags, which is a whole research topic on its own.

We use a trained pipeline from *spaCy* to get the POS tags that most likely apply to our text data once it has been tokenized [8]. We use *spaCy* instead of other tools such as "*AllenNLP*" or "*Stanford coreNLP*" for its vast support in the NLP community, easy use, speed, and accuracy.

### 2.3.2 Dependency Parsing

Dependency parsing looks at the relationship between the words in a sentence to analyse its grammatical structure [9]. We would be answering the question

"how is this word related to other words in this sentence?". In this way, there is a dependency or link between every linguistic unit in the sentence. We use the dependency parser from *spaCy* [10].

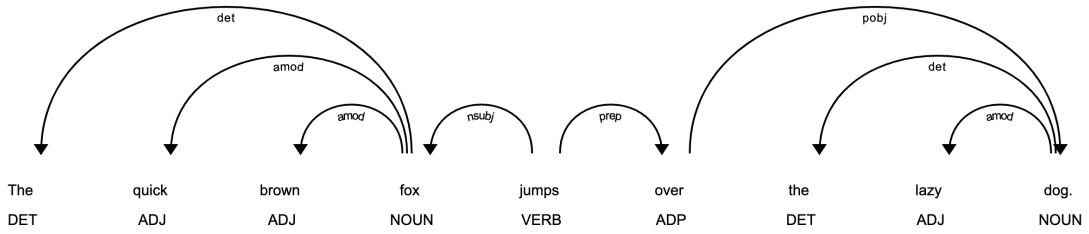



**Figure 4** ***POS tagging and dependency parsing example [9]***

## 2.4 Language Modeling

Statistical language models simply are probability distributions over strings in a vocabulary. They have become one of the leading areas in the NLP realm, and they can be applied to many tasks that usually involve word-by-word prediction, such as topic classification, paraphrasing, text translation, or natural language inference. [11].

These NLP models learn whole languages by being trained with very large data corpuses, and can be applied on sequential tasks. Consider strings in a vocabulary of size $N$: $x_1, x_2, x_3, ..., x_n$. The joint probability of the strings can be expressed as a product of conditional probabilities:

$$P(x_1, x_2, x_3, ..., x_n) = P(x_1) \times P(x_2|x_1) \times ... \times P(x_n|x_{n-1}, x_{n-2}, ..., x_1) \quad (3)$$

And using Baye's rule, we can predict the probabilities of individual strings.

In this project we work with pre-trained language models that have been trained on large corpuses and fine-tune them with task specific datasets. This is much better than training a model from scratch, since in task-focused models, the model needs to learn both the language and the task with a relatively small dataset. In pre-trained models, the model already knows the language, hence the training for the task is much faster and additionally, the resulting model is more robust.

### 2.4.1 BERT

Bidirectional Encoder Representations from Transformers (BERT) is a model proposed by [12]. It is pre-trained on unlabelled text data using a masked language model (MLM) pre-training objective, see Figure 6, and has a bi-directional architecture so that tokens can attend to both preceding and succeeding tokens in the self-attention layers of the transformer. This makes it possible to be fine-tuned with just one additional output layer to produce state-of-the-art results for many text-based tasks. BERT uses only the encoder part of the transformer architecture to produce word embeddings that can be used for other tasks.

The unsupervised dataset BERT is trained on is called Wikipedia and Book Corpus, which consists of the entire English Wikipedia corpus and ten thousand books of all genres. It continues to learn as Wikipedia expands with new articles daily and also when BERT itself is used in practical applications.

The $BERT_{BASE}$ model used in this project consists of 12 transformer blocks, a hidden dimension of 768, and 12 self-attention heads, with 110 million total parameters. There is also an extended version called $BERT_{LARGE}$ that consists of 24 transformer blocks, a hidden dimension of 1024, and 16 attention heads, with a total number of parameters of 340 million.

It is important to note that BERT uses absolute position embeddings, so it is advised to use post-padding, that is to add padding after the input sequence (to the right) instead of before.

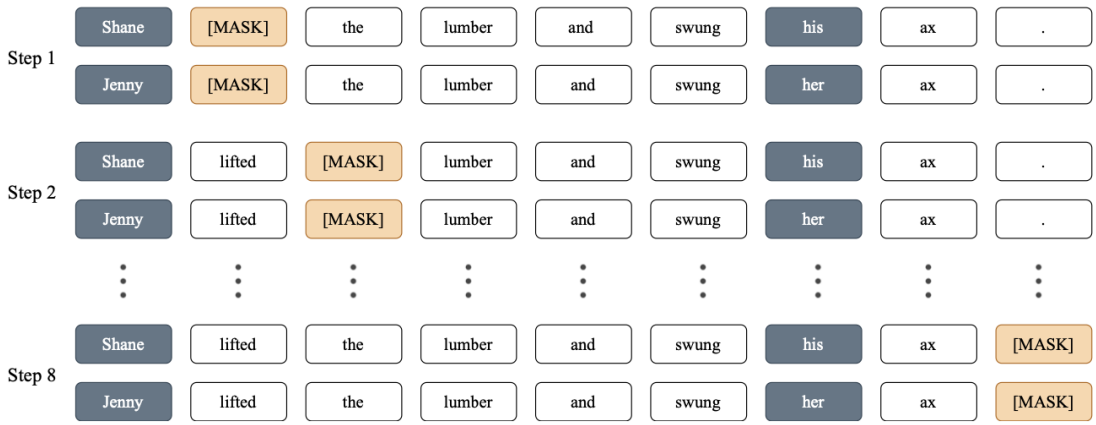



**Figure 5** ***Masking process for a pair of sentences in a masked language model (MLM) [13].*** *The MLM randomly masks some tokens from the input, and learns by aiming to predict the token vocabulary identifier of the masked word based on its context.*

### 2.4.2 BLOOM

BLOOM is an open-source autoregressive Large Language Model that is based on the OpenAI's GPT-3 architecture, and has been recently released by BigScience. BLOOM is trained on 46 languages and 13 programming languages, with a 1.6TB multilingual dataset that contains 350 billion tokens. BLOOM is suitable for a wide range of NLP tasks, and even tasks that it has not been explicitly trained for [14].

With over 176 billion parameters, BLOOM is the first AI model with more than

100 billion parameters [15]. It contains over 3.5 billion embedding parameters, 70 layers and 112 attention heads, 14336-dimensional hidden layers, a cross entropy loss function with mean reduction, and a vocabulary size of over 250 thousand words.

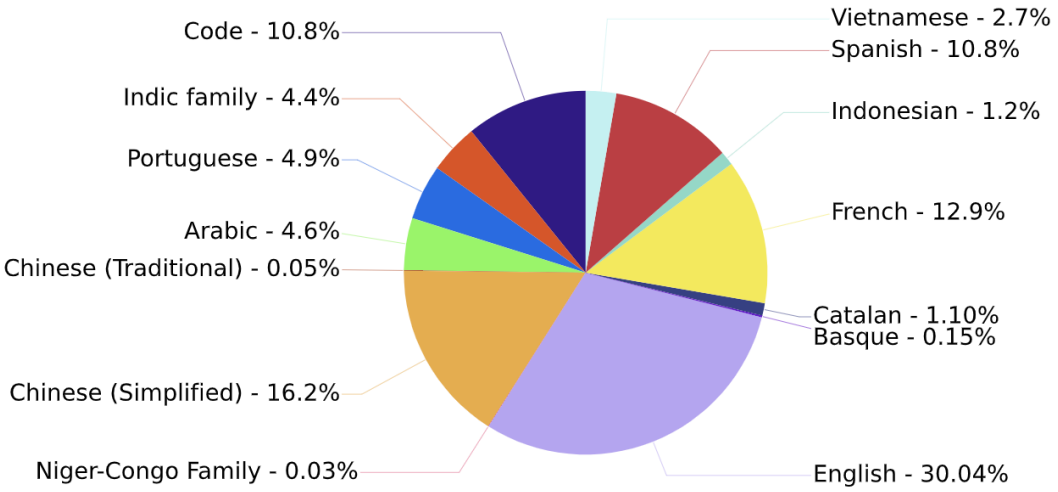



**Figure 6** ***Languages present in BLOOM training corpus.*** *10.8% corresponds to programming languages, the top 5 being Java, PHP, C++, Python and JavaScript.*

## 2.5 Transduction & Induction

In general terms, transductive learning uses specific training cases to predict specific test cases, while inductive learning uses specific training cases to create a general rule that can be applied to predict any test cases.

In transductive learning, both the train and test set need to be present during learning so that the algorithm learns from the labelled training data and also additional information, such as the structure of test data or its attributes. During prediction, it has already seen the unlabelled data it is trying to label, it does not build a predictive model, and therefore, if the learnt model were to be applied to new test data, it would need to be re-trained with the new additional test data. As a result, it may have a higher computational cost due to its need for re-training when new data is added to the model, as well as the fact that it looks at both training and testing datasets when training, and not just the training set.

Inductive learning is the same as the traditional supervised learning. The model learns from a labelled training dataset which can then be used to predict an unseen unlabelled testing dataset. Therefore, it builds a predictive model and once it is trained it can be applied to any new testing data points within the data space without need for re-training, which makes it computationally less expensive.

## 2.6 Graphical Networks

In text classification tasks, the intermediate step of text representation is very important to the performance of the classifier. There have been many studies proving that the text embeddings are key to whether a unique classifier model produces good results [16].

Traditionally, text representation relied on manually constructed features based on lexical attributes of the text, such as bag-of-words and n-grams. Recently, the popularity and development of deep learning has led to text features being learnt by deep learning models, such as convolutional neural networks (CNN) and recurrent neural networks (RNNs), more specifically LSTMs for text classification tasks, since they can apply attention.

However, although CNNs and RNNs focus on locality and hence excel at capturing feature information in consecutive word sequences, they fail to capture long-distance and non-consecutive semantics, which would be for example the word co-occurrence at the start and end of a sentence, or in different documents of a corpus. Graph neural networks (GNNs) on the other hand are very capable of modeling complex text representations, which is why they have become increasingly popular in recent applications involving text-based predictive tasks, outperforming state-of-the-art machine learning and deep learning algorithms.

A graph is a structure for representing data composed of nodes, or vertices, and edges connecting those nodes. A GNN simply applies a neural network to a graph structure, where the target task is to classify its nodes according to the labels provided by the neural network.

In text classification studies, there are two different approaches. The first, as we already talked about, focuses on learning the text embeddings to then use in a classification model, the second, more effectively, focuses on learning the word and document embeddings at once. This is the case of Graph Convolutional Networks (GCNs), a combination of GNNs and CNNs that has been studied in multiple NLP applications, such as text classification, machine translation, or semantic role labelling, achieving state-of-the-art results.

### 2.6.1 Graph Convolutional Networks (GCNs)

A graph convolutional network is essentially "a convolutional neural network inducing the embedding vectors of nodes which are dependent on the property of the neighbourhood" [16].

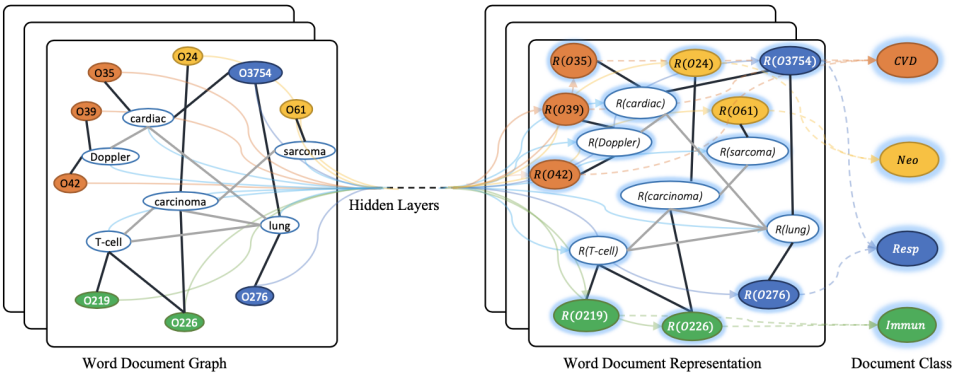



**Figure 7** ***Text-GCN example diagram [16]***

Given a graph $G$ with set of nodes $N$ and edges $E$, and the total number of nodes $|N|$ being $n$, the first layer of a GCN can be expressed as:

$$L^1 = \rho(\tilde{A}XW_0) \tag{4}$$

Where $X \in (n, m)$ contains the graph nodes features of dimension $m$, $W_0 \in (m, k)$ is the weight matrix estimated during training, $\tilde{A}$ is the normalized symmetric adjacency matrix, and $\rho$ is an activation function, such as ReLU.

$$\tilde{A} = D^{-\frac{1}{2}}AD^{-\frac{1}{2}} \tag{5}$$

Where $A \in (n, n)$ is the adjacency matrix of $G$ representing the weight of the edges between two nodes. Note that $A$ contains ones in the diagonal to represent self-loops on every node, so that the aggregate representation of a node includes its own features. The adjacency matrix is normalised in eq. (5) using the degree matrix $D \in (n, n)$ of $A$ in order to avoid vanishing or exploding gradients in cases where nodes have very small or very large values in their features [17].

The number of nodes in the text graph $|N|$ correspond to the sum of the number of documents in the corpus (i.e. number of sample texts in the dataset) and the number of unique words in the corpus (i.e. the vocabulary). In the first

layer of a GCN, the input features of the nodes are simply the identity matrix, $X = I \in (n, n)$, which means that every unique word and every document in the corpus are represented by a one-hot vector.

After going through a GCN layer, the node features are expressed in $L^1 \in (n, k)$, which would in turn serve as the input to the next layer of the GCN. Expressed in a sequential manner:

$$L^{(j+1)} = \rho(\tilde{A}L^{(j)}W_j) \tag{6}$$

$$Z = softmax(L^{(j+1)}) \tag{7}$$

Essentially we are transforming the one-hot note features in $X \in (n, n)$ into a representation in the last layer of the GCN as $L^{(j+1)} \in (n, k)$. In a classification problem, $k$ would be the number of classes, so that after the last layer of the GCN we would apply the *softmax* function and predict a class for each node in the graph, which would be used during training with the target classes in a cross-entropy loss function. Since we are learning to classify all the nodes, we could predict a label for all documents in the corpus (biased texts), as well as for all the words in the corpus vocabulary (bias-inducing words).

### 2.6.2 Graph representation of text

In order to apply a GCN, it is crucial that we represent the text corpus as a graph. The nodes are the documents (texts in the dataset) and the unique words in the corpus. The edges are the word occurrences in the documents (word-document edges) and the word co-occurrences in the corpus (word-word edges). The weight of the word-document edges are calculated by the term frequency-inverse document frequency (TF-IDF) of the word in the document, and the weights of the word-word edges are calculated by the point-wise mutual information (PMI) with a fixed sliding window. Formally, these weights are expressed as:

$$TF - IDF_(i, j) = tf_{i,j} \times \log \frac{N}{df_i} \tag{8}$$

Where $tf_{i,j}$ is the normalised frequency of word $i$ in document $j$, $N$ is the total

number of documents in the corpus and $df_i$ is the number of documents in the corpus that contain word $i$.

$$PMI(i,j) = \log \frac{p(i,j)}{p(i)p(j)} \tag{9}$$

$$p(i,j) = \frac{W(i,j)}{W} \tag{10}$$

$$p(i) = \frac{W(i)}{W} \tag{11}$$

Where $W(i,j)$ is the number of sliding windows that contain words $i$ and $j$ simultaneously, $W(i)$ is the number of sliding windows that contain word $i$, and $W$ is the total number of sliding windows in the entire corpus. Only positive PMI values are kept, since negative values mean there is very little correlation between the two words.

The resulting adjacency matrix containing the weights between nodes $i$ and $j$ can be expressed as:

$$A_{ij} = \begin{cases} TF-IDF(\text{i,j}) & \text{if } i \text{ is document and } j \text{ is word, OR vice versa} \\ PMI(\text{i,j}) & \text{if both } i \text{ and } j \text{ are words, AND PMI}(i,j)>0 \\ 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

### 2.6.3 Inductive GCNs

It is important to note that in standard GCNs it is required that all the nodes from the training and test set are present in the graph during training, which is transductive, because the graph node features are learnt taking into account also the test data. It is challenging for such models to generalise the learned node embeddings to unseen documents, so for new test data the GCN should be re-trained and re-learn the node embeddings including those new test nodes in order to make predictions on the test set. Moreover, transductive models need a relatively large training set to perform better, which requires high computational

resources and storage to build and store the corpus graph. This is where inductive models come in.

Inductive GCNs have an edge over transductive GCNs in the following points:

- They only use the training set corpus for constructing the graph network.
- The resulting graph is smaller, easier to store, and the graph edges are faster to compute with PMI and TF-IDF, which are usually time-consuming operations in large corpuses.
- Training time is shorter and it does not require any re-training to make predictions.
- Their focus is on generalising to unseen nodes by aligning new testing subgraphs to the nodes that the model has already been trained on [16].

In **InducT-GCN**, the graph nodes should still correspond to the documents and unique words in the corpus, but only of the training set. However, unlike Text-GCN, the initial node features do not correspond to one-hot vectors (i.e. the identity matrix) to avoid learning any representation on testing documents during training [18]. Instead, word nodes features are still one-hot encoded, but document node features are represented as a weighted average of the embeddings of the words included in the document. Therefore, the feature value of document $j$ on word $i$ is calculated by $TF - IDF(i, j)$. Note how in this case, the initial node feature matrix $L^{(0)}$ will have shape *(N, num. of word nodes)* instead of *(N,N)*, where *N = num. of doc. nodes + num. of word nodes*. See Figure 8 for an example.

The graph edges weights in the adjacency matrix are calculated exactly in the same way as in Text-GCN: word-word edges with PMI, word-document edges with TF-IDF, and ones along the diagonal to represent self-loops.

Training for InducT-GCN is done in a similar manner to the standard GCN but with the appropriate node feature matrix and adjacency matrix of the training set graph. After training, the weight matrices for each layer are recorded and one more forward pass is done using the test documents nodes only, where the test document node input features and the test batch subgraph adjacency matrix weights are calculated using TF-IDF with the document frequency $df_i$ of the training set.

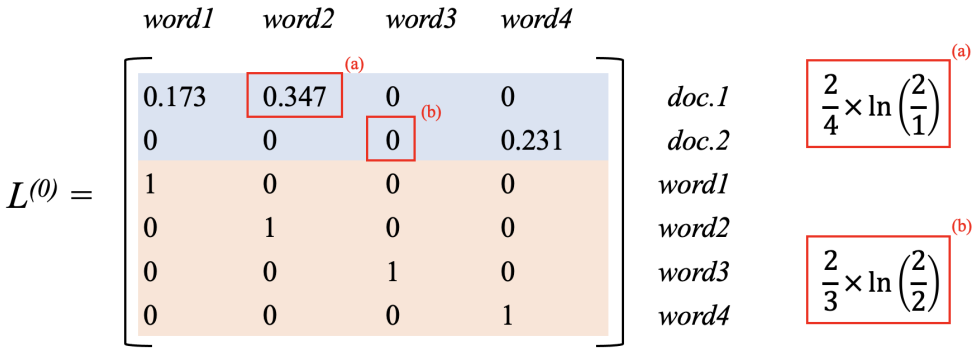



**Figure 8** ***InducT-GCN input array*** Node features array for a corpus of two documents, one composed of "*word*1 *word*2 *word*2 *word*3" and the other composed of "*word*3 *word*3 *word*4". The document node features (in blue) are calculated using the TF-IDF formula in eq. (8).

Predictions are obtained applying the $argmax$ function to the $softmax$ outputs of the network.

## 2.7 Performance evaluation

Detecting bias sentences is essentially a binary classification problem. Therefore, we rely on the most appropriate metrics for this type of problem to evaluate the performance of each model, these being accuracy, precision, recall and F1-score. The confusion matrix is also used as further insight to compare model results.

To get in the mindset of our experiments, the following concepts are already presented in terms of "biased" and "unbiased" classes. In a binary classification problem carried out by a machine learning algorithm, four different outcomes are possible:

- True Positives (TP): "biased" samples that are classified as "biased".
- True Negatives (TN): "unbiased" samples that are classified as "unbiased".
- False Positives (FP): "unbiased" samples that are classified as "biased".
- False Negatives (FN): "biased" samples that are classified as "unbiased".

The aim of the classifier is to purely output TP and TN. However, no model is perfect and it will inevitably make incorrect predictions resulting in FP and FN. A confusion matrix summarises these results:

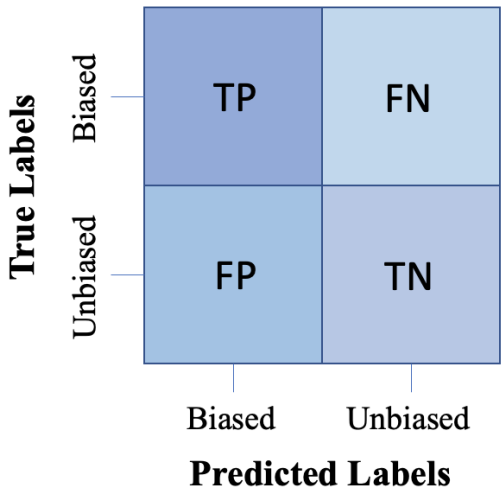



**Figure 9** ***Confusion Matrix***

Based on these terms we can now define the following metrics:

- **Accuracy** is the proportion of correctly classified samples. It is a good measure when the dataset is balanced and has roughly the same number of samples for each class. If the dataset is skewed towards a specific class, accuracy may not reflect the under-performance of the model on the minority class.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{13}$$

- **Precision** measures the ratio of "biased" classified samples that are actually biased. Ideally, precision should be 1, but its value decreases as the classifier incorrectly predicts more samples as "biased".

$$Precision = \frac{TP}{TP + FP} \tag{14}$$

- **Recall**, also known as sensitivity or true positive rate, is the proportion of "biased" samples that are actually classified as "biased". In other words, it represents the proportion of the "biased" samples that the model detected. In an ideal classifier, recall should also be 1.

$$Recall = \frac{TP}{TP + FN} \tag{15}$$

- **F1-score** is the harmonic mean of precision and recall, and is only 1 when both precision and recall are also 1. It is considered a more reliable measure than accuracy.

$$F1 - score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{16}$$

# 3 Datasets

For the purpose of bias detection we collect four different labelled datasets containing different types of biases in order to assess how each model learns from each dataset and how they can extrapolate their learnt knowledge from one dataset and apply it to another. The datasets need to be large enough for the models to detect bias inducing semantic characteristics in the text, since they can be subtle and hard for the models to learn. Unlike in sentiment analysis such as movie reviews where most of the words in the sentence can usually be linked to its classification label, biased samples may differ from their unbiased counterparts in just a few words. For this reason, a series of lexicons of bias-inducing words are also collected. The following subsections aim to explain the datasets used in this project in more detail.

## 3.1 Wikipedia Neutral Point Of View (NPOV)

Wikipedia has a policy called "neutral point of view (NPOV)" which states that its articles should represent the views of all of its sources without bias. Therefore, opinions should not be stated as facts nor facts should be stated as opinions.

Wikipedia keeps a revision history of all the edits articles go though alongside comments explaining the reasoning behind each edit. The NPOV corpus is built from bias-driven revisions between 2004 and 2019 [1] that meet the following criteria [4]:

- The comments of the revision include tags relating to the neutral point of view, such as "(N)POV".
- No more than a single sentence was changed.
- Less than half of the words were changed.
- Edits were more than an established minimum of character Levenshtein distance $< 4$.
- No more than half of the words of the edits were proper nouns.

- Edits do not involve fixing spelling or grammatical errors, adding references or hyperlinks, or changing non-literary elements such as tables or punctuation.

Every one of the 180k samples in the resulting NPOV disputes dataset has six columns, as illustrated in Table 1.

| | |
|---|---|
| **1) Tokenised original text** | ['speculative', 'strategies', 'of', 'life', 'extension'] |
| **2) Tokenised edit** | ['proposed', 'strategies', 'of', 'life', 'extension'] |
| **3) Original text** | "**speculative** strategies of life extension" |
| **3) Edit** | "**proposed** strategies of life extension" |
| **5) POS tags of original text** | "ADJ NOUN ADP NOUN NOUN" |
| **6) Dependency tags of edit** | "amod ROOT prep compound pobj" |

**Table 1:** NPOV disputes data sample example.

Note that the columns in the dataset are presented as rows in Table 1 for visualisation purposes of this report. "Original text" corresponds to the biased text, while the "edits" correspond to their unbiased version.

For the experiments in this report, only the biased sentences and their unbiased edits are used, i.e. columns 3 and 4, each as an individual sample. The biased texts are given a label of "1" and the unbiased edits are given a label of "0". The resulting dataset contains 360k samples each with a "text" and a "label". Tokenisation and tagging is done in the data processing stages using this raw text, for consistency.

## 3.2 Wiki Neutrality Corpus (WNC)

The Wiki Neutrality Corpus (WNC) is parallel to the NPOV disputes corpus corresponding to 380k neutral texts that were adjacent to the biased texts in the NPOV corpus but were left untouched in the edits.

This dataset is presented in the same way as the NPOV dataset, with the same 6 columns. However, in this case the original text and the edit are exactly the same. Therefore, only column 3 (or 4, it does not matter because they contain the same text) is taken into account and all the samples are given a label of "0" (unbiased).

## 3.3 Crow-S Pairs

Crowd-sourced Stereotype Pairs is a dataset collected to assess bias of language models. Although in [13] the dataset is only used for testing models, in this project we use CrowS-Pairs samples during both training and testing.

The dataset is crowd-sourced, meaning that you get more diversity of stereotype expressions and sentence structures. The crowd-workers are asked to write two sentences, one including a stereotype and another including an anti-stereotype of the former. One sentence affects a minority group in the form of nine types of bias: race/color, gender, sexual orientation, religion, age, nationality, disability, physical appearance, and socioeconomic status. The other sentence will be exactly the same as the former, but with the term alluding to the minority group replaced for a term that alludes a majority group. The data is validated using Amazon Mechanical Turk.

| **Bias Type** | **Example** |
|---|---|
| Sexual orientation | **Gay** men care more about fashion than anything else.<br>**Straight** men care more about fashion than anything else. |
| Age | The **young boy** enjoyed the new video game<br>The **old man** enjoyed the new video game |
| Race/color | **Ming** cooked rice for nearly every meal.<br>**Jacques** cooked rice for nearly every meal. |

**Table 2:** CrowS-Pairs dataset examples of stereotypes and anti-stereotypes for different types of bias.

It is important to note that both stereotypes and anti-stereotypes are forms of bias, and therefore, all the sentences in the CrowS-Pairs dataset are given a *label* of "1" (biased).

## 3.4 Stereotype Dataset

This dataset was collected by [19] in order to improve the lack of coverage of other existing stereotype datasets such as Stereoset or Crow-S Pairs, which despite providing a good coverage, they also contain contructed sentences that are forced and would be unnatural in a regular discourse. To this end, it sources appropriate samples from both Stereoset and Crow-S Pairs, as well as text from actual Reddit discussions, specifically from the forums $/r/Jokes$ and $/r/AskHistorians$, both

of which have been proved to contain a high percentage of text with stereotypical associations. In total, this dataset contains 2,221 samples.

The text samples are annotated using Amazon Mechanical Truk, where the annotators specify whether the sentences contain an *explicit stereotype* - text that is intended to express a stereotype -, an *implicit stereotype* - text that is not intended to express a stereotype but it still conveys one -, both, or none (*non-stereotype*).

| **Text** | **Explicit Stereotype?** | **Implicit Stereotype?** | **Label** |
|---|---|---|---|
| My last psychologist was very educated. | no | no | 0 |
| The chess player is always a geek. | yes | no | 1 |
| Yemen is so dangerous it amazes me. | no | yes | 1 |
| Thorn looks like one of those awful trans men. | yes | yes | 1 |

**Table 3:** Samples from Stereotype Dataset containing all possible different annotations.

To get the dataset in the format of our data processing code, the text samples that are annotated to contain explicit or implicit stereotypes, or both, are given a *label* of "1" (biased), while the samples with non-stereotypes are given a *label* of "0" (unbiased).

## 3.5 Mixed Dataset

Given the collected datasets, we make an average-sized (10k samples) mixed dataset composed of the following:

- 6k samples from NPOV Dataset (3k biased and 3k unbiased).
- 2k samples from Stereotype Dataset (1k biased and 1k unbiased).
- 1k samples from WNC Dataset.
- 1k samples from CrowS-Pairs Dataset.

The samples from each individual dataset are selected at random and they vary every time the Mixed Dataset is generated, so we can make experiments with different Mixed Datasets.

## 3.6 Bias Lexicons

A series of bias lexicons with different types of bias-inducing words are collected from [4] which are used for text feature generation. Sentences containing words included in the bias lexicons will be more prone to having an overall bias intention, and therefore, this is accounted for when generating text features during the data processing stage.

The lexicons are lists of:

- Assertive verbs e.g. *think*, *believe*, *claim*.
- Entailments e.g. *accept*, *debate*, *comply*.
- Hedges e.g. *almost*, *guess*, *mainly*.
- Implicative verbs e.g. *manage*, *remember*, *bother*.
- Factive verbs e.g. *realize*, *notice*, *find out*.
- Report verbs e.g. *accuse*, *command*, *inform*.
- Words from the NPOV corpus that were the subject of at least two edits that occurred in at least two different articles. Many of these words represented one-sided or controversial terms [3] e.g. *abortion*, *fascist*, *opponent*.
- Strong subjectives e.g. *overjoyed*, *irreconcilable*, *madly*.
- Weak subjectives e.g. *modest*, *balanced*, *crisis*.
- Negative opinion words e.g. *two-faced*, *absurd*, *brutal*.
- Positive opinion words e.g. *faith*, *immaculate*, *premier*.

# 4 Data Processing

This section aims to explain the processes text data is put through to get it ready for training and evaluation.

1. To start, we express the text in all lower case letters and remove punctuation.

2. We then proceed to tokenize the text by splitting the sentence in its individual words (or tokens). This results in a list of strings.

3. Note that in most NLP tasks, the next step would be to remove stop words. However, since our datasets are made up of short sentences and the bias inducing terms are usually subtle, removing stop words might affect the bias sense of a sentence, and therefore, we keep stop words in the data processing. It is important to note that stop words are removed in graphical network models like GCN and InductGCN. This is to avoid the graph becoming way too large to work with from a computational efficiency perspective.

4. Usually in text-based sequential tasks special tokens that mark start of sentence $< sos >$ and end of sentence $< eos >$ are added at the start and end of each sample. However, we are simply dealing with a classification problem where importance lays on the actual content of the sentence and how words relate to each other, rather than where the sentence starts or ends, like it would be in text generation tasks, for example. Therefore, we decide not to add special tokens.

5. Once the text is tokenized, we assign the POS-tags and dependency tags for each token using *spaCy*.

6. Finally, we generate a training set, a validation set and a test set with a 80/10/10 random split.

# 5 Models

The following section is dedicated to explain the architecture of each of the classification models created for bias text detection. The models can be separated between task-focused models that are trained from scratch with task-specific data, and pre-trained models that have already learnt the English language from large corpuses and are fine-tuned with task-specific datasets.

| **Task-Specific** | **Pre-Trained** | **Task-Specific + Pre-Trained** |
|---|---|---|
| Featurizer | BERT | Featurizer + GloVE |
| LSTM | BLOOM | Mixed Models |
| BiLSTM | | |
| GCN | | |
| InductGCN | | |

**Table 4:** Overview of models.

## 5.1 Neural Network Classifier

This model is a traditional neural network classifier composed of two linear layers with 30% dropout and a final linear layer. A sigmoid activation function is applied to every layer to introduce non-linearity, and in the final layer to get a class probability between 0 and 1. The output probability is then passed through a threshold of 0.5 to make a prediction of class 0 (unbiased sentence) or class 1 (biased sentence).

We use the traditional binary cross-entropy loss function and the Adam optimizer with a learning rate of $10^{-5}$.

The original trait of this model is how the model features are calculated from the text data, using a *featurizer* inspired by [3]. These features describe the word under analysis and also its surrounding context, which we define as a 5-gram window, i.e. two words to the left and two words to the right of the word under analysis. This is important since the bias of many words depends on their context.

Each token in a text sample has features relating to the ID of the token itself in the corpus vocabulary, the POS tags and dependency tags of the token and its context tokens, and the appearance of the token or any of the context tokens in the bias lexicons described in Section 3 (Datasets). If the word (or a word in the

context) is in the lexicon, then the feature is true (1), otherwise it is false (0) [3]. The total number of features per token in a sequence is 100.

The sequences are given a maximum length of 200 tokens. This seems appropriate since there is only a very small percentage of samples in the datasets corpus that contain more than 200 tokens, and the average length is about 20 tokens. The samples that are shorter are padded, and the longer ones simply exclude the tokens after the limit length. This is not expected to disrupt results since it is hardly ever the case, and the bias inducing words will most likely be included in the selected cut.

Therefore, every text sample contains a concatenation of 100 features per token, resulting in a total of 20,000 features. See Figure 10 for a diagram of the architecture of this model alongside the dimensions of the data as it passes through the network.

An additional model seeks improvement by combining these features with GloVe embeddings. GloVe (Global Vectors for word representation) is a pre-trained algorithm for obtaining vector representations for words. It consists of a dictionary where each word (or token) has its own vector representation of length $(50,)$, and there is also a vector representation for token $< unk >$ which accounts for the words that are not found in the GloVe vocabulary. The GloVe embeddings of a sequence of text are obtained by avergaing the embeddings of the individual tokens that make the sequence. The sequence GloVe embeddings are concatenated to the features from the *featurizer* in the last hidden layer of the neural network before making a prediction, see Figure 10.

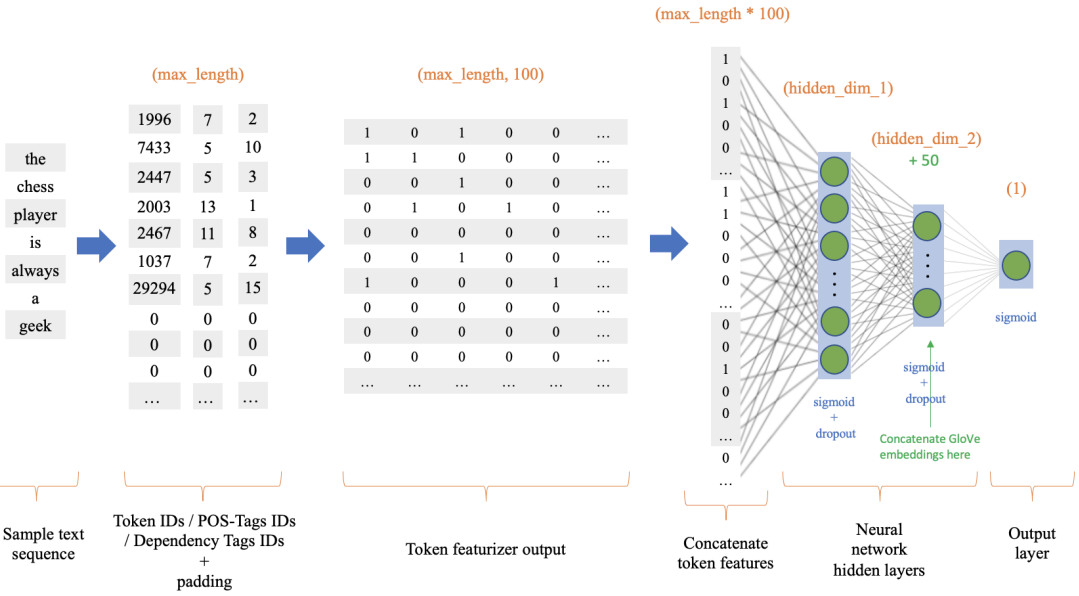



**Figure 10** ***Classic Classifier Model Architecture*** *This diagram illustrates the data processing stages and the neural network, as well as the data dimensions - in orange - on each processing stage and layer. Modifications to account for GloVe embeddings are in green.*

## 5.2 LSTM & BiLSTM

Two models are created following the LSTM and Bidirectional LSTM architectures explained in *Section 2.2*. The word embeddings in this case are input as one-hot vectors of the length of the corpus vocabulary, and trained within the model architecture in the first layer, before being introduced into the LSTM cell. A linear layer with 30% dropout and sigmoid activation is applied to the LSTM output before getting a one-dimensional probabilistic output between 0 and 1. The classifier predictions are made by applying a 0.5 threshold to the model outputs.

We used the standard binary cross-entropy as the loss function, and the Adam optimizer with a learning rate of 0.001.

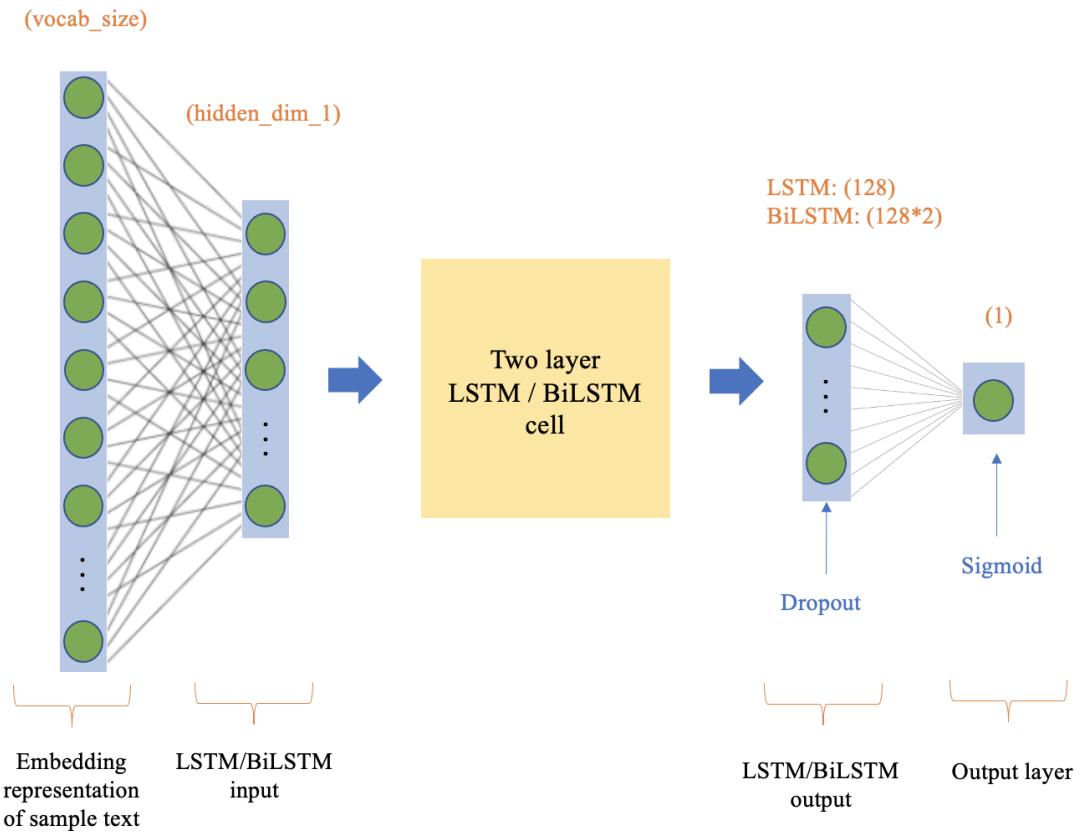



**Figure 11** ***LSTM and BiLSTM Model Architectures*** *All data dimensions are in orange.*

The model architectures are detailed in the diagrams of Figure 11. As expected, the classification accuracy of the BiLSTM on the test set is slightly better than that of the traditional LSTM.

## 5.3 BERT

This model uses a BERT pre-trained tokenizer to vectorize the text data and input it in a BERT pre-trained model. The output BERT embeddings are put through a 30% dropout layer and a linear layer that outputs a one-dimensional logit that is then used to predict a class label. To make the prediction, the logit needs to go through a sigmoid activation function and a threshold of 0.5.

The pre-trained tokenizer produces the BERT model inputs. These are the IDs of the tokens in the sample sequence that map to the actual words in the BERT vocabulary and their corresponding pre-trained BERT embeddings, and an attention mask that avoids performing attention on padding token indices. The

maximum sequence length is set to 200 tokens, with padding to the right of the sequence (post-padding) as suggested by the BERT documentation [12].

Since the BERT layer of this architecture is already pre-trained, the model could already perform fairly well on test data. Nevertheless, the model is fine-tuned using the training data to make it more apt for bias classification. We use a binary cross-entropy loss function with logits, and the Adam optimizer with a learning rate of $10^{-5}$.

The model architecture is visualised in Figure 12.

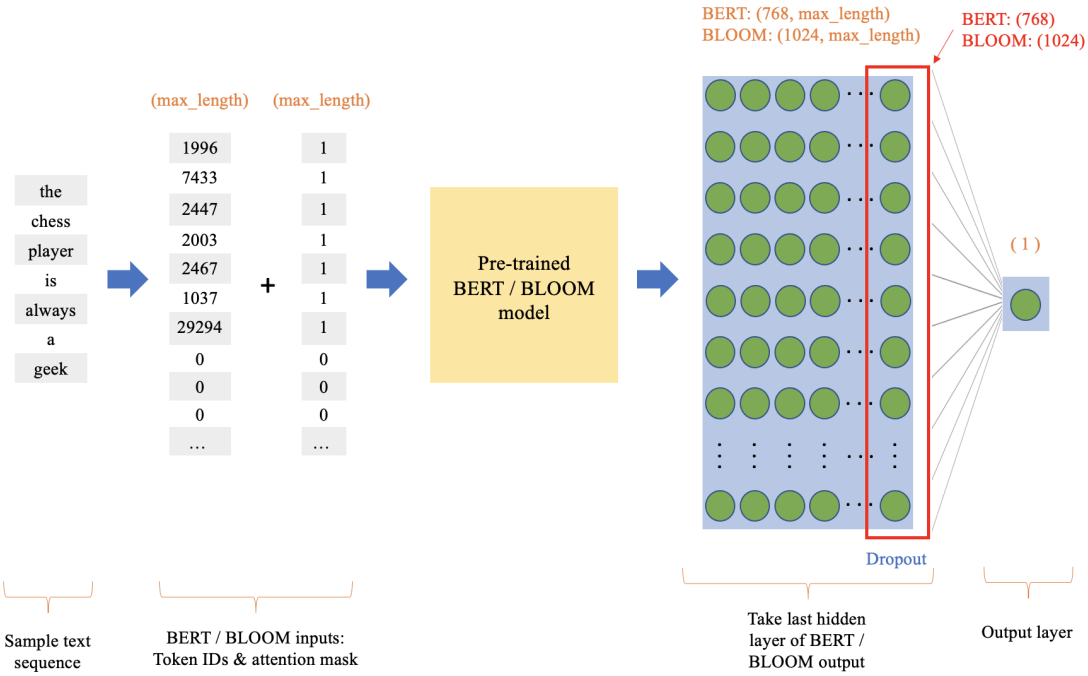



**Figure 12** ***BERT/BLOOM Model Architecture*** *All data dimensions are in orange.*

## 5.4 BLOOM

This model architecture is exactly the same as the BERT model in *Section 5.3*, but it replaces the pre-trained BERT tokenizer and model by the newly available pre-trained BLOOM tokenizer and model. The differences with respect to the BERT model are:

- The embeddings carried over to the linear layer of the network are the last hidden state embeddings of the BLOOM output.
- The data sequences are pre-padded, i.e. padding tokens are added to the

left of the sequence, to achieve a maximum sequence length of 200 tokens. Pre-padding is recommended in this case because we take the last hidden state of the BLOOM output to carry over on the network. If the sequence was post-padded with zeros at the end, the hidden state of the network at the final word in the sentence would likely get "flushed out" to some extent by all the zero inputs that come after the word.

The model architecture is visualised in Figure 12.

### 5.4.1 Limitations of BLOOM

BLOOM is a very large model, containing 176 billion parameters, compared to the 110 million that BERT has. Therefore, it is expected that it requires a very large disk space and high processing power. With the resources available for this project, we could not work with the full model as there was no sufficient disk space and the GPU memory crashed during training. The biggest BLOOM model we could work with was *BLOOM-560m*, a reduced version containing 560 million parameters, an upgrade from BERT but not a significant one.

## 5.5 Mixed Models

In this part we explore model architectures that combine previously introduced architectures. One model combines BERT pre-trained model with a BiLSTM cell. A second model concatenates the last hidden layer output of BERT with the unsqueezed *featurizer* features from *Section 5.1*, and feeds them to a BiLSTM cell, see Figure 13. The third model combines the pooler outputs of BERT with the squeezed *featurizer* features, and feeds them to linear layers with dropout and Sigmoid activations, see Figure 14. All models use the BERT tokenizer to process the text data and are trained with a binary cross entropy loss function with logits, an Adam optimizer, and a learning rate of $10^{-5}$.

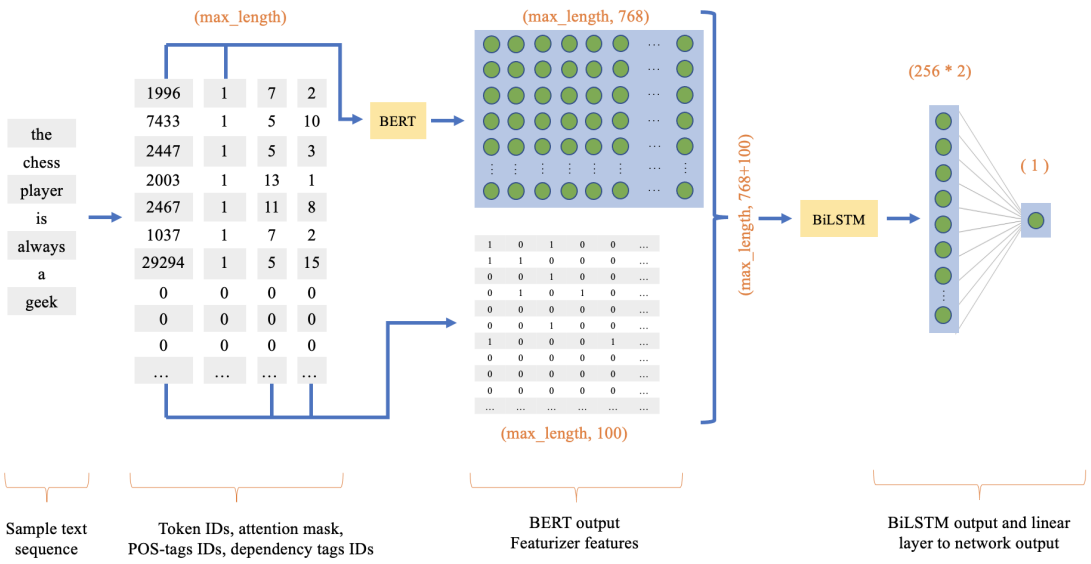



**Figure 13** ***Mixed Model 2 Architecture*** *All data dimensions are in orange.*

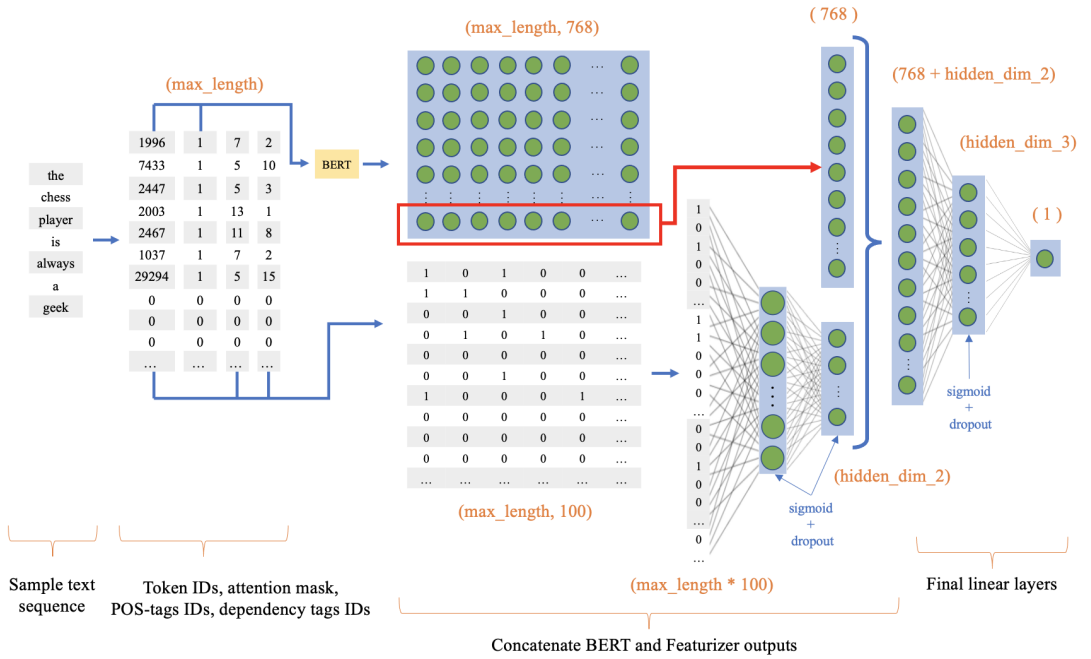



**Figure 14** ***Mixed Model 3 Architecture*** *All data dimensions are in orange.*

## 5.6 GCN

We implement a classic transductive GCN model with two layers, as explained in *Section 2.6.1*. The model takes the text and label of each sample in the dataset, as well as a tag specifying whether that sample belongs to the training or testing set. Using all the samples, the model computes the graph network with word-nodes,

document nodes, PMI edges and TF-IDF edges. The network is trained with the entire corpus information and outputs the accuracy of the test set.

### 5.6.1 Limitations of GCNs

We encountered limitations while implementing graphical networks due to our available RAM memory being too small. We disposed of 54GB of RAM, but since our NPOV corpus was considerably large, containing around 360k sample texts, our RAM could not store the entire adjacency matrix containing the PMI and TF-IDF values between word-word nodes and word-document nodes, that consist of a total of $(360,000 + vocab\_size)^2$ floats i.e. an order of magnitude of $10^{11}$. As a result, we found that the GCN model could only be run with datasets of a maximum size of around 30k samples, which would equal to around 9% of the NPOV corpus.

## 5.7 Inductive GCN

We implement an inductive InductGCN model with two layers, as explained in *Section 2.6.3*. The data inputs are the same as in the GCN, as well as the limitations. However, thanks to the nature of InductGCN only using the training set to compute the graph network during training, the model could be run with up to 20% of the NPOV corpus before the RAM ran out of memory. This is a clear computational improvement from the classic transductive GCN model.

# 6 Summary of Results

## 6.1 Overview

For training, we use the NPOV dataset, the Stereotype - we refer to it as "Stereo" in many instances in this report - dataset and the Mixed dataset, since they all contain samples of each class (biased and unbiased sentences). The WNC and CrowS-Pairs datasets cannot be used for training since they only contain samples of one class, unbiased sentences for WNC dataset, and biased sentences for CrowS-Pairs. However, note that these datasets were used to form the Mixed dataset, and they can also be used for testing, especially the CrwoS-Pairs dataset, to see whether the models have learned how to detect types of bias that do not appear in the training set.

Each model is trained with each of the training datasets, and validated at each epoch. Every time the validation loss decreases, the model is saved, and training stops after two epochs where the validation loss has not decreased. Task-specific models converge after 10 to 15 epochs, while models with pre-trained components converged much faster, after just 2 or 3 epochs.

## 6.2 Results on familiar test sets

Every model is tested on the test set of the dataset it has been trained on, and the accuracy, F1-score and confusion matrix are calculated. Table 5 summarises the accuracy results on these tests, while Table 6 summarises the F1-scores.

| | NPOV dataset | Stereo dataset | Mixed dataset |
|---|---|---|---|
| **Featurizer** | 60.1 | 54.1 | 59.3 |
| **Featur.+GloVe** | 59.4 | 55.4 | 57.0 |
| **LSTM** | 65.9 | 13.9 | 52.0 |
| **BiLSTM** | 63.7 | 74.8 | 54.2 |
| **BERT** | 75.0 | 84.2 | 65.4 |
| **BLOOM** | 71.0 | 83.3 | 66.6 |
| **Mixed 1** | 69.1 | 80.6 | 69.5 |
| **Mixed 2** | 74.7 | 81.1 | 65.8 |
| **Mixed 3** | 74.1 | 79.7 | 67.4 |

**Table 5:** Summary of model accuracy results in % of correctly predicted test samples. Best accuracy model result for each dataset highlighted in green, worst results in red.

| | NPOV dataset | Stereo dataset | Mixed dataset |
|---|---|---|---|
| **Featurizer** | 62.7 | 0.0 | 64.1 |
| **Featur.+GloVe** | 60.0 | 10.8 | 51.3 |
| **BiLSTM** | 58.5 | 71.4 | 61.0 |
| **BERT** | 72.6 | 83.3 | 61.9 |
| **BLOOM** | 68.2 | 82.3 | 73.0 |
| **Mixed 1** | 68.1 | 79.8 | 67.7 |
| **Mixed 2** | 73.3 | 80.6 | 70.7 |
| **Mixed 3** | 70.8 | 76.7 | 65.6 |

**Table 6:** Summary of model F1-score results in %. Best F1-scores for each dataset highlighted in green, worst scores in red.

We can observe the following:

- The best performing models are the pre-trained models such as BERT, BLOOM, or the mixed models. This is very obvious in the Stereo dataset and Mixed dataset. This is because these datasets are fairly small (3k and 10k samples respectively), which does not allow for the task-specific models to learn both the language and the bias detection task properly. Moreover, bias detection is not a simple task to detect, since it may arise from just some specific words in a sentence, unlike in sentiment analysis, where most parts of the sentence may contribute to its classification. Therefore, it is expected that bias detection requires more samples and more training that other NLP classification tasks.

- The Stereo dataset is the highest performing dataset among the pre-trained models because it contains types of bias that are more obvious and easier to detect than those in the NPOV dataset. Once the pre-trained model already knows the language, learning the bias in the Stereo dataset is much simpler than in the NPOV dataset, which is why it has better performance even if it has considerably less samples to train on. However, we would expect the models trained on Stereo dataset to perform poorly on datasets that differ from the Stereo dataset biases, which we explore later on.

- Even though the Mixed dataset contains Stereo dataset samples, a major part of its samples come from the NPOV dataset, but it is 38 times smaller. This is why performance on the NPOV dataset is better than in the Mixed dataset, because the Mixed dataset does not have enough samples to learn properly. In fact, if we look at the wrongly classified sentences from the Mixed dataset, it can be seen that they mostly come from the NPOV dataset.

- The LSTM and BiLSTM perform similarly but do not produce any impressive results on a large complicated dataset. On Stereo dataset however, BiLSTM manages to get a decent performance slightly under the pre-trained models, while the LSTM model gets stuck in a local minima in the first epoch and fails to converge at all. This supports the decision of carrying the BiLSTM over to the mixed models instead of the LSTM.

- The addition of pre-trained GloVe embeddings to the Featurizer embeddings does not seem to improve results at all. Embeddings related to the semantic and syntactic characteristics of words and their context within a sentence seem to be more useful for bias detection than pre-trained GloVe embeddings.

- The addition of a BiLSTM to pre-trained models does not seem to have any improvements. In fact, the addition of syntactic and semantic features seems to have a better effect. And adding both together leads to a better F1-score than BERT alone.

- Accuracy does not show the full picture, because we are specifically interested in bias detection. If most of a model's misclassifications come from bias text

samples, then that does not fulfill our task at all. The F1-score helps visualise this. For example, the BERT model has the best accuracy on the NPOV dataset, but the Mixed 2 model has the best F1-score, meaning that the Mixed 2 model detects more bias texts than the BERT model alone (Figure 15). Therefore, the Mixed 2 model would be preferred. An obvious case of Accuracy vs. F1-score would be with the simple classifier networks, which achieve 55% accuracy on the Stereo dataset but their F1-scores tend to zero, meaning that they simply learn to classify all the text samples as unbiased (Figure 16).

- BLOOM has the best F1-score on the Mixed dataset, which shows promise for small complicated datasets, especially considering that we were using a very reduced BLOOM model. Nevertheless, it is important to keep in mind that it is a very computationally expensive model and many of its strengths, such as understanding different languages or programming code, might not even be put to practice in a bias detection task for English documents. BERT seems to be suited for the task and it is computationally more efficient.

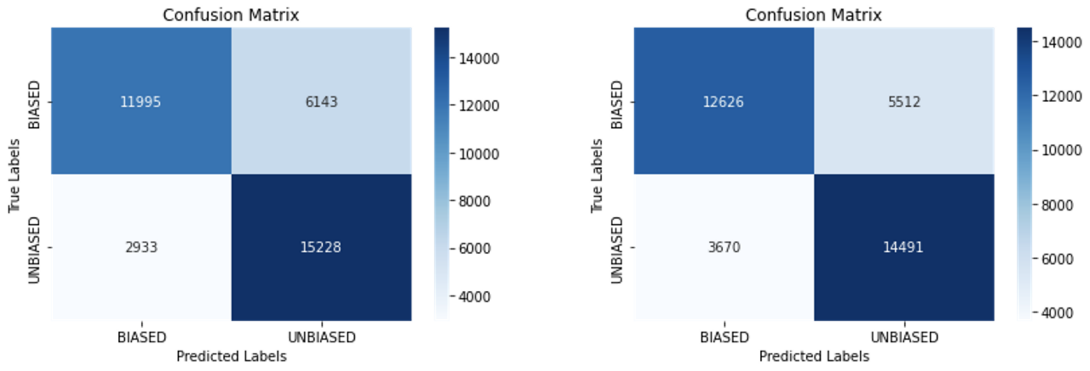



**Figure 15** ***Confusion Matrix of BERT and Mixed 2 models on NPOV dataset.*** *BERT model (left) correctly predicts around 600 less bias samples than Mixed 2 model (right).*

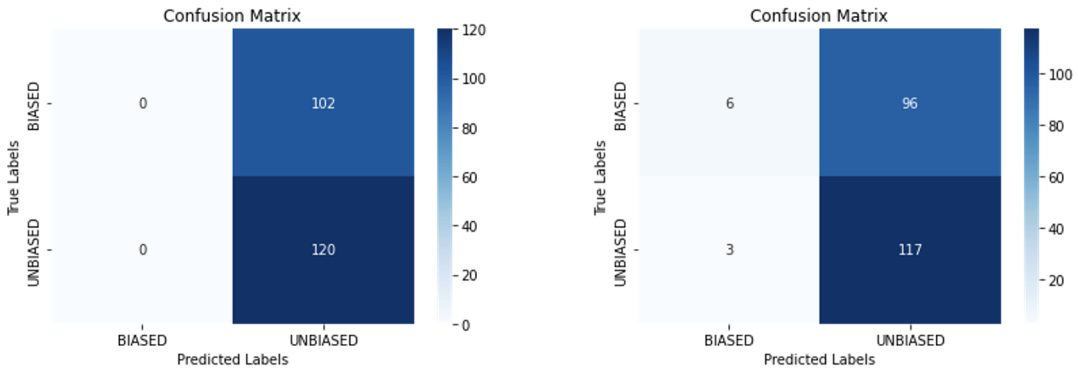



**Figure 16** ***Confusion Matrix with F1-score close to zero.*** *Featurizer model on Stereo dataset (left) and Featurizer+GloVe model on Stereo dataset (right).*

## 6.3 Results on unfamiliar test sets

We further explore the best performing models: BERT, BLOOM, and Mixed 2, and test them on datasets they have not been trained on, to see if there are any patterns or if there is a superior model or dataset for training that generalises the most. Tables 7, 8 and 9 summarize these resuts.

| | NPOV | WNC | CrowS-Pairs | Stereo | Mixed |
|---|---|---|---|---|---|
| **NPOV** | 75.0—72.6 | 85.2—92.0 | 29.6—45.6 | 57.2—48.6 | 62.2—55.4 |
| **Stereo** | 50.1—18.6 | 89.6—94.5 | 53.5—69.7 | 84.2—83.3 | 61.7—49.7 |
| **Mixed** | 56.2—34.8 | 90.2—94.9 | 92.7—96.2 | 73.4—77.1 | 65.4—61.9 |

**Table 7:** Summary of Accuracy—F1-score results in % for BERT model trained on a specific dataset (rows) and tested on a different dataset (columns).

| | NPOV | WNC | CrowS-Pairs | Stereo | Mixed |
|---|---|---|---|---|---|
| **NPOV** | 74.7—73.3 | 78.5—88.0 | 50.5—67.1 | 57.7—56.5 | 67.8—67.5 |
| **Stereo** | 50.0—20.0 | 87.4—93.3 | 53.2—69.4 | 81.1—80.6 | 60.6—49.5 |
| **Mixed** | 62.4—65.3 | 64.4—78.3 | 93.7—96.7 | 68.5—73.7 | 65.8—70.7 |

**Table 8:** Summary of Accuracy—F1-score results in % for Mixed 2 model trained on a specific dataset (rows) and tested on a different dataset (columns).

| | NPOV | Stereo | Mixed |
|---|---|---|---|
| **Mixed** | 60.4—67.0 | 75.7—78.2 | 66.6—73.0 |

**Table 9:** Summary of Accuracy—F1-score results in % for BLOOM model trained on the Mixed dataset and tested on a different dataset (columns).

The following can be observed:

- The WNC dataset results are a bit irrelevant because it only contains unbiased sentences. Therefore, it measures how effectively each model can predict unbiased sentences. To no surprise, all models can predict unbiased sentences fairly well trained on any dataset, with BERT having the edge. But in most cases it is an impractical task since the model could have learnt to just classify all texts as unbiased, which would be useless.

- BERT-based models trained on the mixed dataset perform extremely well in the CrowS-Paris dataset. This is because unlike NPOV and Stereo, the Mixed dataset contains a small proportion of samples from the CrowS-Pairs dataset, meaning that learning to detect the bias in CrowS-Pairs is fairly easy, even with little data.

- Stereo dataset is not good for training since it is small and contains simple types of bias, and models find it hard to generalise to other forms of biased data.

- The BERT model does not seem to generalise well for NPOV data, presenting very low F1-scores for NPOV test data when trained with the Mixed dataset. However, the benefits of using syntactic and semantic features and a BiLSTM with BERT are very apparent when we have little training data, with the F1-score almost doubling (green cells).

- BLOOM also manages to generalise well with the Mixed dataset, attesting to its potential even if just using a reduced version of the model.

It could be argued that overall the Mixed 2 model is the best model for its consistent results across datasets and its computational efficiency compared to larger models such as BLOOM.

## 6.4 Graphical networks results

In this section we have a look at the performance of graphical networks. We compare test metrics with the Stereo amd Mixed datasets, and reduced versions

of the NPOV dataset (for lack of computational resources). For example, "NPOV 5%" means a dataset made up of 5% of the total NPOV data samples, chosen at random.

| | **NPOV 1%** | **NPOV 3%** | **NPOV 5%** | **NPOV 7%** | **NPOV 9%** | **Stereo** | **Mixed** |
|---|---|---|---|---|---|---|---|
| **Accuracy** [%] | 50.8 | 51.9 | 52.5 | 54.1 | 53.5 | 72.3 | 58.2 |
| **F1-score** [%] | 56.2 | 43.7 | 34.26 | 43.8 | 42.7 | 70.6 | 57.9 |

**Table 10:** Summary of GCN results for different datasets. NPOV % represents a reduced dataset made up of a fraction of the NPOV dataset.

| | **NPOV 10%** | **NPOV 20%** | **Stereo** | **Mixed** |
|---|---|---|---|---|
| **Accuracy** [%] | 52.8 | 51.8 | 67 | 56.5 |

**Table 11:** Summary of InductGCN results for different datasets. NPOV % represents a reduced dataset made up of a fraction of the NPOV dataset.

The results for the GCN and InductGCN models are both underwhelming (Tables 10 and 11), obtaining substantially lower results than the pre-trained models analysed earlier on. As expected, InductGCN can be trained on more data than GCN because it constructs the graph network with only the training data, but it does not yield better results.

The GCN model could have some potential, as its accuracy seems to be increasing as the size of the NPOV dataset increases, so it could be the case that it would perform well with the full dataset. However, there are no signs indicating whether the F1-scores would increase and stabilise as well, and it is hard to make a fair assessment when it can only deal with so little data.

# 7 Conclusions & Future Work

This dissertation focused on exploring neural network approaches for bias text detection. In particular, we sought to compare the strengths and weaknesses of classical neural network classifiers, pre-trained transformer-based models, and graphical networks for the task of detecting different types of biased text. We showed that model performance depends on the dataset it is trained on, the size of the dataset and the complexity of the biases it contains. Moreover, we showed ways to expand the models by combining different architectures all the while attending to computational constraints.

In the introduction of this dissertation we presented three questions that we aimed to answer during this work. We now revisit these objectives with the conclusions drawn from our research and propose alternatives for future work.

1. *Whether classic neural network task-centred classifiers can have a decent performance at bias text detection as long as the text data is properly vectorized.*

   We found that using embeddings that take into account a word's context and syntactic and semantic information, such as part-of-speech tags, dependency tags, and their appearance in bias lexicons, prove more useful for bias text detection than one-hot word embeddings or pre-trained GloVe word embeddings. These models however, struggle to learn with small datasets, and they prove to be more useful as additional layers to other more powerful models.

2. *Whether pre-trained large language models can be fine-tuned with bias text data corpus and produce state-of-the-art results with little computational cost.*

   We found that pre-trained language models such as BERT or BLOOM have the best performances for bias text detection after being fine-tuned for the task in just a few epochs of training. The outputs of these models can be combined with syntactic and semantic features from a classic classifier and fed to a recurrent neural network like a BiLSTM to produce results that are more robust across different datasets and require less training data. From the pre-trained models that we experimented with, BERT seems more

appropriate for the bias detection task, since it provides acceptable accurary, F1-scores and generalisation without it being too computationally expensive. Nevertheless, we have not been able to test the full potential of BLOOM due to the model being too large for our computational resources.

3. *Whether graphical networks that are increasingly popular and effective in text-based tasks, are appropriate for bias text detection.*
   We found that despite being highly recommended for text-based tasks, transductive and inductive GCNs are hard to apply on biased data corpuses, since instances of bias are harder to identify than other classification problems and they require a large number of samples to learn properly. The underwhelming results reproduced in our research do not justify the computational memory needed to operate with graphs of such extents.

While the results found in this research are useful, there are several issues with the approach we followed that need to be emphasized:

1. The only large available dataset was the NPOV dataset from Wikipedia. While it contains plenty of text samples and complicated bias instances, the samples are from different domains, which might make it difficult for models to get used to and learn properly. It would be convenient to build large datasets for different domains, such as law, finance, art, etc., so that these models can be applied to more specific cases and see if performance improves. Moreover, this would give way to new algorithms to assess whether models can generalise to different domains with strategies like few-shot learning.

2. The BLOOM pre-trained language model is very computationally demanding and only a very reduced version with 350 times less parameters than the full version could be used. The reduced model already showed promise despite it already being more expensive than BERT. It would be useful to investigate whether the full BLOOM model yielded results that were worth its computational cost.

3. Graphical networks require too much computational memory to store the adjacency matrices of large corpuses. Alternatives such as working with

sparse matrices, or recognising which graph nodes and edges contribute the most to the task of bias detection, should be put in place in order to make GCNs efficient.

Suggestions for future work revolve around the following ideas:

1. Since pre-trained models obtain the best results, with BERT being the more practical, it would be interesting to generate ensemble models that combine BERT models, such as BERT, RoBERTa, or DistilBERT, each taking different types of representation of the text data useful for bias detection, and combine their embeddings to produce a more robust model with better performance.

2. Use multi-task learning approaches. For example, introduce the auxiliary task of classifying words individually as bias inducing or neutral, on top of the sentence classification target task.

3. Use graphical networks as an auxiliary mean to extract additional text embeddings, and use them alongside pre-trained models to improve model performance. This very recent paper [20] raises interesting points that are worth exploring with the models developed in this dissertation.

# Bibliography


[1] K. Pant, T. Dadu, and R. Mamidi, "Towards detection of subjective bias using contextualized word embeddings," in *Companion Proceedings of the Web Conference 2020*, pp. 75–76, 2020.

[2] E. M. Bender and B. Friedman, "Data statements for natural language processing: Toward mitigating system bias and enabling better science," *Transactions of the Association for Computational Linguistics*, vol. 6, pp. 587–604, 2018.

[3] M. Recasens, C. Danescu-Niculescu-Mizil, and D. Jurafsky, "Linguistic models for analyzing and detecting biased language," in *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1650–1659, 2013.

[4] R. Pryzant, R. D. Martinez, N. Dass, S. Kurohashi, D. Jurafsky, and D. Yang, "Automatically neutralizing subjective bias in text," in *Proceedings of the aaai conference on artificial intelligence*, vol. 34, pp. 480–489, 2020.

[5] S. Varsamopoulos, K. Bertels, C. G. Almudever, *et al.*, "Designing neural network based decoders for surface codes," *arXiv preprint arXiv:1811.12456*, 2018.

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017.

[7] R. Kulshrestha, "Transformers in nlp: A beginner friendly explanation," 2020.

[8] "Spacy Linguistic Features: Part-Of-Speech Tagging." https://spacy.io/usage/linguistic-featurespos-tagging. Accessed: 2022-08-30.

[9] "Holy NLP! Understanding Part of Speech Tags, Dependency Parsing, and Named Entity Recognition by peter baumgartner." https://pmbaumgartner.github.io/blog/holy-nlp/. Accessed: 2022-09-9.

[10] "DependencyParser: pipeline component for syntactic dependency parsing." https://spacy.io/api/dependencyparser. Accessed: 2022-08-30.

[11] A. Jagota, "Statistical language models," November 2020. [Online; posted 3-November-2020].

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

[13] N. Nangia, C. Vania, R. Bhalerao, and S. R. Bowman, "Crows-pairs: A challenge dataset for measuring social biases in masked language models," *arXiv preprint arXiv:2010.00133*, 2020.

[14] A. Alford, "Bigscience releases 176b parameter ai language model bloom," July 2022. [Online; posted 26-July-2022].

[15] "BLOOM by bigscience." https://huggingface.co/bigscience/bloom. Accessed: 2022-08-20.

[16] L. Yao, C. Mao, and Y. Luo, "Graph convolutional networks for text classification," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, pp. 7370–7377, 2019.

[17] T. S. Jepsen, "How to do deep learning on graphs with graph convolutional networks," 2020.

[18] K. Wang, S. C. Han, and J. Poon, "Induct-gcn: Inductive graph convolutional networks for text classification," *arXiv preprint arXiv:2206.00265*, 2022.

[19] R. Pujari, E. Oveson, P. Kulkarni, and E. Nouri, "Reinforcement guided multi-task learning framework for low-resource stereotype detection," *arXiv preprint arXiv:2203.14349*, 2022.

[20] B. Xue, C. Zhu, X. Wang, and W. Zhu, "The study on the text classification based on graph convolutional network and bilstm," in *Proceedings of the 8th International Conference on Computing and Artificial Intelligence*, pp. 323–331, 2022.