Automatic Detection of Linguistic Bias in Text

Annabel Kramer

4th Year Project Report
Cognitive Science (Informatics)
School of Informatics
University of Edinburgh

2018

Abstract

Biased language is language that aligns itself with one perspective over another. Some types of biased language are subtle and difficult to detect. Language that contains bias should not be present in reference works, like Wikipedia, and has been shown to affect readers' judgement and decisions. In this project we present a system for automatically detecting biased language in text. Using the Wikipedia Neutral Point of View (NPOV) Edits dataset and a set of pre-compiled lexicons, we train a baseline binary classification model to detect biased language at the word level. Test time accuracy replicates recent findings. We present an analysis of the feature space with relation to the bias detection task, providing evidence for the utility of several lexicons. We motivate the switch from linear models to neural networks, and augment the baseline with word embeddings and lexicon similarity features, gaining an improvement of up to 23.1% in word-based performance metrics.

Acknowledgements

I'd like to thank my supervisor Steve Renals for his gentle guidance, thoughtful comments, and constructive feedback throughout this project.

I would also like to thank friends and family who have provided numerous cups of tea and words of encouragement.

Table of Contents

Chapter 1

Introduction

Humans employ bias throughout their whole lives. It is an important tool we develop to be able to make faster decisions without considering each piece of information we have available to us; instead, we use knowledge from past experiences to make assumptions about the current decision to be made. This can result in making judgements and assessments about people and situations so quickly that we do not realise we have made one.

Our biases, both conscious and unconscious, manifest themselves in various ways; not least through the language we use. Language that expresses opinion, for example, or subscribes to one perspective over another, is considered biased. Linguistic bias is not a negative characteristic in many contexts. Existing research within computational linguistics has focused on detecting markers of linguistic bias such as subjective words, or other indicators of author's stance like sentiment.

More recent research aims to detect and deal with bias in models trained on text, for example through debiasing word embeddings, or identifying bias in political or news domains. In domains like news reporting or the writing of reference works (e.g. Wikipedia), authors have a duty to be objective, neutral, and unbiased. Some biased language, such as framing, is very subtle and hard to identify. An automatic system that highlights words or phrases that introduce bias could be very useful here. There is no well-established method for developing a fully automatic system to detect biased words or phrases; existing efforts have yet to achieve convincing results.

In this project, we build an automatic system for detecting biased language using the Wikipedia Neutral Point of View (NPOV) Edits dataset. Wikipedia is an online encyclopedia and reference work which is built in a 'wiki' format, meaning anyone with Internet access can edit the articles. Wikipedia articles are therefore in a state of continuous change. One of Wikipedia's fundamental writing policies is that of Neutral Point of View (NPOV), which states that all contributions to articles should be written from a neutral perspective, and not contain biased language. The Wikipedia NPOV Edits dataset gives pairs of sentences from Wikipedia articles before and after they were edited. The articles are taken from the website's NPOV Disputes page which lists articles that contain material violating the NPOV policy. Many edits in this dataset are

therefore made to remove biased language. These edits serve as labelled data for a bias detection classifier.

We address the problem of detecting a bias-inducing word in a given sentence. A model that does this would fit in with a larger system that, given a piece of text, identifies potential areas of linguistic bias, detects the words responsible, and corrects them. We pose the problem as a binary classification task, in line with [50], and fit a logistic regression to the most likely word in a sentence to be biased, using word-based feature vectors built from lexical features and pre-compiled lexicons.

We pose several research questions that we answer through experiments on the feature space and extensions to the baseline:

1. What is the link between lexicon coverage and baseline model performance?
2. Are stopwords helpful for the bias detection task or do they introduce unnecessary noise?
3. Is a 5-gram the best context window to use?
4. Do classifiers that capture non-linear, complex relationships (ie neural networks) perform better than linear models?
5. Are embeddings better representations of the word and lemma?
6. Can we achieve comparable or better performance using word embedding feature vectors?
7. Does modeling lexicon similarity instead of binary lexicon membership improve performance?

We achieve an increased performance after switching from a linear model to neural networks which can capture more complex relationships between input and output. We also obtain further improvements of up to 23.1% in word-level metrics over the baseline model when using word embeddings and lexicon similarity features.

1.1 Project Contributions

The contributions of this project include:

- Replication of current state of the art results (Chapter 3)
- Analysis of the feature space using lexicon coverage and ablation studies, revealing the extent to which some lexicons are more useful than others (Chapter 4)
- Novel application of neural networks to the bias detection task, achieving an improvement over performance using linear models (Chapter 5)
- Investigation of the usefulness of word embeddings in the feature vector (Chapter 5)

- Augmentation of the feature vector with lexicon similarity measures resulting in further performance increase (Chapter 5)
- Contribution to the discussion of detecting linguistic bias through identification of further areas of interest (Chapter 6)

Chapter 2

Background

The Internet allows for information to spread more rapidly than seen before. Anyone with a computer and Internet access can log on and create content. This content is not meticulously edited and peer-reviewed like books and newspapers and is thus more likely to contain traces of the author's cognitive biases.

Social media allows us to create huge networks through which information can travel extremely quickly. This makes the spread of misinformation - information which is false or incorrect - as easy as clicking a button to share an article. Research into the spread of misinformation online has suggested that selective exposure results in the forming of 'echo chambers' - homogeneous clusters of information from the same narrative [16], and that false news spreads to larger groups of people than true news [57].

The spread of misinformation has become a rising concern. In 2014 the World Economic Forum identified the "rapid spread of misinformation online" in the top ten trends facing the world[1]. This list also included conflict in the Middle East, persistent unemployment, and inaction around climate change. This year, the UK government set up a security unit specifically to tackle misinformation and fake news[2]. Companies are also taking measures to combat misinformation, with Facebook employing fact-checkers in Italy ahead of its election this year[3].

When a reader is presented with new information, their acceptance of it is influenced significantly by the alignment of the information with the reader's own beliefs - this is a type of cognitive bias known as confirmation bias [49]. Therefore, misinformation can easily be accepted if it complements the reader's current beliefs. This makes it crucial for misinformation to be detected and removed.

There are several areas we can focus on to mitigate the effect of misinformation on readers: raising awareness of and teaching methods for dealing with humans' cognitive

[1] https://www.weforum.org/agenda/2013/11/top-10-trends-facing-the-world-in-2014/)

[2] https://www.theguardian.com/politics/2018/jan/23/new-national-security-unit-will-tackle-spread-of-fake-news-in-uk

[3] https://www.washingtonpost.com/news/worldviews/wp/2018/02/02/facebook-goes-on-the-offensive-against-fake-news-for-italys-election/?utm_term=.a690ede63695

biases, for example, or developing methods for identifying biased information and correcting it. This project focuses on the latter task.

One method of analyzing information to assess bias is to directly analyse the words used in the text (as opposed to metadata about the topic, author, domain etc). Non-neutrality exhibited within language is known as linguistic bias, and is the type of bias that we focus on in this project.

2.1 Linguistic Bias

Linguistic bias has been well studied within social psychology and communications domains. It was only more recently formalised for study within computational linguistics. Precise definitions of linguistic bias vary:

- *A biased sentence reflects a tendency or preference towards a particular perspective, ideology or result.* - Yano et al (2010) [63]
- *A linguistic bias is defined as a systematic asymmetry in word choice that reflects the social-category cognitions that are applied to the described group or individual(s)* - Beukeboom (2014) [6]

We take the definition of linguistic bias to be language that subscribes to a particular viewpoint. Biased language could include language that discriminates based on race, sex, religion, ethnicity, or age. Language that attempts to persuade, argue, or otherwise present an opinion is also biased.

Note that in many situations, bias is not necessarily a negative characteristic. In court, lawyers choose their words carefully in order to convince the judge that their argument is better than the opposition. In politics, members of the government use specific words and phrases that appeal to the people they want to vote for them. *It is not always the case that linguistic bias should be removed.*

However, there are still situations in which biased language should not be used and it is these which will gain the most benefit from research into linguistic bias detection. News reporting strives to be an objective description of the facts and events surrounding the topic; tools that highlight potential bias in the writing could be useful in this context. Reference works, such as encyclopedias, should inform, not argue.

2.1.1 Types of Linguistic Bias

Language that exhibits linguistic bias can be divided into two types that are further explained below.

2.1.1.1 Epistemological

Epistemological bias refers to language that says something about the believability of a proposition. Avoiding this type of bias means avoiding presenting opinion as fact, and fact as opinion. In texts where the authors belief about the topic is not relevant, like a news article or encyclopedia entry, the language used to talk about the topic should not impose a judgement on the truth or validity of what is being discussed.

For example, a journalist would need to take care when using the words *claim* and *state*. To claim is to *state or assert that something is the case, typically without providing evidence or proof* [4], whereas to state (*express something definitely or clearly in speech or writing* [5]) contains no such connotation with lack of proof. It would therefore be preferable to use *state* in contexts where there is no obvious debate about the proof in order to avoid biasing the reader against the proposition being claimed, or insinuating that there is doubt surrounding it.

2.1.1.2 Framing

Framing refers to the choice of language used to present a topic - the words used to describe it, for example, and the people or events that it is linked with (Entman, 1993 [18]). This could be words that subscribe to a certain viewpoint, or words that emphasise the topic in a particularly positive or negative way. The term comes from the framing bias - a cognitive bias in which the way a decision is presented (e.g. as a loss or as a gain) affects the choice of the reader (e.g. Tversky and Kahneman, 1974 [56]; Plous, 1993 [49]).

Identifying how a topic is framed is crucial for understanding the topic (Rein and Shoen, 1996 [51]) and can help negate its effects, as suggested by Baumer et al (2017) [3]. However, the language of framing is subtle and difficult to detect - made even more so by a lack of studied methods for identifying it (Gamson and Modigliani, 1989 [20]; Chong and Druckman, 2007 [14]).

A good example comes from the moral debate surrounding abortion. People in favour of abortion tend to use the phrase *pro-choice* to emphasise the importance of the person deciding for themselves whether to undergo the procedure, whereas those against it often use the phrase *pro-life*, to emphasise their opinion that the procedure terminates a human life.

[4] https://en.oxforddictionaries.com/definition/claim
[5] https://en.oxforddictionaries.com/definition/state

2.2 Towards bias detection

2.2.1 Markers for Linguistic Bias

Certain subclasses of words have been identified as likely to show epistemological or framing bias. Some earlier research has focused on identifying classes of words that show opinion and sentiment as these are good indicators of potentially biased language.

Opinion mining is the task of extracting and quantifying the subjectivity [61] of a piece of natural language. It includes detecting whether a piece of text presents an opinion or not, using various natural language processing methods like sentiment analysis. Recognising whether a given sentence is objective or subjective is useful for bias detection, since subjective sentences are more likely to present an opinion and therefore contain bias.

Wiebe et al (2001) [60] extract n-gram collocations from Wall Street Journal opinion articles, comparing them to collocations in non-opinion articles to derive their subjectivity. Kim and Hovy (2006) [28] use FrameNet, a database of semantic frames and word meanings[6], to identify opinion-bearing words and the entity holding that opinion in sentences from online news articles.

Automatic phrase- and word-level sentiment analysis - deciding whether a piece of text exhibits positive or negative feeling - has been explored using lexicons (e.g. Wilson et al (2005) [62]) and unsupervised techniques that calculate the semantic orientation of a phrase with relation to words with known sentiment (e.g. Turney, 2002 [55]). Recent efforts in sentiment analysis have used deep learning methods such as convolutional neural networks (e.g. Dos Santos et al, 2014 [17]; Kim, 2014 [29]).

2.2.2 Political and News Bias

Other research has focused on bias in political or news domains.

Yano et al (2010) [63] investigate linguistic indicators of bias in political writing. They use Amazon Mechanical Turk to gather human annotations of biased words in sentences taken from American political blogs, showing that particular words or phrases are significant indicators of a liberal or conservative sentence. Lexical framing, sentiment, and subjectivity are identified as important factors for consideration in future work on bias detection. We use sentiment and subjectivity lexicons for the baseline model in this project.

There is a considerable amount of literature surrounding subjective measures for detecting news bias based on observation. These include examining the sources given, or identifying stereotypical presentation and loaded language. These guides place the burden on the reader to identify bias for themselves - what is judged as biased by the reader could be influenced by their own beliefs, and is therefore not an appropriate method for detecting news bias.

[6] https://framenet.icsi.berkeley.edu/fndrupal/

Work has been done on detecting and quantifying the degree to which a news article is biased using a lexicon of biased words and word similarity measures (Patankar et al, 2016 [48]). Results showed that Wikipedia articles were judged to be less biased than news articles, which were in turn less biased than opinion columns. This is a good step towards automatically detecting bias - but applies to a whole document. A more fine-grained detection, at the sentence and word level, is needed to be able to eliminate the linguistic bias from the text.

It is important to note that there is a difference between investigating bias shown by media outlets in terms of which stories they cover compared to the actual bias exhibited in the text that is written. Linguistic bias is more concerned with the bias exhibited by the words themselves, whereas much research into media and news bias investigates the political alignment of the organisation.

2.2.3 Word Embeddings

Identifying bias in language is part of a wider discussion within Artificial Intelligence surrounding how to deal with algorithmic bias (e.g. Hajian et al, 2016 [22]; Baeza-Yates, 2016 [1]). Researchers are becoming gradually more aware of examples of algorithmic bias in many areas; for example, research into facial recognition highlights the biases that exist within the dataset and algorithms used [10].

It is thought that some of the issues arising from algorithmic bias result from existing implicit biases within the data used (other sources of bias include the method of data collection or the amplification of bias through the model using the data). Therefore, some research into debiasing within computational linguistics has focused on identifying and removing bias in word embeddings. Word embeddings are a recent development in natural language processing which represent words as a dense continuous n-dimensional vector, with n typically ranging from 50-100 (e.g. GloVe [7]) to 300 (e.g. word2vec [8]). They have been shown to capture accurate semantic relationships between words; similar words appear close together in vector space, and we can use vector operations on embeddings to query the relationships by analogy. For example, *man - woman* is roughly equal to *king - queen* [44].

Bolukbasi et al (2016) [8] investigate bias within word embeddings, showing that word2vec embeddings trained on the Google News corpus exhibit gender bias. They present a method for de-biasing word embeddings by subtracting the geometric direction representing gender bias from the word vectors, preserving the appropriate relationships in gender neutral word pairs (e.g. *prince* and *princess*) but removing the bias from word pairs judged to contain stereotypical gender associations by human evaluators (e.g. *receptionist* is closer in vector space to *she* than *he* which reflects the stereotype that receptionists are more likely to be female).

This approach relies on *seed pairs*, the word pairs in which the gender relationship is not a stereotype but simply a reflection of lexical semantics, such as *mother - father*.

[7] https://nlp.stanford.edu/projects/glove/

[8] https://code.google.com/archive/p/word2vec/

These pairs are readily available in English for gender, but when we come to apply this method for de-biasing to other demographics, such as age or race, the seed pairs to choose in order to identify the dimension of bias are not obvious. This approach therefore seems quite convenient for examining gender bias, but fails once we wish to consider other dimensions of bias.

Zhao et al (2017) [65] also investigates bias in word embeddings on an image captioning task. They focus on gender bias in their research but note that the same method would be applicable to other categories like racial or ethnic. Their solution constrains the corpus so that gender indicators occur no more often together with elements of the prediction task than in the original training distributions.

Although word embeddings are trained on natural language text, the bias discussed in [8] and [65] arises from the models used to obtain the embeddings. The types of bias investigated, namely gender bias, are also much more explicit in text than the types of linguistic bias defined in Section 2.1; they relate to cultural stereotypes and the desire to remove them reflects a desire to promote equality between groups of people. For example, we want the word vector for *programmer* to be no more aligned with *man* than with *woman* because as a society we regard it as prejudiced to suggest a career is better aligned with one gender over another.

In contrast, the types of linguistic bias we investigate in this project are more subtle; that bias that is exhibited is not evaluated with relation to cultural views, but rather the implications that a biased word has for the propositions put forward in the sentence. We investigate word embeddings for feature extraction with relation to the task of detecting linguistic bias further in Section 5.2.

2.2.4 Going further

Previous work investigating the automatic detection of linguistic bias is fairly limited. Most research has focused on identifying cues to aspects of linguistic bias and very little original research has been found that presents an end-to-end solution to the problem of identifying linguistic bias.

Research discussed above has outlined efforts to detect and deal with limited types of bias in restricted domains: gender bias, political bias, newspaper articles. In order to have a more generalizable and therefore more useful model for detecting bias, we must use data from a much broader source, and focus on detecting more widespread types of linguistic bias. Section 3.2 outlines a method for doing this, which we use as our baseline model for this project.

2.3 Wikipedia

A reference work, such as an encyclopedia, can be referred to for information on a topic and is typically in the form of a book or electronic resource. They present a neutral narrative of a topic and should identify all sides of any disputes or controversies.

Wikipedia is an online reference work that divides its content into articles. Articles often link to other Wikipedia articles. It is built using the 'wiki' model - anyone can write and edit articles. Wikipedia relies on its diminishing number of volunteer editors to create new content, maintain current articles, and participate in discussions surrounding controversial content.

Since its launch in 2001, Wikipedia has risen to become a well-known online resource, with a Wikipedia result on page one for 99% of Google searches [9]. With Google's more than 40,000 searches per second[10], a Wikipedia article has the potential to be read by millions of people in a single day.

2.3.1 Neutrality

As the fifth most visited website in the world[11], it is imperative that Wikipedia's articles remain as unbiased as possible.

One of Wikipedia's main requirements for contribution is its Neutral Point of View (NPOV) policy. This policy states that authors should write articles that represent "fairly, proportionately, and, as far as possible, without editorial bias, all of the significant views that have been published by reliable sources on a topic [12].

Given the websites powerful reach and our knowledge of how linguistic bias can affect readers' judgements [56], a single biased sentence could have far-reaching and significant impact.

Wikipedia has also had its own share of vandalising and inauthentic attempts to edit articles. Some come from employees at firms that have a conflict of interest and want to sway the public opinion on a topic. A notable example of an article being edited by someone with a conflict of interest - the page for the medical procedure kyphoplasty, where the procedures effectiveness went from being described as *controversial* to *well-documented and studied* [13].

The wiki design of the website - the ability for anyone to sign up and edit articles - is both a benefit and, as the example above shows, a drawback for Wikipedia. With Wikipedia editors complaining that they are spending more and more time reversing these types of edits (see footnote 13), this further highlights the need for a system to flag up potentially biased or disingenuous edits. In this project we work on detecting linguistic bias in Wikipedia articles.

[9] https://econsultancy.com/blog/8987-does-google-give-too-much-prominence-to-wikipedia
[10] http://www.internetlivestats.com/google-search-statistics/
[11] http://www.alexa.com/siteinfo/wikipedia.org
[12] https://en.wikipedia.org/wiki/Wikipedia:Neutral_point_of_view
[13] https://www.theatlantic.com/business/archive/2015/08/wikipedia-editors-for-pay

Chapter 3

Wikipedia Edits Dataset

3.1 Dataset

The Wikipedia Neutral Point of View (NPOV) Corpus ([50]) is a dataset of all articles that were in the NPOV Disputes[1] page of Wikipedia as of 2013. The NPOV Disputes page of Wikipedia contains links to all articles that have been tagged `NPOV`. This tag is used by editors to indicate that an article violates Wikipedia's NPOV policy (Section 2.3.1); the article contains biased language and needs to be edited to remove the bias. The NPOV Corpus is in .xml format and contains the entire revision history of each article. A revision history is the sequence of edits to the text of the article that have been made over time, together with information about each edit such as author, size of edit, and edit comment.

In this project we use the NPOV Edits dataset from Recasens et al, 2013 [50]. This was built by taking the articles in the Wikipedia NPOV Corpus and extracting the pairs of sentences between revisions into rows containing edits made to the article (pairs of the original words and the modified words), with metadata about the editor, edit comment, the sentences before and after the edit. Editors make use of the NPOV tag to highlight an area of an article that violates Wikipedias NPOV policy. Therefore, edits whose comment mentions NPOV are assumed to be bias-removing edits.

The Wikipedia NPOV Corpus has also been used to automatically classify edits into categories such as spelling correction, paraphrasing, and NPOV (Daxenberger et al, 2013 [15]). Yatskar et al, 2010 [64] make use of the revision history to extract lexical simplifications (like *collaborate –> work together*). A corpus of corrections and paraphrases was mined also using the Wikipedia revision history by (Max and Wisniewski, 2010 [42]).

The Wikipedia Edits dataset comes in three partitions:

- Train: 1,613,126 edits (ie rows).
- Development: 146,641 edits.

[1] https://en.wikipedia.org/wiki/Category:All_NPOV_disputes

- Test: 180,235 edits.

Data is headed by ten columns:

1. Wikipedia article name
2. Revision number
3. NPOV tag: *True* if the revision text contains an NPOV tag [2]
4. NPOV comment: *True* if the comment associated with the edit contains the string "POV"
5. Editor ID
6. Revision size
7. *Before* string: string modified by the edit
8. *After* string: resulting string of the edit
9. Original sentence containing the *before* string
10. Modified sentence containing the *after* string

We give statistics about the full NPOV Edits dataset in Table 3.1.

	Articles	**Edits**	**Sentences**
Train	5976	1,612,818	1,097,697
Dev	653	146,616	101,515
Test	807	180,364	125512

Table 3.1: Numbers of articles, edits, and sentences in the full NPOV Edits dataset.

We extract a subset of bias-driven edits - indicated by a *True* in column 4 - of this dataset, and use this subset to select sentences for the training set. Details of this preprocessing are in Section 3.4.1.

3.2 An Automatic Bias Detection System

Recasens et al (2013) presents one of the first attempts at building a model for automatic bias detection. They frame the task of bias detection as a classification problem: given a word, categorise it as either biased (class 1) or unbiased (class 0).

The method presented in Recasens et al (2013) is a logistic regression on extracted features of a word. The probability output of the model corresponds to how confident the model is that a word is biased or unbiased. It is evaluated sentence by sentence, with the model accuracy calculated by dividing the number of sentences whose biased word was correctly predicted by the total number of sentences tested.

[2] https://en.wikipedia.org/wiki/Wikipedia:NPOV_dispute

Preprocessing of the Wikipedia NPOV Edits dataset (Section 3.1) is required to obtain a list of sentences that only had a one-word, bias-driven edit (ie an edit made to one word in order to remove linguistic bias). Features are extracted from each word in each selected sentence. Nine lexicons designed to capture classes of words thought to introduce bias are used for feature extraction. Additional features include lexical properties such as grammatical relation, part of speech, lemma. The model built is therefore a word-based model that identifies the single biased word in a given sentence that is already known to contain bias.

Amazon Mechanical Turk is used to gather human evaluations of biased words in the 230 test set sentences of the Wikipedia NPOV Edits corpus. Participants were asked to provide the one word that introduced bias into the sentence for 10 sentences, after being shown two examples of epistemological and framing bias (see Section 2.1 for an explanation of each). The task revealed the human annotators to correctly identify the biased word with an accuracy of 37.39%.

This approach uses a very straightforward model, and instead focuses on extracting the most useful features for the bias detection task. It assumes that the task of bias detection at the sentence-level has already been carried out.

3.3 Terminology

This chapter makes use of some terminology specific to the dataset and the task. The following definitions may be useful:

- *before* string: the sequence of words that was edited
- *after* string: the sequence of words that the before string was changed to
- edit: a row in the Wikipedia NPOV Edits dataset, containing a sentence from a Wikipedia article before and after it was edited, as well as the *before* and *after* strings, and other metadata such as editor and size of edit
- bias-driven edit: an edit whose comment includes the string *NPOV*

3.4 Preprocessing

Data is provided in the form of the Wikipedia NPOV Edits corpus. This is a dataset containing all the edits made in the history of a Wikipedia article. An article is not a static object, but a dynamically changing piece of text, characterised by a chronological sequence of edits.

Two preprocessing steps were implemented: a filtering of the NPOV Edits dataset to obtain suitable edits for the bias detection task; sentence selection from the suitable edits to be used for feature extraction.

3.4.1 Wikipedia Edits Filtering

Several filters needed to be applied to obtain the data that the logistic model will be trained on. Recasens et al report several filters used:

1. *before* string is 5 words or less
2. *after* string is 5 words or less
3. Levenshtein distance between the *before* and *after* strings is greater than or equal to 4
4. The edit did not only add a hyperlink to the text

We developed the script that implements this preprocessing step from scratch, using the following tools:

- `distance`: Python library for measuring Levenshtein distance
- `difflib`: Python library for comparing character differences between two strings

Considerable time was spent on initial preprocessing in order to match the statistics reported in Table 1 of Recasens et al to make our results as comparable as possible. The final list of filters used in the Edits filtering step is displayed in Table 3.2.

Filter	Explanation
Before string length	The *before* string contains a maximum of five words
After string length	*after* string contains a maximum of five words
Levenshtein distance	The Levenshtein distance between the *before* and *after* strings is greater than 4
Hyperlink	The edit added more information than just a hyperlink, denoted by [[..]]
Numerical / punctuation*	The edit did not only change numbers or punctuation marks
Stopwords*	The edit did not only change stopwords

Table 3.2: Filters used to extract suitable edits from the Wikipedia NPOV Edits dataset. *from M. Recasens, personal communication

3.4.2 Sentence Extraction

Once suitable edits were obtained, sentences were then selected from this dataset that have only one NPOV edit consisting of a single word in the *before* string. This is the subset of sentences in the dataset that have only one edit where the column *NPOV comment* is True (there could be other edits to this sentence but we are not concerned with them if their *NPOV comment* column is False) and that same edit contains only one word in its *before* string.

3.4.3 Validation

Table 1 of Recasens et al reports statistics of the NPOV Corpus and Edits datasets after their filtering and preprocessing. This table provided a good reference point for validating the choices we made in terms of how to preprocess the data. Considerable time was spent taking care to match these numbers as closely as possible so that further results we obtain throughout this project are comparable to this baseline and to the original paper. We report the statistics we achieved compared to the originals in Tables 3.3 and 3.4.

Method	**Number of Edits**		
	Train	**Dev**	**Test**
Recasens et al	13807	1261	1751
Split on whitespace	14024	1157	1606
NLTK Tokenize	7996	671	941

Table 3.3: Comparison of number of edits in NPOV Edits dataset marked suitable for use when using different tokenising methods.

Section 3.4.1 introduces some ambiguities in the filtering step as reported in Recasens et al. Since one of the filters uses the number of words in the *before* and *after* strings, we had to decide on a tokenising method. Tokenising is the process of splitting a sentence into tokens - usually corresponding to the words in the sentence. This seems a straightforward task in English; the following sentence *Joe caught the ball* contains the words *Joe, caught, the,* and *ball*, so it is sufficient to define the words as the string of characters with whitespace on either side.

Ambiguity arises when we have non-standard character sequences including full stops in the middle of sentences (which typically denote the end of a sentence), hyphens in a compound word, or apostrophes in contractions (e.g. *can't*). In each case, a decision must be made about where the word boundary lies. There are multiple methods for obtaining a list of tokens given a sentence, including:

1. `NLTK word_tokenize`
2. `Stanford CoreNLP`
3. `gensim tokenize`
4. Split on whitespace
5. Parse sentence and take the units that the parser produces as tokens

Recasens et al report that they used Stanford's `CoreNLP` for dependency parsing and POS tagging, so we considered it plausible that a similar NLP package had been used for tokenising. Table 3.3 reports the number of edits obtained after using different tokenising methods. Our experiments showed that splitting on whitespace and removing hyphens and other punctuation characters produced numbers closest to those reported in Recasens et al. This is surprising as it would have been more intuitive (and probably easier) to use a pre-written process for tokenisation.

We also match the number of sentences selected from the subset of edits almost exactly. These numbers are reported in Table 3.4.

	Number of Sentences		
	Train	**Dev**	**Test**
Recasens et al	1843	163	230
Current work	1826	163	231

Table 3.4: Comparison of number of sentences extracted from NPOV Edits dataset.

3.5 Feature Extraction

To create training data, features are extracted from every word in each sentence. The model relies heavily on this curated set of extracted features, which are outlined in this section.

The feature extraction script was written from scratch in Python 3.6, with help from the following tools and libraries:

- `nltk WordNet Lemmatizer`: to get lemma form of each word
- `Stanford DependencyParser`: a tool for calculating the dependency parse of a sentence, part of the StanfordCoreNLP toolkit. Also gives POS tags of the words.

Each word in each of the selected sentences constitutes a data point for training/testing. Size of training and test data is given below in Table 3.5.

	Number of Samples
Train	49448
Test	6511

Table 3.5: Number of training and test samples after feature extraction.

3.5.1 Part-of-Speech tags

Part-of-Speech (POS) tags are assigned to each word in a sentence, providing information about what role the word plays according to the sentence's constituency parse.

By far the most common POS tag in the training set is NN as shown in Figure 3.1. Similarly, the most common POS tag among biased words in the training set is NN.

3.5.2 Lexicons

Most of the features that we extract are based on pre-existing lexicons from previous research. Each lexicon was developed to capture prominent instances of particular

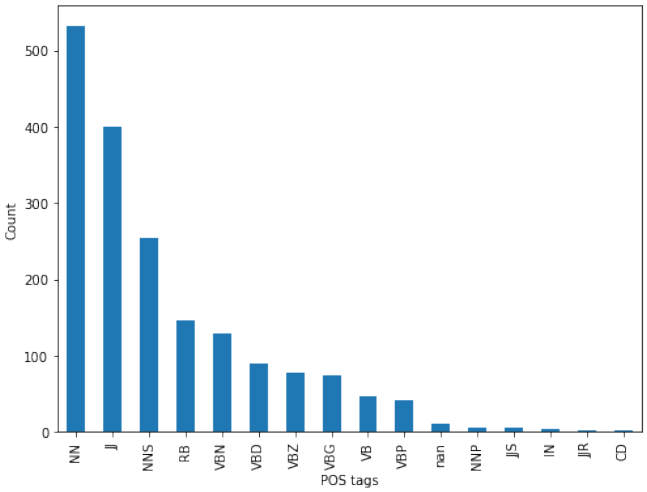


Figure 3.1: Distribution of POS tags among biased words in the training set.

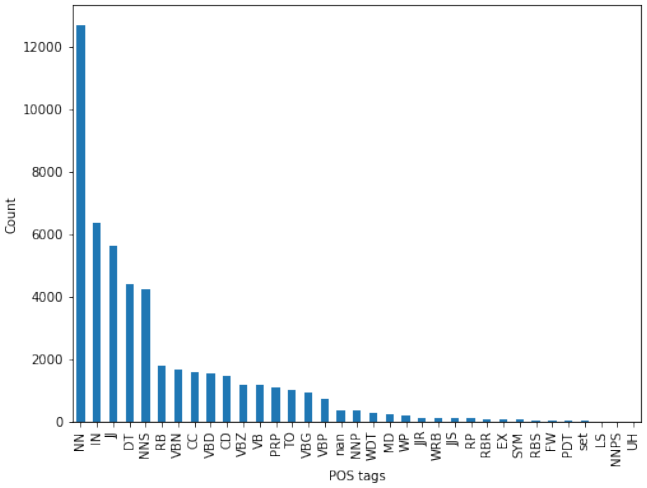


Figure 3.2: Distribution of POS tags in the training set.

groups of words, such as entailing words, or words that show sentiment.

This section presents the lexicons used and reviews their relevance to the bias detection task.

3.5.2.1 Sentiment

The sentiment lexicon was taken from Liu et al (2005) [40]. Sentiment refers to the polarity in feeling of a word or phrase - course-grained sentiment analysis classifies text into positive or negative, while more fine-grained analyses use more categories. Sentences which are strongly positive or negative may show some bias since sentiment often comes from personal experience or opinion. Words which carry particularly positive or negative connotations may also introduce bias to the sentence e.g. if an author is expressing how much they enjoy / dislike something, then that sentence contains

bias (and certainly has no place in a reference work).

3.5.2.2 Subjectivity

Lexicon from [60] The lexicon aims to capture words which indicate use of opinions and includes words such as *divisive, embarrass,* and *heartily*. Section 2.1 explains that opinions are biased because they are aligned with one perspective over another (remember, this is not an undesirable characteristic in some contexts).

3.5.2.3 Hedges

The lexicon of hedges was taken from Hyland (2005) [24]. Hedges are a class of words that deliberately prevent the author from committing to a stance on a topic, thus signaling their opinion instead of fact. Typical hedge words include *suggest*, *perhaps*, *might*. They are often used to present plausible explanations instead of stating as fact, for example, why an experiment achieved unexpected results. An author may employ the use of hedges if their personal beliefs or cognitive biases conflict with a statement that indicates otherwise. They may be reluctant to present the truth, and so use a hedge word to portray the possibility that the statement is false (or true, depending on the context).

3.5.2.4 Entailment

The entailment lexicon is from Berant et al (2012) [5]. Words in this lexicon entail a further meaning to the one that they contain. For example, *murder* entails *kill* because you cannot murder someone without killing them. Using an entailment word without carefully considering what is entailed by it could easily lead to a biased sentence if the author has misunderstood what they are writing about; they may imply something that is not true or did not happen.

3.5.2.5 Factives

The factives lexicon is taken from Hooper (1975) [23]. Factive verbs necessitate that their complement clause is true, as opposed to presenting the clause as a stance or opinion. *realise* is a good example of a factive verb: *he realised that eating animals was wrong* presents *eating animals was wrong* as an objective truth, when really the statement reflects the author's stance on eating animals.

3.5.2.6 Assertives

The lexicon of assertives is also from Hooper (1975) [23]. Assertives are similar to factives in that their complement clauses assert a proposition, but this proposition's

certainty depends on the asserting verb. *claim*, for example, calls into question the truth of the proposition, and implies a lack of evidence or proof. *point out*, on the other hand, signals that its complement clause should be taken as fact. Both of these verbs appear in the assertives lexicon.

3.5.2.7 Implicatives

The Implicatives lexicon comes from Karttunen (1971) [27]. Similar to assertives, implicatives are words that imply the truth or falsity of the complement or infinitive clauses that follow them.

1. *Mary managed to open the bottle.*

For example, stating the sentence above entails that Mary opened the bottle. Here, the implicative is *managed*, and indicates that its infinitive complement *to open the bottle* is true. Implicatives can also imply that their complement is false.

2. *Mary failed to open the bottle.*

Stating the above entails that Mary did not open the bottle. The implicative in this sentence is *failed* - it implies that there was an event in which Mary did not open the bottle.

Contrast this with what Karttunen refers to as a 'non-implicative':

3. *Mary hoped to open the bottle.*

Our understanding of the event is now different - there was no definitive event in which a bottle was or was not opened, as there would have been when an implicative is used. Non-implicatives do not assert any truth values over their complements.

It is straightforward to see the significance of this group of words in the context of bias detection. An author's beliefs may lead them to insert an implicative before a complement whose truth value is not widely accepted (a controversial claim, for example). Presenting the argument of the implicative as true or false is a form of epistemological bias that we want to identify (and then correct).

3.5.2.8 Report Verbs

Report verbs, from Recasens et al (2013) [50], are words for presenting what others have said. Words include *reveal, signal, speculate, state, suggest, think, underline*.

3.5.2.9 Bias

This lexicon is from Recasens et al (2013) [50] and was directly extracted from the Wikipedia NPOV Edits dataset. It is made up of words from the dataset that were NPOV-edited at least twice and appear in at least two different articles. This lexicon

has also been used in other tasks, for example to quantify the degree of bias in news articles [48].

3.5.2.10 Lexicon Summary

In total, nine lexicons are used of varying sizes and from both recent and older research. Word counts of these lexicons are given in Figure 3.3. Given the varying size of these lexicons, some may be more useful in the bias detection task than others due to a higher coverage of words. We investigate lexicon coverage in Section 4.1.

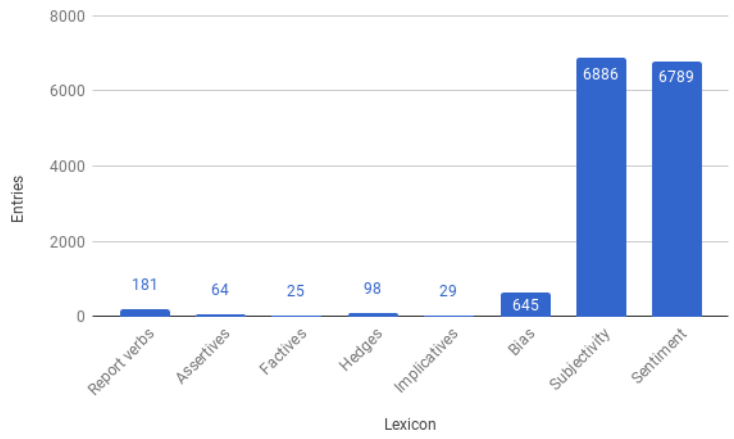


Figure 3.3: Number of words in each lexicon. The *Entailment* lexicon is not shown because its number of entries warped the scale too much: 52163433.

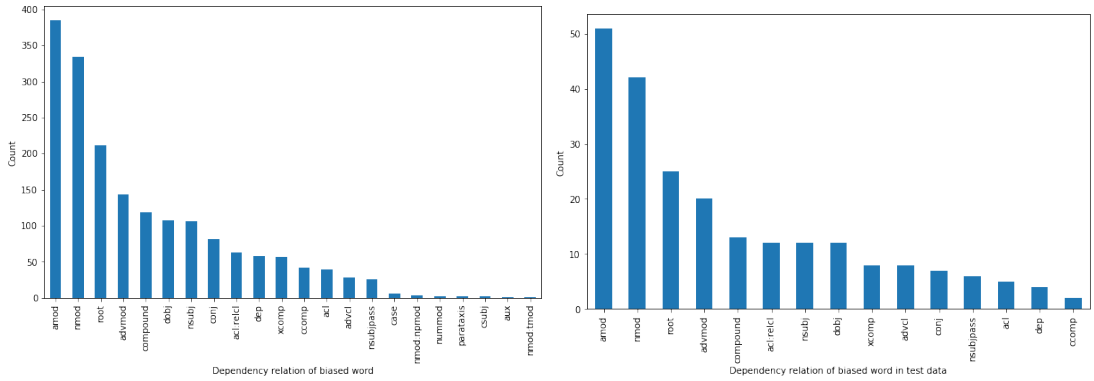
3.5.3 Dependency relation

The dependency relation of the word in question is also included in the feature vector. The dependency relation is determined through a dependency parse of the whole sentence. Given a sentence, a dependency parser returns relations between each word and the root word in the sentence (typically the finite verb). This feature contrasts with the POS tags, which provide information about the constituency parse of the sentence.

We calculate the dependency parse of a sentence using the `DependencyParser` in the Stanford `CoreNLP` toolkit.

Figure 3.4 shows the distribution of dependency parse relations in the training and test sets. The most frequent relations are:

- `amod`: adjectival modifiers e.g. *green* in *she painted green pictures*
- `nmod`: nominal modifier e.g. *children* in *give the toys to the children*
- `root`: typically the main inflected verb in the sentence



(a) Training set. (b) Test set.

Figure 3.4: Distribution of the dependency relations of biased words in the training (*left*) and test (*right*) sets.

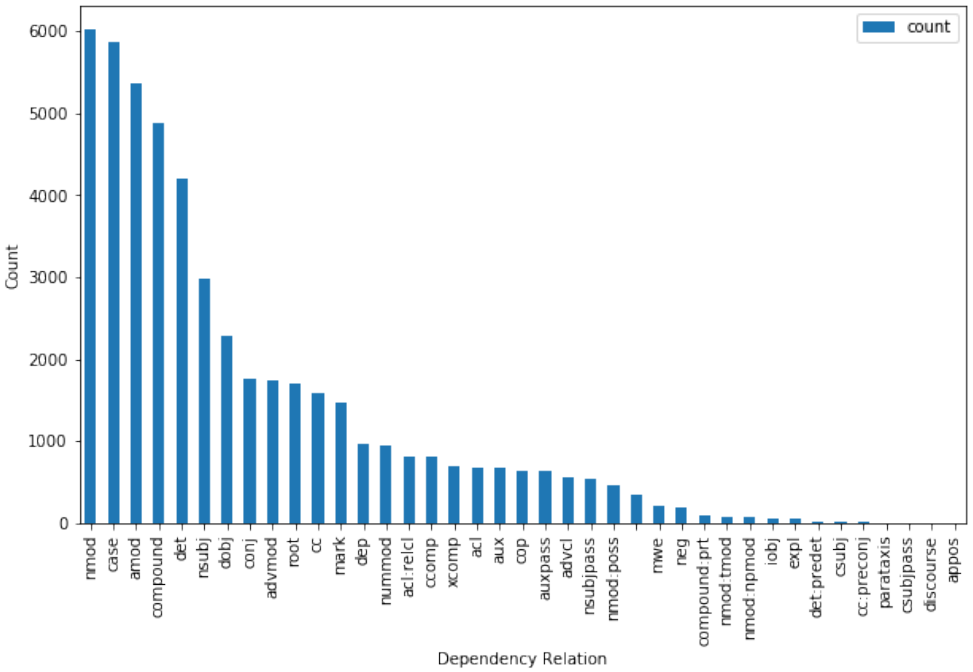


Figure 3.5: Distribution of dependency relations in the training set.

This distribution differs from the distribution in the whole training set (Figure 3.5) where `case`, `compound`, and `det` relations feature much more frequently. These relations refer to the following types of words:

- `case`: case markers which are separate words, including postpositions and prepositions e.g. *of, after*
- `compound`: a word that is part of a compound e.g. *phone* in *phone book*
- `det`: determiner words e.g. *the, a*

These distributions tell us that the dependency relations of biased words are still among

the most common dependency relations in the training set - but tend to describe content words like adjectives adjectives). The relations listed above tend to be describing function words which carry little meaning, whereas the most common dependency relations of biased words refer to content words. In fact, there are no `det` relations at all in Figures 5.3a or 5.3b. This demonstrates the potential usefulness of this feature; an accurate model could learn that biased words tend to have content-describing dependency relations.

3.5.4 Collaborative Feature

The final feature captures article-specific information. It is calculated for each training sample by dividing the number of times the word was edited to remove bias by its occurrence in the dataset.

3.5.5 Feature Summary

To summarise, we filter the NPOV Edits dataset down using filters reported in Recasens et al and select sentences from this subset that have a single-word, bias driven edit. Each word in each of these sentences constitutes a single training sample. For each word in each sentence, a feature vector is constructed from lexicon features (e.g. POS tags), binary lexicon membership features, and context features (POS tags of words in context, and whether any words in the context are found in the lexicons). The training targets are 1 if the word is the one that was edited, and 0 otherwise.

Table 3.6 gives a concise summary of the features used in the baseline model and their values.

3.6 Label encoding

Label encoding was required to turn the categorical fields of the feature vector into binary features. The fields that this was applied to are:

- Words
- Lemmas
- Dependency relation
- POS tag
- Position of word in sentence
- Polarity of word (from sentiment lexicon)

Feature	Value	Resource
Word	string	
Lemma	string	NLTK Lemmatizer
POS of word	VB, NN, etc	Stanford DependencyParser
POS of word - 1	VB, NN, etc	Stanford DependencyParser
POS of word - 2	VB, NN, etc	Stanford DependencyParser
POS of word + 1	VB, NN, etc	Stanford DependencyParser
POS of word + 2	VB, NN, etc	Stanford DependencyParser
Dependency relation	subj, amod, etc	Stanford DependencyParser
Report verb	1, 0	Report lexicon
Report verb in context	1, 0	Report lexicon
Assertive	1, 0	Assertive lexicon
Assertive in context	1, 0	Assertive lexicon
Bias	1, 0	Bias lexicon
Entailment	1, 0	Entailment lexicon
Entailment in context	1, 0	Entailment lexicon
Strong subjective	1, 0	Subjectivity lexicon
Strong subjective in context	1, 0	Subjectivity lexicon
Weak subjective	1, 0	Subjectivity lexicon
Weak subjective in context	1, 0	Subjectivity lexicon
Polarity	1, 0	Subjectivity lexicon
Position in sentence	start, middle, end	
Positive	1, 0	Sentiment lexicon
Positive in context	1, 0	Sentiment lexicon
Negative	1, 0	Sentiment lexicon
Negative in context	1, 0	Sentiment lexicon
Factive	1, 0	Factive lexicon
Factive in context	1, 0	Factive lexicon
Implicative	1, 0	Implicative lexicon
Implicative in context	1, 0	Implicative lexicon
Hedge	1, 0	Hedge lexicon
Hedge in context	1, 0	Hedge lexicon
Collaborative feature	float	Manually extracted

Table 3.6: Features used in baseline model. *in context* features are 1 if any of the context words (ie the 5-gram window around the current word) are in the lexicon.

The `pandas get_dummies`[3] function was used for this. Given a table (e.g. `pandas DataFrame`) and column name to encode, this function adds a new column to the table for every unique value in the column to be encoded, putting a 1 in the relevant cell if the original value corresponds to the value that the new column represents.

Encoding in this one-hot manner increases the dimensionality of the feature space by a large factor. From the original 32-dimensional feature vector, one-hot encoding of the words, lemmas, dependency relations, POS tags, position of word in sentence, and

[3]https://pandas.pydata.org/pandas-docs/stable/generated/pandas.get_dummies.html

polarity produces a feature vector with length 23,043.

3.7 Baseline Model

Recasens et al uses the Wikipedia NPOV Edits dataset as labelled data to fit a logistic regression model. To evaluate the model's performance, they calculate three accuracy scores: top 1, top 2, and top 3. These are sentence-level accuracies, calculated by by predicting the probability of each word in the sentence to be biased, and considering the sentence correct if the target biased word is in the top 1, 2, or 3 most likely words to be biased.

3.7.1 Logistic Regression

We use a binary logistic regression model to classify words into biased (class 1) or unbiased (class 0).

A logistic regression takes as input an n-dimensional feature vector **f** = f_1, f_2, ... , f_n and outputs 1 or 0.

The logistic regression maps variable x onto probability p using the equation:

$$p = \frac{1}{1+e^{-x}} \tag{3.1}$$

where x is a weighted linear combination of the input features f_1, ... , f_n plus bias term w_0:

$$x = w_0 + w_1 f_1 + ... + w_n f_n \tag{3.2}$$

Given target values for each training vector, the model learns the weights w_0, w_1, ... , w_n by minimising the error between the model's prediction and the given target using gradient descent.

3.7.2 Scoring

At test time, we extract a feature vector for each word in each sentence in the same manner as the training data. Results are calculated on a sentence by sentence basis; accuracy of the model is equal to the number of sentences with the correctly predicted biased word divided by the total number of sentences:

$$accuracy = \frac{n_c}{N} \tag{3.3}$$

where n_c is the number of sentences whose target biased word was correctly identified and N is the total number of sentences.

Typically in a logistic regression a test data point would be classified by comparing the output to a pre-determined threshold (in most cases 0.5). If the probability is higher than the threshold, then the data point is classified as class 1, else it is class 0.

However, for this model we directly consider the probability estimates of each word in the sentence, sorting by the most likely to be biased (ie highest probability for class 1). We consider the top n words most likely to be biased in the sentence, and report accuracy for values of n = 1, 2, and 3 in order to compare implementations. A sentence was considered correctly predicted if the word with target 1 was in the top n words sorted according to their probability to be biased as output by the model. When considering the top 2, an accuracy of 45% would mean that for 45% of the sentences, either the most likely or the second most likely biased word as predicted by the model matched the target biased word for that sentence.

3.7.2.1 Evaluation Alternatives

We report model accuracy on the test set in order to compare our results to Recasens et al. However, accuracy is calculated sentence by sentence and is therefore not a word-based metric. Since our training samples represent words, not sentences, there seems to be a mismatch. Therefore accuracy may not be the most appropriate metric to tell us about model performance, especially if we are interested in how the model classifies words.

Accuracy is also not the most appropriate metric when dealing with highly imbalanced data as we have here (see Table 6.1). Therefore, we propose to use alternative metrics of model performance.

Precision and recall are often used in classification tasks to evaluate model performance, especially in cases of class imbalance like we have here. Precision measures the proportion of samples predicted to be in the positive class that are actually in the positive class. Recall measures the proportion of positive data points in the dataset that are correctly classified as positive by the model. For our task, that means the proportion of biased words that were correctly identified. The equations are as follows:

$$precision = \frac{T_p}{T_p + F_p} \tag{3.4}$$

$$recall = \frac{T_p}{T_p + F_n} \tag{3.5}$$

$$f1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{3.6}$$

where:

- T_p: the number of true positives - biased words that were correctly predicted to be biased
- F_p: the number of false positives - unbiased words that were incorrectly predicted to be biased
- F_n: the number of false negatives - biased words that were incorrectly predicted to be unbiased

$f1$ combines *precision* and *recall* into one metric. An $f1$ value of 1 indicates a "perfect" model while 0 indicates a very badly performing model. There is generally a trade-off between *precision* and *recall* - increasing one tends to decrease the other. The optimal values of each are task-specific - for some problems, a higher precision will be critical, while in others the highest recall possible is most important.

In the bias detection task, f1, precision, and recall are word-level metrics, whereas accuracy is sentence-level. An increase in one may not be reflected in the other, and this has implications for the conclusions that we make. If accuracy increases, we can only conclude that the model performs better at identifying biased words in sentences when the words are compared to *the other words in the same sentence*. However, f1 is a word-level metric that does not rely on classification in the context of a sentence, and therefore better reflects the classification power of a model on words alone.

3.7.3 Model Alternatives

Recasens et al posed the problem of bias detection as a binary classification task at the word level: given a biased sentence, predict the most likely word in the sentence to the biased. We classify at the word level and therefore are only capable of identifying words that are biased. This makes the use of lexicons well-suited to this task.

However, this approach has a number of drawbacks:

- It is limited to detecting only the most biased word. This ignores linguistic bias that is introduced by phrases; when preparing our training data in line with the approach in Recasens et al, we selected only edits whose *before* string was one word. This discarded a large number (6473/17808 in training set) of edits whose *before* strings were made up of phrases e.g.
- By only considering words in the local sentence, we ignore any information from longer-distance contexts such as the paragraph or indeed the whole document. It's possible that a sentence or phrase by itself is neutral but its interaction with information presented in previous or succeeding sentences introduces some type of bias into the text.

We should consider methods to identify biased phrases instead of just biased words. We could borrow techniques from language processing: text is represented as a sequence of words, and sequential models are used.

We could reformulate the bias detection task as a tagging problem: given a sequence of words, tag each word with either *bias* or *no-bias*. Sequential models have been used

for several tagging and sequence labeling tasks:

- POS tagging with Hidden Markov Models (HMMs) is well-established (e.g. Kupiec, 1992 [33]; Schutze, 1994 [52]; Brants, 2000 [11])
- Named Entity Recognition (NER) using HMMs (e.g. Zhou et al, 2002 [67]; Morwal et al, 2012 [45])
- Recurrent Neural Networks (RNNs) are a family of neural network architectures that explicitly process sequences of data. Introduced by , they have been used for numerous tasks that generate sequences and have been successful in advancing the state of the art in many NLP tasks [26]:
  - POS tagging (e.g. Wang et al, 2015 [58])
  - NER (e.g. Lample et al, 2016 [35])
  - Sequence to Sequence tasks like Neural Machine Translation (e.g. Sutskever et al, 2014 [54])

A sequential model would likely require labelled training data in the form of sentences and their tag sequences but this would be straightforward to using the NPOV Edits corpus - we have the sentence before it has been edited, and the edit string, so all words in the edit string would get a *bias* tag, and remaining words in the sentence would be tagged *no-bias*. The main strength of this sequential approach would be able to identify a biased sequence of words as opposed to a single word. We would not be restricted to labeling sentences, but could also label paragraphs and even documents.

3.8 Results

3.8.1 Baselines

We implemented two baselines according to the seed paper. The sections below present the results.

3.8.1.1 Sentiment

The sentiment baseline used only features from the sentiment lexicon. These were based on the lexicon developed by Liu et al (2005) [40]. The feature vector has four columns with values as follows:

- Negative: 1 or 0, whether a word has a negative connotation
- Positive: 1 or 0, whether a word has a positive connotation
- Negative context: 1 or 0, whether at least one of the two words before and after the current word has a negative connotation

- Positive context: 1 or 0, whether at least one of the two words before and after the current word has a positive connotation

Table 3.7 shows the performance implemented baseline compared with the baseline from Recasens et al (2013).

	Recasens et al, 2013	**Dissertation**
Top 1	14.78	12.12
Top 2	22.61	20.78
Top 3	27.83	26.84

Table 3.7: Comparison of sentiment baseline model accuracy on the test set when considering the top 3, 2, and 1 most likely words to be biased.

3.8.1.2 Subjectivity

The subjectivity baseline uses the subjectivity lexicon from [60].

The feature vector has nine columns, as follows:

- Strong: 1 or 0, whether a word is a strong subjective
- Weak: 1 or 0, whether a word is a weak subjective
- Strong context: 1 or 0, whether at least one of the two words before and after the current word has a negative connotation
- Weak context: 1 or 0, whether at least one of the two words before and after the current word has a positive connotation
- Polarity: One of *positive, negative, both, neutral, none*. This indicates the sentiment of the word. These five values were converted into a one-hot encoding to produce a final feature vector of length nine.

Table 3.8 presents the accuracy of our Subjectivity model against that reported in Recasens et al. When considering the top 1, 2, and 3 most likely words to be biased in each test sentence, our model performs slightly worse in all cases compared to accuracies reported in Recasens et al.

	Recasens et al, 2013	**Dissertation**
Top 1	16.52	14.72
Top 2	25.22	21.65
Top 3	33.91	29.71

Table 3.8: Comparison of subjectivity baseline model accuracy on the test set when considering the top 3, 2, and 1 most likely words to be biased.

3.8.2 Full Feature Set

Table 3.9 shows the results obtained after testing our model trained on the full feature set (all features described in Section 3.5). Brackets show the difference over the reported results in Recasens et al (2013).

	Recasens et al, 2013	**Dissertation**
Top 1	34.35	29.57 (-4.78)
Top 2	46.52	43.88 (-2.64)
Top 3	58.7	53.81 (-4.89)

Table 3.9: Comparison of our model accuracy (%) on the test set to Recasens et al when considering the top 3, 2, and 1 most likely words to be biased. Difference from Recasens et al is given in brackets.

We also report in Table 3.10 precision, recall, and f1 score when considering the most likely word to be biased, and compare our results to Kuang and Davidson (2016) [32] who also implement a bias detection model according to Recasens et al.

	Dissertation	**Kuang and Davidson**
f1	0.262	0.236
precision	0.219	0.245
recall	0.325	0.228

Table 3.10: Comparison of our model f1, precision, and accuracy on the test set to Kuang and Davidson (2016).

To calculate accuracy, we sorted all the words in a given test sentence by their probability of being biased, and for the top 1, 2, and 3 scoring methods we consider the biased word in the sentence to have been correctly predicted if the biased word appears in the top 1, 2, or 3 (respectively) words.

To calculate precision and recall we need class labels instead of probabilities for each test data point. Therefore, we need to choose a threshold such that if a test point's predicted probability to be biased is above that threshold, we assign it to class 1, else class 0. Kuang and Davidson perform a search over values from 0 to 1 in increments of 0.01 and choose the threshold that yields the highest f1 score on the training set.

We choose the threshold for classification using the same method. Figure 3.6 shows the f1 score for varying threshold values from 0 to 0.7. 0.14 was shown to give the best f1 score (0.262).

3.8.3 Discussion of Results

It is disappointing that in all baselines (sentiment, subjectivity, and full feature set) the accuracies obtained by our baseline model did not exactly match the reported accuracies by Recasens et al.

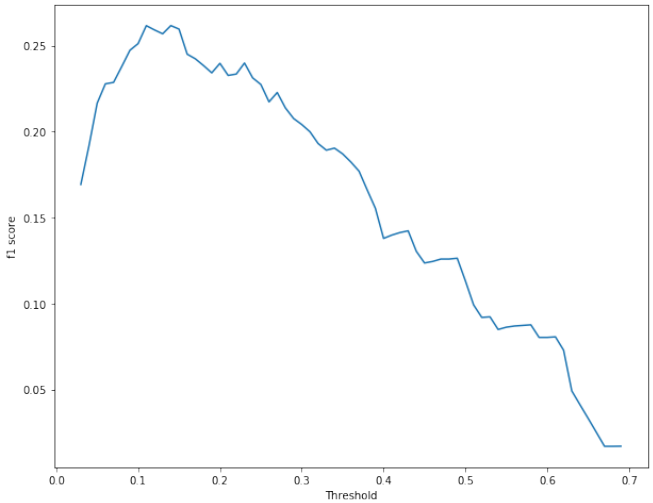


Figure 3.6: f1 score for varying threshold values.

One possible explanation for this is down to differences in the filtering and preprocessing methods; though we obtained comparable Edit and Sentence counts, this does not guarantee that the data itself is the same (although it's the best indication we could have used). Thus, the data that the model was trained on could have differed from that used in Recasens et al. It's plausible that these differences in the training and test sets could have caused the small difference between model accuracies.

We also reported f1, precision, and recall on the test set and compared this with results obtained by Kuang and Davidson's implementation [32] (See Table 3.10). Our f1 score was higher overall, but precision lower and recall much higher than they reported.

Perhaps these differences were caused by differences in choosing the threshold, or differences in training and test data. The latter seems likely as they note that they implemented additional filters to get closer to the dataset statistics reported in Recasens et al, and report smaller statistics that we obtain. If they trained on less data, perhaps a lower f1 score isn't so surprising.

Chapter 4

Feature Analysis

4.1 Lexicon Coverage

This section presents findings about lexicon coverage which ultimately lead to a conclusion about how useful the lexicons are in the bias classification task. We investigate lexicon coverage after noting the widely varying sizes of the lexicons (see Section 3.5.2.10 and Figure 3.3), hypothesising that some lexicons may be more useful than others due to capturing a larger number of words.

There are nine lexicons used in the classification task. We calculated the number of words in the training set that appeared in at least one of the nine lexicons. Table 4.1 shows our findings.

Table 4.1: Word counts in the training set for the biased and unbiased classes. A value of *True* in *No Coverage* means that the word did not appear in any of the nine lexicons.

Class	**No Coverage**	**Count**	**Percentage**	*Total*
Unbiased	True	32809	68.9	
	False	14813	31.1	47622
Biased	True	963	52.74	
	False	863	47.26	1826
Total		49448		49448

Table 4.1 shows that the majority of both biased and unbiased words do not appear in any lexicon at all, making up 68.3% of the total training data. This has serious implications; these lexicons are the best resource the model has for figuring out whether a word is biased or not. If most of the training examples do not appear in any of the lexicons, perhaps it is not surprising that we achieve an accuracy of only 29.57%.

4.1.1 Factives

The Factives lexicon only identifies 45 words in the training set, the lowest identification rate out of all the lexicons. Of these 45, 0 are biased. This calls into question the effectiveness of this lexicon in the feature set; what useful information is this lexicon adding if it doesn't even identify a single biased word in our training set?

Despite poor identification during training, we thought perhaps the lexicon is more useful and recognizes more words at test time. Unfortunately, this is not the case as there are only four words (all unbiased) in the test set that appear in the lexicon.

The factive lexicon was included to capture words exhibiting epistemological bias (Section 2.1.1.1): words which assume that their complement clause is true. For example, *she realized that the Earth was not flat* presents the Earth not being flat as a true fact. Using factives can introduce bias when the complement clause of the factive is something controversial or not widely accepted.

No biased words are identified by the factives lexicon. This means every word that is identified by the factives lexicon is unbiased. Given that the lexicon was meant to identify biased words, this could skew the model to learn that a word appearing in the factives lexicon is always unbiased.

Given this low identification rate, would the baseline model perform better if we removed the relevant features (namely whether a word is in the lexicon, and whether a word's context contains any words in the lexicon)? Table 4.2 presents a comparison with the baseline model.

	Baseline	**Factive lexicon removed**
Top 1 Accuracy	29.57	29.48 (-0.09)
Top 2 Accuracy	43.88	41.99 (-1.89)
Top 3 Accuracy	53.81	50.22 (-3.59)
f1	0.262	0.262
Precision	0.219	0.219
Recall	0.325	0.325

Table 4.2: Comparison of model accuracy (%) on the test set to baseline after dropping the factive lexicon.

Removing the factive and factive-in-context features seems to harm model accuracy but has no effect on f1, precision, or recall. These lexicons were included to help identify biased words, so decreasing accuracy when removing a lexicon that only identifies unbiased words seems unintuitive at first. However, no change in f1 score means we can conclude it has made no difference to classification at the word level - it is only when we identify the most biased word in the sentence that the factive lexicon improves accuracy. We conclude it is the combination of lexicons that helps to identify biased words in sentences, so the responsibility is not on only one lexicon.

4.1.2 Out of Lexicon Words

The lexicons are the main clues the model has for identifying whether a word is biased or not. We look at the *biased* words which do not appear in any lexicon to identify whether there are patterns that link them together, or whether a further lexicon could help capture some of them. This list of words appear in Appendix A.1.

Some words in this list contain spelling mistakes or are two words without a space between them: *cliam*, *antichoice*, *popband*, *sometimesheated*, *unstopable*, for example. For spelling mistakes, edit distance could be used to identify whether a word is an incorrectly spelled form of a word in a lexicon - if the misspelled word is below some threshold then we mark it as in the lexicon. This would capture *cliam*, which we assume is the misspelled form of *claim*, in the list above.

Additional groups of words include:

- Related to religion: *Islamists*, *dogmas*, *Muslim*, *buddhism*, *baptised*, *anti-semitic*
- Political beliefs: *far right, marxist*
- Violence: *militias, hijacked, battalion, attacking, terrorists, murdered, martyred, far-right, anti-terrorists*

The clear groupings of some of these words indicates that additional lexicons may be useful for identification.

4.2 Lexicon Ablation

To empirically test the effectiveness of each lexicon, an ablation study was performed. It is likely that some lexicons provide more useful discriminatory power to the model, and the results of this study reveals which types they are. We removed each lexicon from the feature set one by one and retrained and tested the model. We report model accuracies, f1, precision, and recall in Table 4.3. We also include baseline results for comparison (*None*) and results of dropping every lexicon (*All*).

Removing all the lexicons has the highest (negative) impact on model performance as measured by both accuracy and f1, however generally removing any individual lexicon also hurt performance across most performance metrics.

Removing the *subjectivity* lexicon decreases the Top 1 accuracy and f1 scores by the highest amount, suggesting that it is an important lexicon.

Few lexicons produced a decrease in recall when removed, and if they did it was very small (-0.005). Changes in f1 after removing a lexicon seem to be driven by changes in precision - suggesting that the lexicons are most important for correctly picking the biased words out of the words returned as potentially biased.

It's interesting to note that dropping the entailment and sentiment lexicons produced an increase in Top 1 accuracy, although a decrease is seen is Top 2 and 3. These two

Lexicon dropped	Top 1	Top 2	Top 3	f1	Precision	Recall
None	**29.57**	**43.88**	**53.81**	**0.262**	**0.219**	**0.325**
All	25.11	41.99	52.81	0.254	0.218	0.303
Report	29.44	42.43	50.22	*0.263*	*0.221*	0.325
Entail	*30.3*	41.99	49.78	0.261	0.218	0.325
Assertives	29.44	41.99	50.22	0.259	0.217	0.32
Factives	29.44	41.99	50.22	0.262	0.219	0.325
Hedges	29.44	42.43	50.65	0.262	0.219	0.325
Implicatives	29.44	42.43	50.22	0.262	0.219	0.325
Bias lexicon	29.44	42.43	52.81	0.26	0.218	0.32
Subjectivity	26.41	41.56	49.78	0.259	0.217	0.32
Sentiment	*30.3*	40.69	51.08	0.262	*0.22*	0.325

Table 4.3: Model accuracy (%) on the test set after removing the features extracted using each lexicon. Baseline is in **bold**. Unexpected results are in *italics*.

lexicons are the largest (entailment) and third largest (sentiment). This increase is not reflected in the f1 score. This could indicate that larger lexicons slightly harm accuracy at the sentence level - perhaps they are ”over-identifying” words.

Figure 4.1 shows the difference in f1 achieved after removing each lexicon from the feature vector compared to the baseline. Again, removing all the lexicons clearly has the worst impact on f1. Removing the assertives and subjectivity lexicons also seems to have a high negative impact.

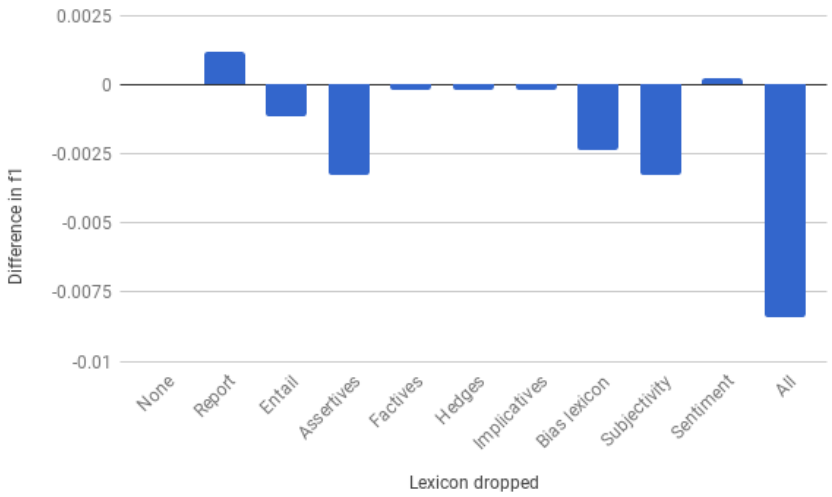
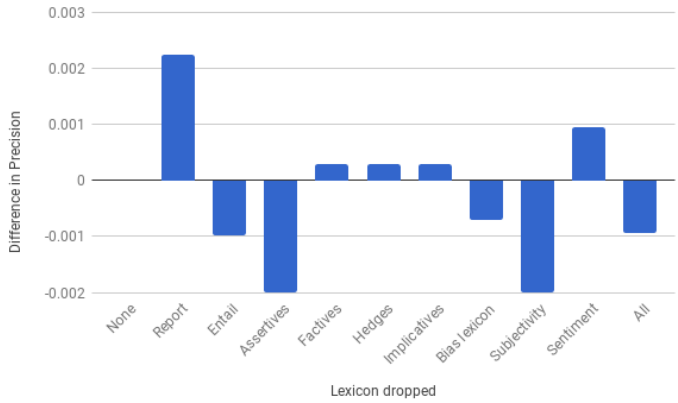
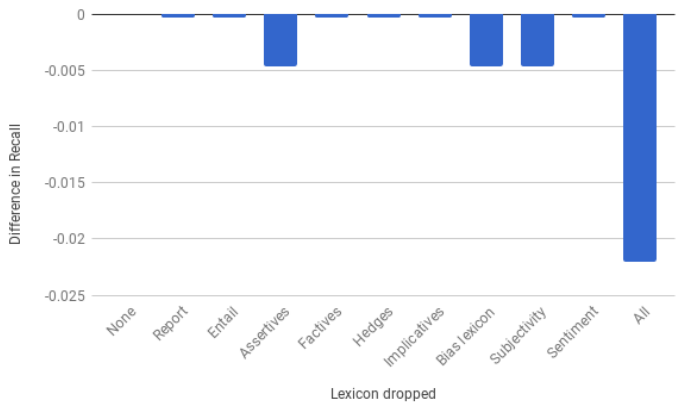


Figure 4.1: Difference in f1 when removing each lexicon from feature space.

Removing the entailment and sentiment lexicons showed an increase in Top 1 accuracy as seen in Table 4.3. Figure 4.1 also shows a slight increase in f1 after removing the sentiment lexicon, but this is very, very small. F1 decreases after removing the entailment lexicon - this is probably driven by the decrease in precision as seen in Figure 4.2a.



(a) Difference in precision.



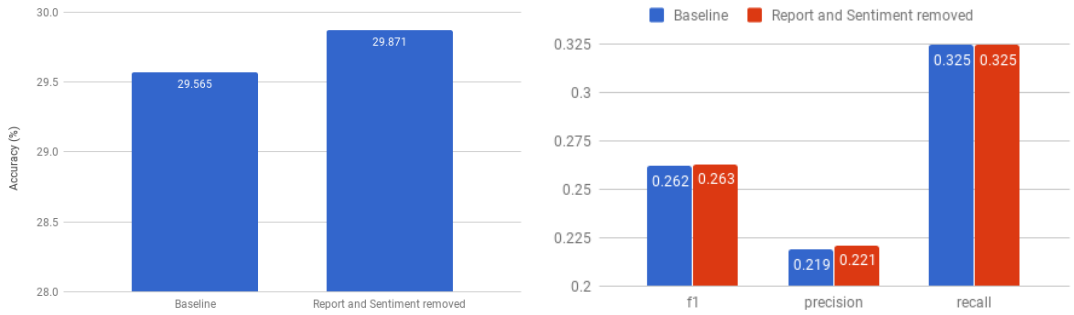
(b) Difference in recall.

Figure 4.2: Differences in precision (*top*) and recall (*bottom*) after removing each of the lexicons from the feature space.

By inspecting the differences in precision and recall after removing the *sentiment* lexicon we can conclude that the increase in f1 likely comes from the increase in precision. An increase in precision after indicates that more of the words that the model predicts to be biased are truly biased. This could suggest that the *sentiment* lexicon contributes well to identifying potentially biased words, because the recall decreases after removing it, but that few of these words are true biased words, because precision increases after removing it. The lexicon identifies 6789 words, making it the third largest by a considerable amount. Perhaps this contributes to the higher recall when including it.

We observe an increase in f1 after removing the *report* lexicon. Figure 4.2a suggests that this is driven by a large increase in precision after removing the lexicon.

Given the increases in model performance as measured by f1 score when removing the *sentiment* and *report* lexicons individually, we removed both at the same time to evaluate their combined effect on model performance. Figure 4.3 shows an increased accuracy and f1 after doing this. Again, f1 increase seems to be driven by an increase in precision. The increase in precision while recall remains the same suggests the *report*



(a) Difference in accuracy.

(b) Difference in f1, precision, and recall.

Figure 4.3: Differences in model performance after removing both the *report* and *sentiment* lexicons.

and *sentiment* lexicons were identifying false positives. However, the difference is very small - a increase in accuracy of 0.002, and 0.001 increase in f1.

4.2.1 Stopwords

Stopwords are frequently-occurring function words that do not affect the meaning of a sentence significantly if they are removed, for example *the, a, in, and, as*. Stopwords are often removed in NLP tasks as many tasks are interested in the content words only. Since stopwords carry little meaning, they are less likely to introduce bias into a sentence, so we hypothesised that a successful model would assign low class 1 probabilities to stopwords.

We inspected the baseline model's list of words in each sentence sorted by probability of being biased and found that stopwords almost always appear at the bottom of the list. 98.7% of words appearing at the bottom of the ranked list are stopwords as per `NLTK`'s English stopword list[1]. This suggests the baseline model correctly learns that function words like stopwords tend not to introduce bias.

4.3 Stopword Exclusion

Motivated by the observation that stopwords are the least likely words to be biased, we investigated the effect of excluding stopwords from the dataset before training. Excluding stopwords is a standard preprocessing step in many NLP tasks and has the potential to improve the baseline model's performance as it removes words which will (most likely) never be biased.

Biased edits whose *before* strings were stopwords were discarded already during preprocessing but stopword exclusion takes this further and discards all training samples whose word is a stopword.

[1] http://www.nltk.org/book/ch02.html, Section 4.1

In this section, we compare the effect of including and excluding stopwords and digits from the dataset before training. Removing stopwords changes the number of training and test samples we have (at the word level, not the sentence level). Changes in these counts are reported in Table 4.4. No biased words in the training or test set were stopwords.

	Train	**Test**
Baseline	49448	6511
Stopwords removed	30401	3895

Table 4.4: Number of training and test samples before and after removing stopwords.

Figure 4.4 shows that removing stopwords from the training set gives an increase in f1 driven by small increases in both precision and recall. This suggests that stopwords are unimportant for the bias detection task and should be removed before training a model.

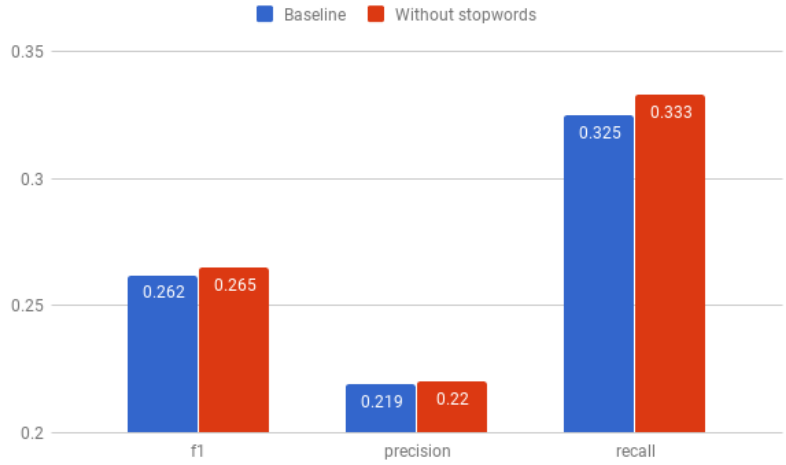


Figure 4.4: Comparison of f1, precision, and recall after removing stopwords from training examples.

Chapter 5

Extensions to the Baseline

This chapter presents experiments carried out on the baseline model and extensions made to improve accuracy on the bias detection task. We investigate varying the size of the context window. We also investigate classifying with neural networks, augmenting the feature vector with word and lemma embeddings, and lexicon similarity measures. We see good improvements in the latter experiments.

5.1 Context Window Size

The baseline model extracts features about the word itself and the words around it. The number of words before and after is referred to as the *context window*. Recasens et al uses a 5-gram context window: two words preceding, the current word, and two words after. It's possible that this is not the optimal window size for the bias detection task, so we experiment with varying the size and observe results on model accuracy.

Varying the context window size affects 15 features which depend on the context words: there is a feature for each lexicon to capture the membership of any context words, plus POS tags for each context word.

We individually vary the number of words before and after the current word, training and evaluating a logistic regression using combinations of up to 5 words before and up to 5 words after. The Top 1, 2, and 3 accuracies do not change significantly for any context window, and we report the seven combinations which increase any of the accuracies in Table 5.1.

We do not observe any increases in Top 1 accuracy when we vary the context window which indicates that the equal 5-gram context window size is the best for sentence-level accuracy. However, most the combinations listed improve on both Top 2 and 3 accuracies, with 1 word before and 4 words after producing the highest increase.

Considering up to 4 words after the current word seems like a large window, but makes sense when we revisit the types of words that the lexicons capture. Many of the lexicons describe verbs, with specific focus on the clause coming after the verb. For

Words before	Words after	Top 1	Top 2	Top 3
2	*2*	*29.57*	*43.88*	*53.81*
1	4	26.961	45.565	55.481
1	3	26.961	44.721	55.481
3	2	26.092	45.565	55.481
3	1	26.092	43.877	55.481
1	2	26.106	46.409	54.64
1	1	26.106	45.565	54.64
2	3	26.106	43.877	54.64

Table 5.1: Comparison of Top 1, 2, and 3 accuracies for feature vectors with varying context window sizes. Baseline accuracies in *italics*.

example, factives presuppose that their complement clause is true: *the study **revealed** that dogs are friendlier than cats* presents dogs being friendlier than cats as fact, rather than what the study suggests. Perhaps the increase in Top 2 and 3 accuracies when considering up to four words ahead of the current word is driven by the need to capture enough information about the clause after the current word.

5.2 Word Embeddings

Word embeddings have become an increasingly popular method for representing words in NLP tasks. Word embeddings are continuous vector representations of words, with the characteristic that words with similar meanings appear closer together in vector space. They are able to capture semantic and also some syntactic information. Mikolov et al (2013) [44] presents methods for learning these embeddings, namely skip-gram and continuous bag-of-words, which are now widely used for learning word embeddings. Mikolov later released `word2vec`, a collection of models for training word embeddings, and pre-trained word embeddings which we use in this section. For all experiments, we use the pre-trained `word2vec` embeddings, trained on 100 billion words using the Google News Corpus[1].

Word embeddings can be obtained in several ways. Continuous bag-of-words (CBOW), a popular method presented in [44], obtains word embeddings by training a feedforward neural network to predicted a target word given the two word before and after it. This is inspired by a key idea in distributional semantics: a word's meaning comes from the words surrounding it (Firth, 1957 [19]). This implies that words appearing in similar contexts have similar meanings.

The feature vectors used by Recasens et al represent words using a one-hot encoding. This produces a high-dimensional, sparse vector for each word (see Section 3.6), which, when combined with other features (some of which also require one-hot encoding such as the dependency relation), produces a feature vector tens of thousands of dimensions long. Replacing the one-hot encoding of words and/or lemmas would

[1] https://code.google.com/archive/p/word2vec/

reduce the dimensionality of the feature vector as well as provide semantic information that may be useful.

We conduct several experiments and extensions:

1. Replace one-hot encoding in baseline feature vector
   (a) Word embedding to replace word
   (b) Word embedding to replace lemma
2. Word embeddings as standalone features
   (a) Logistic Regression
   (b) Neural Network
3. Replace binary lexicon features with cosine similarities
   (a) Lexicon similarities + one-hot encoding
   (b) Lexicon similarities + word encoding
   (c) Lexicon similarities + lemma encoding
   (d) Only lexicon similarities and word embedding as features
   (e) Train and test a logistic regression and a feedforward neural network for feature configurations (a) to (d)

5.2.1 Replacing One Hot Encoding

In this section, we report results from experiments with replacing the one hot encoding of baseline feature vector with word embeddings. The study of this is motivated by findings from Kuang and Davidson (2016) [32] who found that augmenting the baseline feature vector with word embeddings increased f1 by 0.063 with improvements in both precision and recall.

Words and lemmas in the baseline model are represented using a one hot encoding. Given the vocabulary of the training set, we assign a feature for each of these words and place a 1 in the column that corresponds to the word and lemma of each training sample.

One hot encodings of words have a number of disadvantages.

- Scale: as the size of the vocabulary that is being encoded grows, so does the length of the encoding vector.
- Sparsity: only one feature is 1 and the rest are 0. For the space used to hold this vector, not much information is being represented at all
- Orthogonality: no similarity between words - they are all orthogonal to each other

- Unseen words: Since the vocabulary is fixed and based on the words seen in the training set, it's possible and likely that words will appear while testing that don't have a corresponding feature in the one-hot encoding. This is solved by using a special 'UNK' feature for words which haven't been seen before.

Word embeddings are an alternative representation for words in the bias detection task. We start by replacing the one-hot encoding of the word in the baseline feature vector with word embeddings. Since we are using pre-trained word embeddings, not every word in the training set is in the word embedding vocabulary. For these words, we use the 'UNK' embedding - a special embedding used for words not in the vocabulary when the `word2vec` embeddings were trained.

In the baseline, both the word and the lemma are encoded. We compare models trained on the following feature configurations to isolate the effect of replacing the one-hot encoding of words and lemmas with word/lemma embeddings. Unless otherwise noted, all other 243 features remain the same as the baseline. An embedding is 300-dimensional. For example, Model 3 uses one embedding to represent the word (300), one embedding to represent the lemma (300), and the rest of the features (243) to form a feature vector of total length 300 + 300 + 243 = 843.

Model number	**Word representation**	**Lemma representation**	**Feature Vector Length**
Baseline	One-hot	One-hot	23,043
Model 1	Embedding	One-hot	11,418
Model 2	One-hot	Embedding	12,468
Model 3	Embedding	Embedding	843
Model 4	Embedding	None	543
Model 5	None	Embedding	543

Table 5.2: Word and lemma representations used in experiments on replacing one-hot encoding. We only modify the representations of the word and lemma features so the 243 other features from the baseline model remain the same. Embeddings are of length 300.

Figure 5.1 shows the Top 1, 2, and 3 accuracies for each model compared to the baseline. Top 2 and 3 accuracies tend to gain an increase when using any type of word embedding, but this pattern does not appear to extend to the Top 1 accuracy. This suggests that word and lemma embeddings help the model to identify multiple potential biased words accurately, but they tend to be the second or third most likely words to be biased.

Word and lemma embeddings increase bias detection accuracy at the sentence level when considering multiple potentially biased words. The improvement of Top 2 and 3 accuracies but not Top 1 suggests that recall is being improved as more words are being returned as potentially biased. We report f1, precision, and recall to make further conclusions about the effect on word-based performance of word and lemma embeddings in place of one-hot encoding.

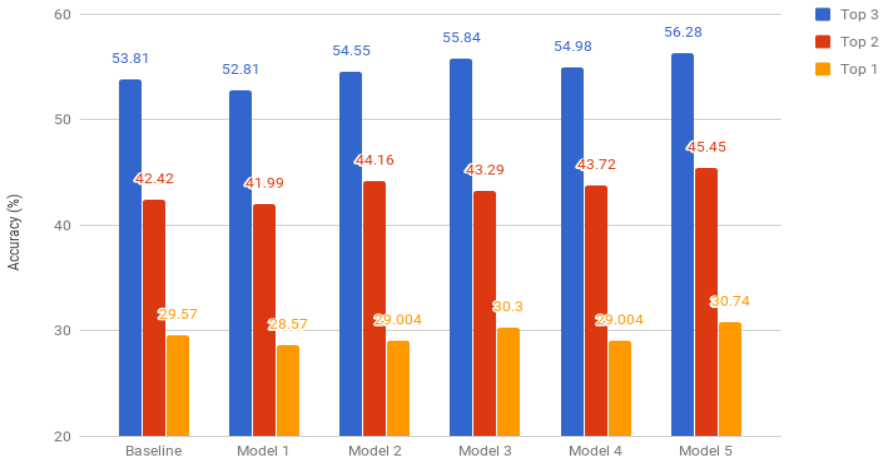


Figure 5.1: Comparison of Top 1, 2, and 3 accuracies on models with different configurations of word and lemma embeddings.

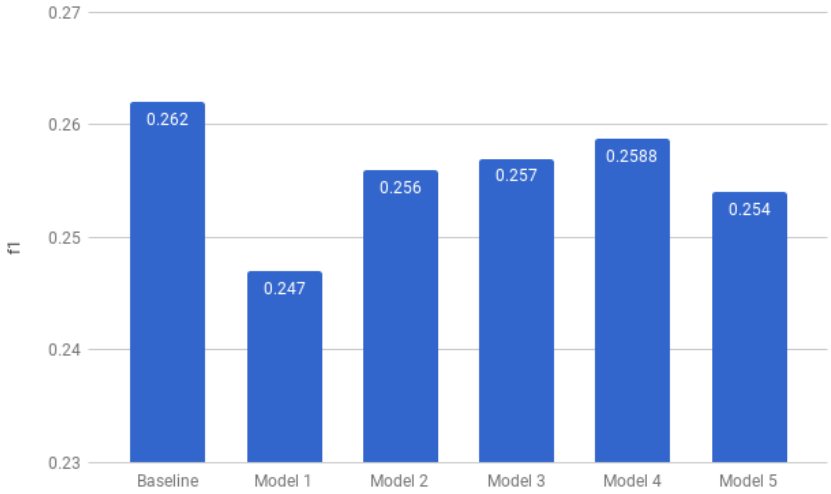


Figure 5.2: Comparison of f1 scores across models.

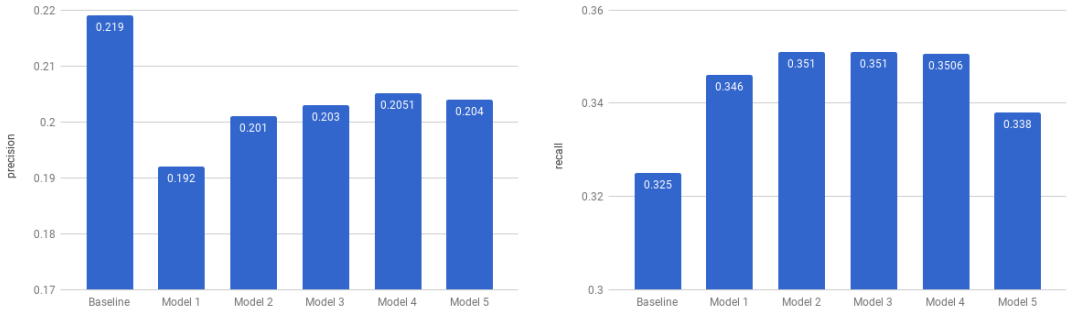
Figure 5.2 shows that f1 decreases when using a word or lemma embedding in any configuration that was tested. Model 1, a word embedding with a lemma one-hot encoding, shows the biggest decrease in f1. Model 4 achieves the closest f1 score to the baseline, suggesting that a feature vector that uses a word embedding representation and no representation of the lemma is best (out of the models that use embeddings).

Two models, 2 and 5, showed an increase in Top 1 accuracy over the baseline as seen in Figure 5.1. The f1 scores of these however do not show an increase over the baseline. This would suggest that these configurations of features are better for accurate identification of a single biased word in a sentence, but not for individually categorising words as biased.

Breaking down the models' f1 scores into precision and recall shows that using word / lemma embeddings decreases precision across all models when compared to the base-

line, but recall increases across all models.

Since precision and recall are word-based metrics, this indicates that using word and lemma embeddings helps models to identify more potentially biased words. This could be beneficial depending on the context in which a bias detection system is used; (examples here?).



(a) Precision across models

(b) Recall across models

Figure 5.3: Comparison of precision *left* and recall *right* across models.

The results also allow us to compare the effect of embedding the lemma (Models 2 and 5) vs the word (Models 1 and 4). As noted above, models 2 and 5 show an increase in Top 1 accuracy over the baseline, which suggests that using embeddings to represent the lemma is superior to representing the word. Between these two models, Model 5 gives the highest accuracy, which further suggests that a representation of just the lemma is better than representing the lemma and the word.

The conclusions above apply best if the sentence-level metric is most important in the context in which bias detection is performed. If classification performance on individual words, without being ranked against other words in the sentence, is most important then the feature set where the word is represented with an embedding and the lemma is not represented at all (ie deleted from the feature set completely) performs best, with the least decrease in precision compared to other feature sets, and highest increase in recall (8%) over the baseline.

There are slight differences between our results and those of Kuang and Davidson (2016) who also implement a logistic regression word-level classifier in the same manner as Recasens et al. They report an increase in f1 of 0.063 after augmenting the baseline feature vector with word embeddings, while we found a decrease. They do not report accuracies, so we cannot compare these to our model. They use custom trained `GloVe` embeddings while we used `word2vec` which is the likely explanation for the difference in findings, as well as differences in preprocessing methods and threshold values used for calculating f1 scores.

5.2.2 Word Embeddings as Standalone Features

Given the improvements in model performance when using word embeddings to replace the one hot encoding, we further experimented on the use of tri-gram word embeddings to make up feature vectors by themselves as input to a binary classifier. In this section we report findings from several models:

1. Baseline feature vector with a feedforward neural network
2. Trigram word embedding as feature vector
   (a) Logistic Regression
   (b) Feedforward neural networks
3. Best n-gram from context experiments
   (a) Feedforward neural networks

We use a logistic regression in order to be able to compare results from the new feature vector to the baseline. We also report results when neural networks with a single hidden layer compared to multiple hidden layers.

5.2.2.1 Neural Networks

Neural networks are models that can perform non-linear mappings from input to output. They are made up of inter-connected nodes which perform a (non-linear) transformation of their input, and send their output onto the nodes they are connected to. Connections between nodes have weights which are adjusted during training.

Feedforward neural networks are usually structured as layers of nodes where each layer's connections are only to the layer ahead of it - ensuring the information only propagates forward through the network. See Figure 5.4 for an example.

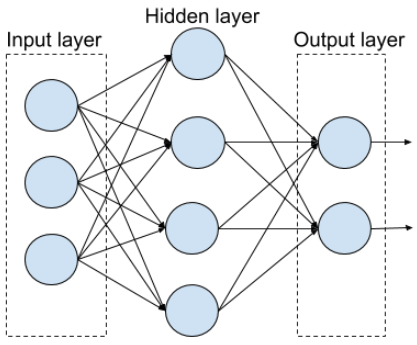


Figure 5.4: Topography of a feedforward neural network with a 3-dimensional input, a hidden layer of size 4, and 2-dimensional output.

Each node in the hidden and output layers takes as input the sum of the outputs of its previous nodes, weighted by the connection weights. Its output is the result of passing this sum through a non-linear function, called the activation function:

$$output = \phi(\sum_{i}^{N} w_i x_i) \tag{5.1}$$

where ϕ is a function such as *sigmoid*, *tanh*, or *ReLU* [21].

The set of weight matrices between layers is typically randomly initialised, and adjusted using backpropagation [39].

Neural networks have been shown to be able to model complex, non-linear relationships, especially deep neural networks (with multiple hidden layers). Deep learning with neural networks has advanced the state of the art in many areas of NLP (LeCunn et al, 2015 [38]). Motivated by this, we propose to investigate the effectiveness of neural networks on the bias detection task.

5.2.2.2 Neural Network Baseline

We first experiment with using a feedforward neural network as a replacement for the logistic regression classifier. This has two purposes: it gives us an initial understanding of how and to what degree classifying with a neural network improves over the baseline; secondly, it acts as a good baseline against which to compare further experiments.

We use Python machine learning library `sci-kit learn`'s `MLPClassifier`[2] class to initialise a feedforward neural network, train, and predict probabilities at test time. All other scripting - preparing the data for training, and testing the model - was implemented from scratch.

There are numerous hyperparameters that can be tuned when setting up a feedforward neural network. Significant ones include: activation function, size and number of hidden layers, optimiser, and learning rate. Regularisation - a group of techniques designed to prevent a neural network from overfitting to the training data - like dropout [53] and gradient clipping [47] can also be included while training. An `MLPClassifier` has default values for most of these parameters, but these are obviously not guaranteed to be optimal for each task. Initial results showed the *ReLU* activation function to produce the best results compared to *tanh* and *sigmoid*, this is also in line with recent findings (e.g. [31]).

Therefore, a grid search was performed over size and number of hidden layers and optimisers in order to gauge which configurations give better accuracy and f1 scores. We search over the following parameter space:

- Size of hidden layers: 100, 500, 1000, 100x100, 100x100x100, 500x500x500

[2] http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

- Optimiser: Adam [30], Stochastic Gradient Descent (SGD), LBFGS[3]

We report results in Table 5.3.

Hidden layer size	Optimiser	Accuracy	f1	Precision	Recall
100	SGD	22.51	0.231	0.22	0.24
100	Adam	26.84	0.196	0.25	0.16
100	LBFGS	22.51	0.205	0.23	0.19
500	SGD	21.65	0.231	0.22	0.24
500	Adam	24.68	0.209	0.2	0.22
500	LBFGS	23.81	0.18	0.23	0.15
1000	SGD	23.38	0.217	0.214	0.22
1000	Adam	24.03	0.216	0.19	0.25
1000	LBFGS	23.51	0.202	0.18	0.23

Table 5.3: Grid search over size of hidden layer and optimiser to identify the best combination. All models used a ReLU activation function.

There is a clear negative relationship between the hidden layer size and accuracy - increasing the number of units in the hidden layer tends to decrease accuracy (Figure 5.5. Precision also follows this pattern. However, f1 increases as the size of the hidden layer increases, driven by a large increase in recall. This suggests that a higher-dimensional hidden layer allows a neural network to identify more words that are potentially biased - indicating a direction for further investigation. However, none of these values are higher than the baseline accuracy and f1.

The results in Figure 5.5 are for neural networks with a single hidden layer. Adding more hidden layers gives the network the potential to learn more complex mappings, so we experiment further with increasing the number of hidden layers. Our results are reported in Table 5.4.

In several instances, precision and recall scores come close or are greater than baseline values:

- Hidden layer sizes 100 and 250 produce higher precision scores than the baseline
- Hidden layer sizes 100 x 100 x 100 and 250 x 250 x 250 produce higher recall scores than the baseline

These observations indicate that multiple hidden layers, specifically three, is important for achieving higher recall. However, precision is best in networks with a single hidden layer. More generally, as the number of hidden layers increases, recall is observed to increase, while precision decreases.

Overall, using a neural network classifier on the baseline features in the architectures we have investigated has increased precision and recall in separate models, showing that there is potential for neural networks to better identify biased words.

[3]https://en.wikipedia.org/wiki/Limited-memory_BFGS

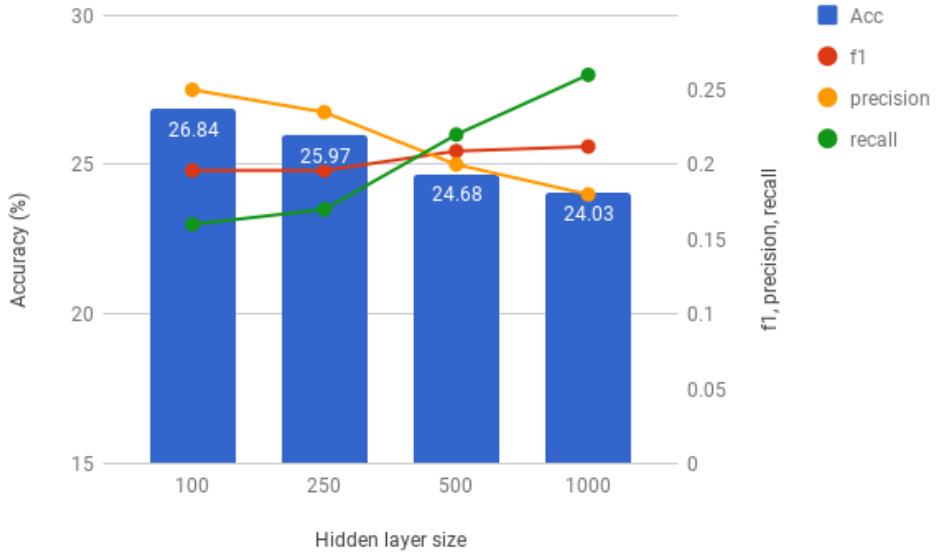


Figure 5.5: Accuracy, f1, precision, and recall for feedforward neural networks with ReLU activation function, Adam optimisation, and a single hidden layer of sizes 100, 250, 500, and 1000.

Hidden layer size	Accuracy	f1	Precision	Recall	Comments
100	26.84	0.196	0.25	0.16	Higher precision
100 x 100	22.94	0.199	0.16	0.27	
100 x 100 x 100	20.78	0.21	0.146	0.38	Higher recall
250	25.97	0.196	0.235	0.17	Higher precision
250 x 250	25.54	0.22	0.179	0.286	
250 x 250 x 250	22.1	0.195	0.135	0.351	Higher recall
500	24.68	0.209	0.2	0.22	
500 x 500	20.35	0.197	0.145	0.31	
500 x 500 x 500	22.94	0.182	0.134	0.29	

Table 5.4: Accuracy, f1, precision, and recall for neural networks with ReLU activation, Adam optimisation, and varying sizes and dimensionality of hidden layers (n x n x n means 3 hidden layers of n units each). *Higher* or *lower* comments mean higher/lower than the baseline.

5.2.2.3 Trigram word embeddings

We saw improvements over the baseline recall score of 8% after using embeddings to represent words or lemmas in the feature vector (Section 5.2.1). We experiment with a new feature vector, made up of trigram word embeddings, and report accuracy and f1 when classifying using a logistic regression and neural networks.

Figure 5.6 illustrates the feature extraction method for the word *and* in the phrase *the dog and the cat*. For each training sentence, we extract a training sample for each word in the sentence. The training sample for the word *and* is the concatenation of the word

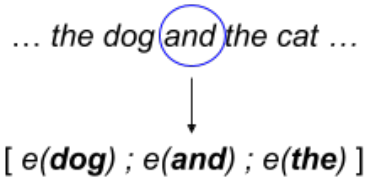


Figure 5.6: Feature extraction for the trigram word embedding experiments. For a given word *w*, *e(w)* gives its word embedding.

embeddings of the current word (*and*), its previous word (*dog*), and the following word (*the*).

The ReLU activation function and Adam optimiser showed success in our experiments with the baseline feature vector, so we use these parameter values as a starting point for exploring classification with neural networks. The same work also revealed interesting relationships between precision, recall, and size and number of hidden layers, so we perform an initial grid search over size and number of hidden layers. Results are reported in Table 5.5.

Hidden layer size	Accuracy	f1	Precision	Recall	Comments
100	22.1	0.183	0.23	0.15	Higher precision
100 x 100	17.75	0.2	0.195	0.21	
100 x 100 x 100	19.91	0.209	0.192	0.23	
250	20.35	0.157	0.169	0.147	
250 x 250	19.91	0.195	0.19	0.199	
500	23.81	0.198	0.241	0.167	Higher precision
500 x 500	22.51	0.195	0.18	0.21	
1000	25.97	0.213	0.256	0.18	Higher precision
1000 x 1000	18.18	0.18	0.215	0.16	
1000x1000x1000	21.65	0.204	0.204	0.203	

Table 5.5: Accuracy, f1, precision, and recall for neural networks classifying trigram word embedding feature vectors. All networks had ReLU activation and Adam optimisation and we vary the hidden layer size (n x n x n means 3 hidden layers of n units each). *Higher* or *lower* comments mean higher/lower than the baseline.

It is encouraging to note that we already achieve a higher precision score than the baseline in some cases! However, recall results are very low (baseline recall was 0.325). It's possible that the combination of activation function and optimiser is not as well-suited to classifying so we investigate optimising with stochastic gradient descent (SGD) and *tanh* activation functions as well. *tanh* is often used in binary classification problems, while *Adam* is a specialised version of *SGD*, so the latter is worth investigating to make sure we are not using a more complicated method than we need. We report results in Table 5.6.

Table 5.6 shows that *SGD* produces higher accuracies and f1 scores than the *Adam*

Activation	Optimiser	Hidden layer size	Accuracy	f1	Precision	Recall
relu	Adam	100	22.1	0.183	0.23	0.15
relu	SGD	100	22.94	0.213	0.177	0.268
relu	Adam	250	22.94	0.225	0.208	0.247
relu	SGD	250	23.81	0.234	0.194	0.294
relu	Adam	500	23.81	0.216	0.229	0.203
relu	SGD	500	25.11	0.249	0.206	0.316
relu	Adam	1000	25.97	0.213	0.256	0.18
relu	SGD	1000	25.11	0.247	0.201	0.32
tanh	Adam	100	20.78	0.191	0.176	0.208
tanh	SGD	100	25.54	0.225	0.199	0.26
tanh	Adam	250	22.51	0.165	0.187	0.147
tanh	SGD	250	24.68	0.231	0.198	0.277
tanh	Adam	500	22.1	0.164	0.178	0.15
tanh	SGD	500	22.1	0.241	0.209	0.286
tanh	Adam	1000	22.1	0.188	0.175	0.203
tanh	SGD	1000	23.5	0.245	0.212	0.29

Table 5.6: Accuracy, f1, precision, and recall for neural networks with a single hidden layer classifying trigram word embedding feature vectors.

optimiser for both *ReLU* and *tanh* activations. We therefore switch to *SGD* optimisation in all further experiments. We also observe that a *tanh* activation doesn't produce higher accuracies or f1 scores than *ReLU*, so we continue to use *ReLU*.

Again, we borrow insight from our first set of experiments (Section 5.2.2.2) which showed that recall increased as the number of hidden layers increased. We can observe this effect again in the results in Table 5.5, which suggests it would be useful to investigate additional hidden layers. These results are presented in Table 5.7.

Hidden layer size	Accuracy	f1	Precision	Recall
100	22.94	0.213	0.177	0.268
100x100	25.11	0.227	0.185	0.294
100x100x100	19.05	0.203	0.166	0.26
250	23.81	0.234	0.194	0.294
250x250	24.675	0.237	0.186	0.325*
250x250x250	18.6	0.192	0.176	0.21
500	25.11	0.249	0.206	0.316
500x500	28.14	0.256	0.205	0.342*

Table 5.7: Accuracy, f1, precision, and recall for neural networks classifying trigram word embedding feature vectors for varying sizes and number of hidden dimension layers. Activation is *ReLU*, optimisation with *SGD* for all networks. *means a result higher than the baseline.

Table 5.7 shows that we can improve upon the baseline's recall score using a neural network with 2 hidden layers of 500 nodes each. Accuracy is also just 1.43% below

the baseline accuracy level which indicates that the neural network performs almost as well at the sentence level - given a sentence, it can identify the biased word in the sentence at almost the same accuracy as the baseline.

The difference in feature extraction between the baseline model and this neural network is quite large - it took several days to extract all the features for the training samples for the baseline, compared to a few minutes for the trigram word embedding training samples. There is also a large difference in dimensionality of the feature vectors between the two models: the baseline takes a feature vector of length 23,043 whereas the trigram word embedding feature vectors are length 900 (3 x 300). In comparison, however, the number of parameters needed to define each model gives the advantage to the baseline which only needs the same number of parameters as there are features, whereas a neural network with input size 900, 2 hidden layers of size 500 each, and an output of 2 gives 900 x 500 x 500 x 2 = 450,000,000 parameters. This could explain why we see diminishing results at the top end of number and size of hidden layers - there is insufficient data to fit so many parameters accurately.

5.2.2.4 Integrating results from the context experiments

We saw slight increases in model performance as measured by sentence-level accuracy when using a 6-gram context window (one word before, and four words after) for some of the baseline features. We also saw increased performance when using a neural network to classify trigram word embeddings. We experiment with combining these to investigate if any further increase over the baseline is achieved; we use a neural network to classify 6-gram word embedding feature vectors. Figure 5.7 shows how the feature vector for the word *dog* in the phrase *the dog and the cat sat* would be extracted; we take the concatenation of the word embeddings of the current word (*dog*) along with one word before it (*the*) and the four words after it (*and, the, cat, sat*). This produces a feature vector of length 300 x 6 = 1800.

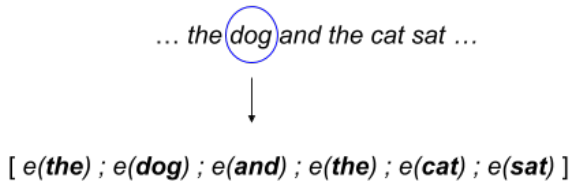


Figure 5.7: Feature extraction for the 6-gram word embedding experiments. For a given word *w*, *e(w)* gives its word embedding.

Despite varying parameters of the neural network, no configuration we tested produced especially convincing results. Our best model achieved:

- Accuracy: 24.68%
- f1: 0.0521

- precision: 0.0522
- recall: 0.0519

The low f1 score likely comes from a classification threshold that is too high for the model; the threshold is used to determine whether a test data point is in the positive or negative class based on its probability. We use the same threshold of 0.14 throughout our experiments to calculate f1, precision, and recall, and previous models achieve much better scores than the ones reported above.

From this, we can conclude that integrating the results from the context experiments has not worked well. This may be because the feature space in the two models were quite different - we found the optimal context window for the baseline feature set where a word in a sentence is represented by many different characteristics: lexical features (e.g. POS tags), lexicon membership, lexical features of its context words etc. However in this experiment, a word in a sentence is represented using an n-gram context of word embeddings. Therefore, the feature space is quite different; the optimal context window size on the baseline feature set is not necessarily the optimal context window size when using an n-gram word embedding representation.

5.2.3 Vector Space Similarity with Lexicons

Given the success of replacing the one hot encoding with word embeddings, we decided to further investigate how word embeddings could be used to improve over our baseline.

An interesting characteristic of word embeddings is the vector calculations that can be done with the embeddings. These reveal relationships between words. Cosine distance between embeddings indicates the 'similarity' of the two words, for instance. Patankar and Bose (2017) [48] calculate bias scores for articles using the average bias score of the sentences in the article. The bias score of a sentence is calculated by breaking the sentence into words and using the average cosine similarity between each word's embedding in the sentence and the bias lexicon used in this project (see Section 3.5.2.9).

We employ a similar approach in this experiment; for each word in the training set, we calculate its average similarity to each lexicon and use these averages to replace the binary lexicon features used in the baseline. For a word w and lexicon $\mathbf{L} = l_1, l_2, \ldots, l_n$, the entry in the feature vector is:

$$similarity(w, L) = \frac{1}{n} \sum_{i=1}^{n} cosine(e(w), e(l_i)) \tag{5.2}$$

$$cosine(x, y) = \frac{x \cdot y}{\| x \| \cdot \| y \|} \tag{5.3}$$

where $e(\mathbf{w})$ gives the word embedding of $\mathbf{w}$.

This involves translating each word in each lexicon to its word embedding. The entailment lexicon was not included in these experiments as it was too computationally expensive to work with given its size (52 million words).

While the binary feature extraction method is less computationally expensive, a lexicon similarity value would not result in such a sparse matrix as the original feature extraction method currently does. The sparsity comes from the lexicons only covering a small number of words - a majority of the words in the training set do not actually appear in any lexicon. This was discussed in Section 4.1. Lexicon similarity would also capture more graded relationships with each lexicon - the degree to which a word is similar to the lexicon as opposed to whether the exact word is in the lexicon or not. Word embeddings allow morphological variants of the word to also be considered similar to the lexicons as well.

This extension was implemented to investigate if lexicon similarity would give an increase in model performance as measured by accuracy and f1 compared to using binary features. We compare results of logistic regression (LR) and neural network (NN) classifiers trained on three different feature sets as described in Table 5.8.

	Classifier	**Lexicon features**	**Word representation**	**Vector Length**
Baseline	*LR*	*Binary*	*One-hot*	*23,043*
Model 1	LR	Lexicon similarities	One-hot	12168
Model 2	LR	Lexicon similarities	Word embedding	543
Model 3	NN	Lexicon similarities	One-hot	12168
Model 4	NN	Lexicon similarities	Word embedding	543
*Model 5**	NN	Lexicon similarities	Word embedding	311

Table 5.8: Feature vectors and models for six systems we evaluated. LR = logistic regression, NN = neural network. Baseline configuration given for comparison. *Uses only lexicon similarities and word embeddings in the feature vector - no other features from the baseline are used.

Best results obtained for each configuration are reported in Table 5.9.

Model	**Accuracy**	**f1**	**Precision**	**Recall**
Baseline	29.57	0.262	0.219	0.325
Model 1	30.3*	0.277*	0.238*	0.33*
Model 2	27.71	0.257*	0.21	0.342*
Model 3	27.71	0.282*	0.239*	0.342*
Model 4	31.17*	0.287*	0.224*	0.4*
Model 5	27.27	0.264*	0.209	0.36*

Table 5.9: Comparison of accuracy and f1 scores across the models described in Table 5.8. *means a result higher than the baseline.

Results from Models 1 and 2 show that replacing the binary lexicon features in the baseline model can improve both precision and recall over the baseline. Interestingly,

the better performing model used a one-hot encoding of the word instead of a word embedding. Overall, this indicates that our new lexicon similarity features are useful and provide an immediate improvement over the binary features.

For Models 3, 4, and 5 we evaluated multiple combinations of parameters, as we found in our previous experiments that this can help identify the highest performing models. Initial experiments showed that *logistic* and *tanh* activations were unable to compete with *ReLU* in terms of accuracy and f1. Initial experiments also suggested that *Adam* optimisation performed worse than *SGD* with *ReLU* activation, so we explore different sizes and numbers of hidden layers as we did in our standalone word embedding experiments.

We report best results for Model 3 in Table 5.10. We manage to consistently improve over the baseline in both precision and recall, which confirms that neural networks can better discriminate between biased and unbiased words.

Hidden layer size	Accuracy	f1	Precision	Recall
50	22.1	0.203	0.207	0.199
50x50	26.41	0.275*	0.23*	0.33*
100x100	26.84	0.27*	0.23*	0.325*
250x250	27.71	0.282*	0.239*	0.342*
500x500	28.14	0.264*	0.223*	0.325*

Table 5.10: Accuracy and f1 scores for Model 3: *ReLU* activation, *SGD* optimisation, and varying hidden layer sizes. *means a result higher than the baseline.

Hidden layer size	Accuracy	f1	Precision	Recall
250x250	31.17*	0.287*	0.224*	0.4*
100x100	29.437*	0.28*	0.223*	0.377*
500x500	28.87	0.262*	0.202	0.372*
1500	28.14	0.253	0.207	0.325*
500	30.74	0.25	0.207	0.32*
1000	29.44	0.24	0.195	0.312
100x100x100	25.97	0.23	0.18	0.33
500x500x500	25.11	0.22	0.18	0.29
100x100x100x100	25.54	0.209	0.17	0.18

Table 5.11: Accuracy and f1 scores for Model 4: *ReLU* activation, *SGD* optimisation, and varying hidden layer sizes. Results ordered by highest to lowest f1. *means a result higher than the baseline.

For Model 4, we report results in Table 5.11. These also show that we can achieve better precision and recall than the baseline; networks with two hidden layers are needed for this. We achieve higher improvements compared to Model 3 which indicates that representing the word with an embedding is better than using a one-hot encoding. It's possible that this improvement in performance comes from the smaller feature vector - Model 4's feature vectors are two orders of magnitude smaller than Model 3.

We report results for Model 5 in Table 5.12. We improve upon the baseline's f1 score in only one instance: a network with 3 hidden layers of size 100. However, this increase in f1 is driven by an increase in recall, while precision is worse than the baseline. We do not manage to improve upon precision in any instance, only matching it in one. This indicates that the full range of the baseline features is important for achieving good precision.

Hidden layer size	Accuracy	f1	Precision	Recall
50	24.68	0.195	0.187	0.203
100	21.2	0.198	0.186	0.212
250	21.65	0.191	0.188	0.195
500	20.78	0.196	0.197	0.195
50x50	21.65	0.257	0.226*	0.3
100x100	25.1	0.235	0.204	0.277
250x250	25.54	0.234	0.198	0.286
500x500	28.14	0.256	0.219*	0.307
50x50x50	28.14	0.257	0.203	0.35*
100x100x100	27.27	0.264*	0.209	0.36*
250x250x250	28.57	0.254	0.199	0.35*

Table 5.12: Accuracy and f1 scores for Model 5: *ReLU* activation, *SGD* optimisation, and varying hidden layer sizes. *means a result higher than the baseline.

Table 5.9 shows that Model 4 improves over the baseline in both sentence- and word-level metrics. Table 5.11 shows that this is consistent for several different architectures. Therefore, we can conclude that a feedforward neural network classifier and a feature set that uses lexicon similarity and word embeddings with the rest of the baseline features performs the best.

5.2.3.1 Discussion

The binary lexicon features capture lexicon membership in a crude manner: the feature value is 1 if the word matches exactly a word in the lexicon, and 0 if not. There is no consideration of morphological variants of the word; although some lexicons contain inflections of the stems as well (for example, the hedge lexicon contains both *claim* and *claims*). Allowing for this is important because many of the lexicons describe verbs, whose form changes to agree with their subject, but does not change the meaning of the verb itself. We argue it would be better to place the brunt of the work on a feature extraction process which could capture morphology automatically, instead of exhaustively listing every inflection of the lexicon entries. One way to do this would be as we have outlined in this section: calculate lexicon similarities between word embeddings.

A possible explanation for the increased performance with lexical similarity features is that word embeddings capture semantics instead of just word form, and cosine sim-

ilarity between them provides a graded measure of membership more suitable for capturing inflections and synonyms.

Chapter 6

Discussion and Future Work

6.1 Baseline

We implemented a baseline model for linguistic bias detection at the word level according to methodology in Recasens et al. We evaluate the baseline using accuracy when considering the top 1, 2, and 3 most likely words in the sentence to be biased. This was done in order to compare our results to Recasens et al. We achieve an average of 4.1% lower accuracy.

Evaluation We motivated the move to word-level evaluation metrics: f1, recall, and precision. Accuracy is a sentence-level metric - correct identification of the biased word depends on other words in the sentence - and does not tell us enough information about how our classifier is performing on individual words. Overall this has proven to be a good decision, as sentence-level accuracy shows little variation in our experiments, but f1, precision, and recall show systematic variation that allow us to draw conclusions.

Lexicons The baseline model makes use of nine lexicons to extract binary lexicon membership features. While lexicons are well suited to a word-based classifier, they are inflexible in the sense that only the exact words appearing in the lexicon will be matched; there is no consideration for morphological variants or words with similar meanings but different forms. We investigate a method for dealing with this by using cosine similarity to calculate the average similarity between a word and each lexicon. Using these features together with a word embedding representation and a neural network classifier produces the highest increases in performance of the baseline.

Levenshtein distance Levenshtein distance is one of the filters used to identify suitable edits to use as training data. It was used to filter out edits that did not make substantial changes to the *before* string. While this would discard trivial edits like *teh* - *the*, it would also discard edits whose edit distance is small but the change in meaning is significant like *any* - *all*.

Sentiment lexicon In the lexicon ablation studies, we saw that removing the sentiment lexicon actually increases the precision of the model, which suggests that the sentiment

lexicon plays a role in identifying false positives - it helps to flag up words as biased when they are not. Consider the following sentence from the training set:

Like other forms of discrimination such as homophobia the discriminatory or intolerant behaviour can be direct e.g. harassment assault or even murder, or indirect e.g. refusing to take steps to ensure that transgendered people are treated in the same way as nontransgendered people ' contains many words that appear in the sentiment lexicon:

- *like*: positive
- *discrimination*: negative
- *discriminatory*: negative
- *harassment*: negative
- *assault*: negative
- *murder*: negative
- *refusing*: negative

The biased word to be removed is actually *non-transgendered*, which is not in the sentiment lexicon or in any other lexicon. It's possible that, if the model is assigning a larger weight to the sentiment lexicon, that the many other words in the sentence which appear in the lexicon are ranked as more likely to be biased, pushing the true target word further down the list. While it's good that the lexicon is identifies many of the words in this sentence, as that reduces sparsity in the baseline feature vector, it may come at the cost of rating them too highly and therefore pushing the true biased word further down the list.

6.2 Feature Space Analysis Results

We reported the numbers of words in the training set that appear in at least one lexicon, discovering that the majority of both unbiased and biased words do not appear in any of the lexicons (68.3% of training data). Given that the lexicons were included to identify potentially biased words, they are the best features the model has to judge whether a word is biased. The factives lexicon covers the least amount of words, identifying just 45, of which none are biased. This called into question the effectiveness of the lexicon, but removing it from the feature space revealed a slight decrease in sentence-level accuracy.

Lexicon ablation studies revealed to what extent removing each lexicon impacts model performance in terms of sentence- and word-level metrics. We discovered that removing the sentiment and report lexicons together increases model performance. Increase in precision suggested that these models play a role in identifying false positives, since recall stayed the same. However, differences in metrics were very small, and further significance tests would be needed to confirm whether the differences are significant or not.

We also identified that stopwords were least likely to be classified as biased by the model, which aligned with our intuitions. Excluding stopwords from the training set improved both sentence-level accuracy as well as both precision and recall, which suggested that stopwords are not useful for the bias detection task. This is expected; it is difficult to imagine a sentence in which the bias-inducing word is *to*, for example.

6.3 Experimental Results

We gained improvements over the baseline from several extensions:

- Using the baseline feature vector with a neural network classifier
- Replacing one-hot encoding with embeddings
- Augmenting the feature vector with lexicon similarities instead of binary features
- Combining lexicon similarity features with word embedding representations and the remaining baseline features

6.3.1 Replacing One-hot Encoding

It was expected that using word embeddings over a one-hot encoding would increase model performance, because embeddings capture both semantic and some syntactic information in a more useful manner whereas each word representation in a one-hot encoding is orthogonal to every other word, so there is no information about semantic similarity. The one-hot encoding also produced a feature vector that was longer than the word embeddings by two orders of magnitude (Table 5.2). The curse of dimensionality [4] proposes that the number of training examples needed to fit a function grows with respect to the number of features used to represent the training examples - the more input features there are, the more training samples are needed. Since word embeddings need less features to represent them - we use embeddings of length 300 - there are relatively more training samples per feature than a one-hot encoding. It's possible that this difference helps the model better discriminate between biased and unbiased words. This is supported by an 8% increase in recall in models which use embeddings instead of one-hot encodings.

6.3.2 Neural Networks

When using a neural network classifier instead of a logistic regression on the baseline feature vector, we see improvements over the baseline in precision when using a single hidden layer and recall when using three hidden layers. Precision decreases as the number of hidden layers increases, which may be due to the lack of training data for higher numbers of parameters. This is further supported by the fact that the highest increase in recall was achieved by the network with three hidden layers of the smallest size tested.

The *rule of 10*[1] acts as a guideline for linear models, suggesting that at least 10 training samples per parameter are needed for good performance. It has been suggested as a good lower bound for number of training samples needed for neural networks[2],[3]. Our training set contains 49k samples, so according to the rule of 10 this would be enough for around 4900 features. The baseline model fits 23,043 features, an order of magnitude above 4900. The neural networks that reached higher precision values however, had many more than the logistic regression due to the hidden layer(s) in the network. Their ability to achieve higher precision and recall suggests that the logistic regression is not powerful enough for the bias detection task.

There are also practical factors to consider when switching to neural network classifiers: these models took longer to train than a logistic regression, and require more storage due to a larger number of parameters. Increasing size and depth of hidden layers increases training time, computational resources needed, and also the amount of training data needed to achieve a good performance. In several experiments, increasing the number of hidden layers increased f1 score but only until a certain point - this could have been due to insufficient data for the number of parameters that needed to be learned.

6.3.3 Vector Space Similarity with Lexicons

We see good improvements over the baseline model when combining a word embedding representation with vector space similarity to lexicons instead of binary lexicon features. We achieve improvements in both precision and recall when classifying with a neural network with two hidden layers of size 250. Increasing the size and number of the hidden layers decreased performance, which may be due to insufficient training samples as discussed above.

The entailment lexicon was not included in the vector space similarity feature extraction due to its size (52 million words) and unavailable computational resource to calculate the average similarity between these words and each word in the training set. Further work could include this lexicon to investigate its effect.

6.3.4 Dataset

This dataset was extracted in 2013. Given another 5 years' worth of edits, it may be worth using the same method to build a bigger, updated dataset. This would provide many more training samples and thus allow for bigger networks to be trained and evaluated. Since each Wikipedia article is being continuously edited (ie it is never "finished"), there may have been some biased words in the sentences we used that weren't edited out already.

[1] https://en.wikipedia.org/wiki/One_in_ten_rule

[2] https://medium.com/@malay.haldar/how-much-training-data-do-you-need-da8ec091e956

[3] http://fastml.com/how-much-data-is-enough/

6.4 Class Imbalance

Classification models are strongly influenced by the class priors in the training dataset - that is, the counts of each class present in the training set can affect the performance of the model in sometimes undesirable ways. In extreme cases, the model will not learn anything about the underlying structure of the data and will simply predict the majority class all the time because it leads to a higher accuracy.

Table 6.1 displays the counts of each class in the training and test sets. There is a clear dominance of the unbiased class at a ratio of more than 19:1. We can conclude that the baseline model doesn't simply predict unbiased for all the test data - if it did, the accuracy would be 96.3%. However, the vast imbalance could mean that useful information about the biased samples is being obscured by the sheer quantity of the unbiased samples. This could happen in both linear models and neural networks [43].

	Dataset			
	Train		**Test**	
Class	**Count**	**Percentage**	**Count**	**Percentage**
Unbiased	47622	96.31	6280	96.45
Biased	1826	3.69	231	3.55
Total	49448		6511	

Table 6.1: Number of words (ie number of samples) in each class.

Class imbalance is a well-known problem in machine learning, and some research has focused on mitigating its effects (e.g. Liu et al, 2009 [41]; Japkowicz et al, 2002 [25]; Batista et al, 2004 [2]).

Several methods are available that involve re-sampling the dataset. Augmenting a dataset with synthetically generated data points of the minority class is called *over-sampling*. Removing data points of the majority class is known as *undersampling*. The SMOTE algorithm [13] can be used to carry out both methods. Other methods for dealing with class imbalance include adjusting the class priors

We experimented briefly with SMOTE on the baseline model to see if any clear improvements could be made by re-sampling the dataset. We tried oversampling the minority (biased) class, undersampling the majority (unbiased) class to 2000 to be comparable to the minority, and a mixture of the two where both classes were re-sampled to have 5000 counts each. However, sentence-level accuracy decreased in each instance.

Given our success with neural networks in Chapter 5, work going forward should focus on neural methods for classification instead of linear. We have shown promising results using neural networks through our experiments on this imbalanced dataset, so it's possible that our concerns about class imbalance are no longer as significant as when we were using linear models. Existing research has reported improvements when using re-sampling techniques and neural classifiers (e.g. [59], [46]), so future work could focus on combining these for the bias detection task.

6.5 Future Work

While our work has contributed to the progression of the bias detection task, there are many possible avenues for future focus. We discuss our ideas for future work and outline at a high level some solutions.

6.5.1 Training Word Embeddings

In these experiments we used pre-trained *word2vec* word embeddings. This was because we did not have enough data to train our own word embeddings; typical datasets used for obtaining word embeddings contain billions of words. `word2vec`, for example was trained on 100 billion words[4].

Given more data, it would be interesting to train our own word embeddings. This would provide several advantages:

- Embeddings would be optimised for the Wikipedia Edits dataset and the bias detection task
- Get representations for uncommon words that were not in the *word2vec* vocabulary but are important content words in the article that the sentence appears in.

To overcome the limitation of insufficient data, word embeddings could be initialised to `word2vec` values and then fine-tuned with additional training using the Wikipedia Edits dataset.

6.5.2 Document Representations

We saw and experimented with word embeddings in Section 5.2. Since their widespread adoption within NLP, further work has been done to develop methods for producing embeddings for larger units: sentences, paragraphs, and even documents.

Previous to document embeddings, simple bag-of-words representations were used. These can be obtained in a straightforward way: given a corpus of documents and a vocabulary, each document is represented as a vocabulary sized vector, with each entry's value set to the number of times that word occurs in the document. This approach is simple to calculate and just relies on counting methods, but ignores sequences of words and word semantics.

Le and Mikolov (2014) [37] present an unsupervised method for learning embeddings of arbitrary word length that aims to use both the sequence and semantics of the words to produce the embeddings. This overcomes limitations of the bag-of-words approach. `gensim doc2vec`[5] is an open source Python package that implements this (along with `word2vec`).

[4] https://code.google.com/archive/p/word2vec/, section 'Pre-trained word and phrase vectors'

[5] https://radimrehurek.com/gensim/models/doc2vec.html

Other representations have also been proposed, such as fuzzy bag-of-words (Zhao and Mao, 2017 [66]) which uses cosine similarity between word embeddings to encode more semantics in its document representations than simple bag-of-words. They achieve higher accuracies on several document classification tasks using the two methods they propose compared to bag-of-words, Latent Semantic Analysis (LSA) ([36]), Latent Dirichlet Allocation (LDA) ([7]), and Word Mover's Distance ([34]) representations.

These embeddings of larger units could provide a bias detection system with further contextual information that could potentially be useful. Embeddings could be used as additional features in a feature vector for a word-based bias detection model.

Unsupervised methods for representing documents for classification have also been proposed. Butnaru et al (2017) [12] use k-means clustering to obtain "super words" of word embeddings from a collection of documents, and represent each document as a "bag-of-super word embeddings". This method achieves higher accuracy on a polarity classification task - predicting sentiment from movie reviews. This method could potentially be adapted to classify Wikipedia articles or paragraphs into *biased* or *unbiased*.

6.5.3 Topic Modeling

Each edit that we use as a training sample comes from a sentence in a Wikipedia article. The range of articles in the Wikipedia NPOV Corpus that we used is wide, covering items from political events (e.g. *Venezuelan recall referendum, 2004*) to topics relating to religion (e.g. *Islamophobia*) to food and culture (e.g. *Instant noodles*). Topic modeling would help group articles into similar topics, and an analysis of these groupings may reveal that some groups contain similar types of linguistic bias.

Boydstun et al (2013) [9] identify hierarchical topic modeling as a potentially effective method for detecting frames both generically and within specific issues or topics.

`gensim` is a Python package that can be used for topic modeling using Latent Dirichlet Allocation (LDA)[6]. LDA models topics as a distribution over words; the combination and counts of words appearing in the document are indicative of which topics the document covers. Models of topics can be inspected to see which words are most important for recognising that topic, while documents can be tagged with the topic(s) that they cover.

LDA could be performed over the articles with biased sentences in from the Wikipedia NPOV Corpus. It's possible that some topics would show similar types of linguistic bias: articles about politics, for example, could

Some words may be biased in some domains, but unbiased in others. *orthodox*, for example, has multiple meanings[7]:

[6] https://radimrehurek.com/gensim/wiki.html#latent-dirichlet-allocation

[7] https://en.oxforddictionaries.com/definition/orthodox

1. *Of the ordinary or usual type; normal.*
2. *Following or conforming to the traditional or generally accepted rules or beliefs of a religion, philosophy, or practice.*

Meaning 1 is more general than meaning 2, which relates specifically to a set of beliefs.

(3) shows a sentence pair from the article *Mormonism and Christianity*. This sentence appears in the test set, and the biased word *orthodox* was edited to *traditional*. Since it is describing Christianity, *orthodox* here follows Meaning 2.

3. (a) *However, the Latter Day Saint movement's uneasy relationship with **orthodox** Christianity has not changed.*

   (b) *However, the Latter Day Saint movement's uneasy relationship with **traditional** Christianity has not changed.*

Contrast (3) with (4), a sentence from the article *History of Alternative Medicine* that is not related to religion and contains the word *orthodox*. Here our understanding of the word *orthodox* is related to *mainstream*, *accepted*, *widely practised*; much more similar to Meaning 1 than Meaning 2.

4. *Confirmations of the unscientific nature of **orthodox** medicine starts coming in as many physicians start to openly realize the poverty of scientific evidence to support the majority of current medical practices as evidence-based medicine (EBM) started to develop.*

The word *orthodox* in relation to religion has gained negative connotations[8,9], often being used to mean traditional or conforming, implying lack of open-mindedness. Topic modeling could help us identify the domain in which a potentially one-sided or loaded term appears, thereby allowing a decision about whether to investigate its potential bias further. For example, a document which has religion as one of its topics and also contains the word *orthodox* could be flagged up for further investigation. Therefore, topic modeling could provide the first step in filtering out numerous edits to ones that are likely to need further consideration by either human editors or automatic bias detection systems.

6.5.4 Detecting Biased Sentences

A model that identifies biased words in a sentence is only one part of the bigger task of debiasing a text. Recasens et al's method identifies biased words in sentences that were already known to contain bias, but in real-world text not every sentence contains bias. How do we identify those that do in the first place?

This task could also be regarded as a classification problem: given a sentence, does it contain bias or not? If it does, we can then pass it to the word-level bias detection model.

[8] https://www.huffingtonpost.com/elad-nehorai/dontcallmeanorthodoxjew_b_1596960.html
[9] https://en.wikipedia.org/wiki/Orthodoxy#Non-religious_contexts

Classification could be performed in a number of ways. We would probably need a way of representing the sentence to be classified. This could come in the form of a bag-of-words representation or sentence embeddings, for example (see Section 6.5.2 above). Alternatively, features could be extracted from each sentence based on the lexicons we used in this project, for example the counts of words in the sentence that appear in each lexicon. A sentence representation along with its target (whether it contains biased language or not) would then constitute a training sample for a classifier.

Detecting biased sentences could also be framed as a sequence classification task. A sentence is typically broken down into words. A sequence model then takes as input each word in the sentence one by one, and outputs a class once it has consumed all of the input.

This is analogous to sentiment analysis in which a sequence of words is predicted to have positive or negative sentiment. Recurrent Neural Networks (RNNs) have frequently been used for sentiment analysis and could work well for detecting biased sentences. Since RNNs produce output at every time step, everything apart from the last output could be ignored.

Chapter 7

Conclusion

In this project, we have addressed the problem of linguistic bias detection. The task is framed as a classification task: given a sentence, which word in the sentence is biased? We use the Wikipedia NPOV Edits dataset from Recasens et al, a collection of edits that give pairs of strings before and after a change to a Wikipedia article. The articles chosen were in the NPOV Disputes section of Wikipedia because they violate Wikipedia's Neutral Point of View (NPOV) writing policy - they contain biased text that must be removed.

We implemented a baseline model for linguistic bias detection at the word level according to methodology in Recasens et al. We evaluate the baseline sentence by sentence, calculating accuracy when considering the top 1, 2, and 3 most likely words in the sentence to be biased. This was done in order to compare our results to Recasens et al. We achieve an average of 4.1% lower accuracy than they report. Differences in model performance may be down to differences in training and testing data - although we achieved similar statistics after dataset filtering, this does not guarantee the data itself is the same as what Recasens et al used.

We performed an analysis of the baseline feature space using lexicon ablation, statistics about lexicon coverage, and identified topics in out-of-lexicon words. We concluded that stopwords may be introducing unnecessary noise during training. The usefulness of the factives lexicon was called into question, since it fails to identify any biased words at all in the training set. Removing this lexicon results in a slight decrease in accuracy in the baseline model, so it is concluded that a lexicon's individual coverage of words in the training set is not as important as the coverage of the combination of lexicons. Analysis of out-of-vocabulary words can be grouped into religious, political or violent meanings, indicating that topic modeling may improve the system by identifying topics that may be more at risk of containing biased language.

We performed experiments to answer the following research questions:

1. Is a 5-gram the best context window to use?
2. Do classifiers that capture non-linear, complex relationships (ie neural networks) perform better than linear models?

3. Are embeddings better representations of the word and lemma?

4. Can we achieve comparable or better performance using word embedding feature vectors?

5. Does modeling lexicon similarity instead of binary lexicon membership improve performance?

We found that a context window of one word before and four words after the current word produced an increase in Top 2 and 3 accuracies, but not in Top 1. This suggested that using this context window would increase recall of the model.

The baseline feature vector represents both the word and the lemma as one-hot encodings. We replaced these with word/lemma embeddings from pre-trained `word2vec` vectors. Using embeddings for either form increased the Top 2 and 3 accuracies, which suggested that embeddings help improve recall. This is confirmed by increases in recall of up to 8% over the baseline. However, precision decreased from the baseline value, which suggested that using embeddings may return more false positives. Representing the word with an embedding and not representing the lemma at all performed the best. This may be due to the much smaller length of the feature vector when using this set of features.

We took recent successes in applications of neural networks to NLP tasks as motivation to explore their performance on the bias detection task. We found that even applied to the baseline feature vector, networks with a single hidden layer outperform the baseline's precision by more than 14%, while deeper networks gain at 17% increase in recall.

We constructed a new feature set made up of a trigram context of word embeddings and again showed an increase in precision of 16.9%, though recall scores were poorer than the baseline. Switching from *Adam* to *SGD* optimisation produced higher recall than the baseline by 5%.

Lastly, we used cosine similarity between word embeddings to calculate average similarity between each sample and the lexicons' words, replacing the binary lexicon features. Combining these features with word embedding representations and the rest of the baseline features and classifying with a neural network produced increases in sentence-level accuracy by 5.4%, f1 by 9.5%, precision by 2.3%, and recall by 23.1%.

We identified several areas for future work:

- **Class imbalance**: re-sampling methods for dealing with class imbalance at the data level could produce better results when combined with neural network classifiers as shown by recent research.

- **Training word embeddings**: could produce word embeddings that are better suited to the bias detection task, further increasing performance. -

- **Document representations**: could be useful for providing additional contextual information for a bias detection system.

- **Topic modeling**: could help identify areas that contain biased language in a document or documents that are more at risk of containing biased language.
- **Identifying biased sentences**: this is a necessary step before word-level bias detection. This could be done with a sequence model like an RNN in a similar manner to POS tagging or NER.

Bibliography

[1] Ricardo Baeza-Yates. Data and algorithmic bias in the web. In *Proceedings of the 8th ACM Conference on Web Science*, pages 1–1. ACM, 2016.

[2] Gustavo EAPA Batista, Ronaldo C Prati, and Maria Carolina Monard. A study of the behavior of several methods for balancing machine learning training data. *ACM SIGKDD explorations newsletter*, 6(1):20–29, 2004.

[3] Eric PS Baumer, Francesca Polletta, Nicole Pierski, and Geri K Gay. A simple intervention to reduce framing effects in perceptions of global climate change. *Environmental Communication*, 11(3):289–310, 2017.

[4] R. Bellman and R.E. Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton Legacy Library. Princeton University Press, 1961.

[5] Jonathan Berant, Ido Dagan, Meni Adler, and Jacob Goldberger. Efficient tree-based approximation for entailment graph learning. In *The 50th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, July 8-14, 2012, Jeju Island, Korea - Volume 1: Long Papers*, pages 117–125, 2012.

[6] Camiel Beukeboom. Mechanisms of linguistic bias: How words reflect and maintain stereotypic expectancies. In *Social Cognition and Communication*, pages 313–330, 2014.

[7] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[8] Tolga Bolukbasi, Kai-Wei Chang, James Y. Zou, Venkatesh Saligrama, and Adam Tauman Kalai. Man is to computer programmer as woman is to homemaker? debiasing word embeddings. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 4349–4357, 2016.

[9] Amber E Boydstun, Justin H Gross, Philip Resnik, and Noah A Smith. Identifying media frames and frame dynamics within and across policy issues. 2013.

[10] Engin Bozdag. Bias in algorithmic filtering and personalization. *Ethics and Inf. Technol.*, 15(3):209–227, September 2013.

[11] Thorsten Brants. Tnt: a statistical part-of-speech tagger. In *Proceedings of the sixth conference on Applied natural language processing*, pages 224–231. Association for Computational Linguistics, 2000.

[12] Andrei M Butnaru and Radu Tudor Ionescu. From image to text classification: A novel approach based on clustering word embeddings. *Procedia Computer Science*, 112:1783–1792, 2017.

[13] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[14] Dennis Chong and James N Druckman. Framing theory. *Annu. Rev. Polit. Sci.*, 10:103–126, 2007.

[15] Johannes Daxenberger and Iryna Gurevych. Automatically classifying edit categories in wikipedia revisions. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013, 18-21 October 2013, Grand Hyatt Seattle, Seattle, Washington, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 578–589, 2013.

[16] Michela Del Vicario, Alessandro Bessi, Fabiana Zollo, Fabio Petroni, Antonio Scala, Guido Caldarelli, H. Eugene Stanley, and Walter Quattrociocchi. The spreading of misinformation online. *Proceedings of the National Academy of Sciences*, 113(3):554–559, 2016.

[17] Cicero dos Santos and Maira Gatti. Deep convolutional neural networks for sentiment analysis of short texts. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 69–78, 2014.

[18] Robert M Entman. Framing: Toward clarification of a fractured paradigm. *Journal of communication*, 43(4):51–58, 1993.

[19] J.R. Firth. A synopsis of linguistic theory 1930-1955. In *Studies in Linguistic Analysis*, pages 1–32. Blackwell, Oxford.

[20] William A Gamson and Andre Modigliani. Media discourse and public opinion on nuclear power: A constructionist approach. *American journal of sociology*, 95(1):1–37, 1989.

[21] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.

[22] Sara Hajian, Francesco Bonchi, and Carlos Castillo. Algorithmic bias: From discrimination discovery to fairness-aware data mining. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 2125–2126. ACM, 2016.

[23] Joan Hooper. On assertive predicates. *Syntax and Semantics*, 4:91124, 1975.

[24] Ken Hyland. Stance and engagement: a model of interaction in academic discourse. *Discourse Studies*, 7(2):173–192, 2005.

[25] Nathalie Japkowicz and Shaju Stephen. The class imbalance problem: A systematic study. *Intelligent data analysis*, 6(5):429–449, 2002.

[26] Andrej Karpathy. The Unreasonable Effectiveness of Recurrent Neural Networks, 2015 (accessed April 1, 2018). "`http://karpathy.github.io/2015/05/21/rnn-effectiveness/`".

[27] Lauri Karttunen. Implicative verbs. *LANGUAGE*, 47(2):340–358, 1971.

[28] Soo-Min Kim and Eduard Hovy. Extracting opinions, opinion holders, and topics expressed in online news media text. In *Proceedings of the Workshop on Sentiment and Subjectivity in Text*, SST '06, pages 1–8, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics.

[29] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[30] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[31] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[32] Sicong Kuang and Brian D Davison. Semantic and Context-aware Linguistic Model for Bias Detection. *Proc. of the Natural Language Processing meets Journalism IJCAI-16 Workshop, July 10, 2016*, pages 57–62.

[33] Julian Kupiec. Robust part-of-speech tagging using a hidden markov model. *Computer Speech & Language*, 6(3):225–242, 1992.

[34] Matt Kusner, Yu Sun, Nicholas Kolkin, and Kilian Weinberger. From word embeddings to document distances. In *International Conference on Machine Learning*, pages 957–966, 2015.

[35] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. *CoRR*, abs/1603.01360, 2016.

[36] Thomas K Landauer, Peter W. Foltz, and Darrell Laham. An introduction to latent semantic analysis. *Discourse Processes*, 25(2-3):259–284, 1998.

[37] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pages II–1188–II–1196. JMLR.org, 2014.

[38] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[39] Yann LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–50. Springer, 1998.

[40] Bing Liu, Minqing Hu, and Junsheng Cheng. Opinion observer: Analyzing and comparing opinions on the web. In *Proceedings of the 14th International Conference on World Wide Web*, WWW '05, pages 342–351, New York, NY, USA, 2005. ACM.

[41] Xu-Ying Liu, Jianxin Wu, and Zhi-Hua Zhou. Exploratory undersampling for class-imbalance learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 39(2):539–550, 2009.

[42] Aurélien Max and Guillaume Wisniewski. Mining naturally-occurring corrections and paraphrases from wikipedia's revision history. In *Proceedings of the International Conference on Language Resources and Evaluation, LREC 2010, 17-23 May 2010, Valletta, Malta*, 2010.

[43] Maciej A. Mazurowski, Piotr A. Habas, Jacek M. Zurada, Joseph Y. Lo, Jay A. Baker, and Georgia D. Tourassi. Training neural network classifiers for medical decision making: The effects of imbalanced datasets on classification performance. *Neural Networks*, 21(2):427 – 436, 2008. Advances in Neural Networks Research: IJCNN 07.

[44] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[45] Sudha Morwal, Nusrat Jahan, and Deepti Chopra. Named entity recognition using hidden markov model (hmm). *International Journal on Natural Language Computing (IJNLC)*, 1(4):15–23.

[46] Yi L Murphey, Hong Guo, and Lee A Feldkamp. Neural learning from unbalanced data. *Applied Intelligence*, 21(2):117–128, 2004.

[47] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013.

[48] A. A. Patankar and J. Bose. Bias discovery in news articles using word vectors. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 785–788, Dec 2017.

[49] Scott Plous. *The psychology of judgment and decision making.* Mcgraw-Hill Book Company, 1993.

[50] Marta Recasens, Cristian Danescu-Niculescu-Mizil, and Dan Jurafsky. Linguistic models for analyzing and detecting biased language. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, ACL 2013, 4-9 August 2013, Sofia, Bulgaria, Volume 1: Long Papers*, pages 1650–1659, 2013.

[51] Martin Rein and Donald Schön. Frame-critical policy analysis and frame-reflective policy practice. *Knowledge and policy*, 9(1):85–104, 1996.

[52] Hinrich Schütze and Yoram Singer. Part-of-speech tagging using a variable memory markov model. In *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*, pages 181–187. Association for Computational Linguistics, 1994.

[53] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[54] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc., 2014.

[55] Peter D Turney. Thumbs up or thumbs down?: semantic orientation applied to unsupervised classification of reviews. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 417–424. Association for Computational Linguistics, 2002.

[56] A Tversky and D Kahneman. Judgment under Uncertainty: Heuristics and Biases. *Science (New York, N.Y.)*, 185(4157):1124–31, 9 1974.

[57] Soroush Vosoughi, Deb Roy, and Sinan Aral. The spread of true and false news online. *Science*, 359(6380):1146–1151, 2018.

[58] Peilu Wang, Yao Qian, Frank K. Soong, Lei He, and Hai Zhao. Part-of-speech tagging with bidirectional long short-term memory recurrent neural network. *CoRR*, abs/1510.06168, 2015.

[59] Shoujin Wang, Wei Liu, Jia Wu, Longbing Cao, Qinxue Meng, and Paul J Kennedy. Training deep neural networks on imbalanced data sets. In *Neural Networks (IJCNN), 2016 International Joint Conference on*, pages 4368–4374. IEEE, 2016.

[60] Janyce Wiebe, Theresa Wilson, and Matthew Bell. Identifying collocations for recognizing opinions. In *Proceedings of the ACL-01 Workshop on Collocation: Computational Extraction, Analysis, and Exploitation.*, 2001.

[61] Janyce M Wiebe. Tracking point of view in narrative. *Computational Linguistics*, 20(2):233–287, 1994.

[62] Theresa Wilson, Janyce Wiebe, and Paul Hoffmann. Recognizing contextual polarity in phrase-level sentiment analysis. In *Proceedings of the conference on human language technology and empirical methods in natural language processing*, pages 347–354. Association for Computational Linguistics, 2005.

[63] Tae Yano, Philip Resnik, and Noah A. Smith. Shedding (a thousand points of) light on biased language. In *Proceedings of the 2010 Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk, Los Angeles, USA, June 6, 2010*, pages 152–158, 2010.

[64] Mark Yatskar, Bo Pang, Cristian Danescu-Niculescu-Mizil, and Lillian Lee. For the sake of simplicity: Unsupervised extraction of lexical simplifications from wikipedia. *CoRR*, abs/1008.1986, 2010.

[65] Jieyu Zhao, Tianlu Wang, Mark Yatskar, Vicente Ordonez, and Kai-Wei Chang. Men also like shopping: Reducing gender bias amplification using corpus-level constraints. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, pages 2979–2989, 2017.

[66] R. Zhao and K. Mao. Fuzzy bag-of-words model for document representation. *IEEE Transactions on Fuzzy Systems*, 26(2):794–804, April 2018.

[67] GuoDong Zhou and Jian Su. Named entity recognition using an hmm-based chunk tagger. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 473–480, 2002.

Appendices

Appendix A

A.1 List of words which don't appear in any lexicon

biotechnology
plm
buddhism
wrocaw
hostagetakers

reusable
intifada
rankings
terry
rivieri
attracts
maintains
nonetheless
migrations
myths
gravelly
outlandish
murdered
martyred
yaser
convetions
baladim
conception
progovernment
afflicted
lectured
poisons
invaded
tar
remarked
worldrenowned

bestselling
techniques
thinkers
militias
kept
pirated
starting
technicalities
honorouble
intervening
jihadis
smoothing
intimidated
counterprotested
sits
naziparty
spun
occurs
remixed
battalion
attacking
terrrorists
upstart
discredited
sheepishly
prepartition
aghuank
condoms
athlete
hijacked
equally
conceived

zoophile
manuski
instigated
sindhia
impacted
uncritically
quotes
farright
kaffirbashing
dolphins
euphemism
cleansed
delivers
indianoccupied
contradicts
croatiaslavonia
notionally
exofficers
revealed
weu
baptised
phenomenel
scattering
saved
exprostitutes
crowds
manisaspor
preplanned
handpainted
harvard
thugs
breached

disciplinary
asserted
nonscience
holds
babies
copyprotection
tunes
gaia
tormentors
pinko
nationals
universit
esb
nonissues
investment
affirms
aids
koreanjapanese
rouges
contradicted
stalinist
hijackers
courtbattle
declares
claiming
quebecers
pioneered
villa
respected
shortterm
seats
mwalimu
survives
speculated
rigidly
proselytism
panned
rergions
decalared
confirms
opined
terrorists
vardariotes
lifesaving
revered
evangelists
longdispute
roots
invitationonly
sterile
perception
deforming
likoudis
wilted
chinesealigned
ashkenazic
psuedoscientific
saws
termed
ojukwu
reformist
aicte
propagandised
mistakently
explored
showing
prompting
deification
conquerors
township
feudatories
characterizes
cdc
changing
intenselylocal
liberated
encompasses
bacchus
contraversialy
antizionism
9member
bruders
arbitrarily
alienating
degenerated
awarding
handpicked
avvenire
echoed
richards
dispells
democrat
antiterrorists
asserts
irredentist
leibrecht
gurus
sic
unaccredited
terroristic
ignored
morale
troublemakers
kikwetehopes
noting
conceptualized
correcting
mountainous
antisemetic
protibet
speach
termination
neighbours
ostensibly
hypothezise
bosniaherzegovina
follows
roadpeace
marixst
dealt
nontransgendered
commanding
vying
purportedly
conquering
judicial
ravaged
bannside
situated
cultlike
egyptology
disporder
noninvasively
wellknown
promoroccan
covertly
greedily
jurisprudence

avoiding
prous
3term
fiercely
epithet
safest
sectarianism
allstate
beaten
antiamericanism
piratability
bloodless
actively
loudest
wallonians
predominance
carries
aberrational
sports
ultraradical
theniraqi
circumstances
29th
poisoning
acknowledges
2millionplus
selfreliance
adherants
abkhazian
islamists
decendent
exposes
anonymous
sordid
communistic
rumasa
dogmas
relocation
miltancy
nonconformist
juridically
occupies
holland
liquidation
replied
revised
antichoice
defied
persecutions
archetype
railroaders
epl
additional
ultranationalistic
supervision
slavbulgarian
uzbekistan
copying
fixedpoint
unstopable
3storey
oppressed
foreigners
mythological
specialist8
ultraright
waging
collects
itemizes

popband
odan
pioneering
cliam
admitting
emotive
presenter
uninitialized
strengthening
governs
seized
feigned
debunker
cameramen
computerized
contagion
tempered
recoccupying
barbarians
phillip
tribal
dodging
emphasizes
prishtinas
propoganda
dravidians
bermenschconcept
ramdev
maharaj
postings
cremated
descent
nonscientific
countered
heroes
stressing
unionist
misogynists
betterment
nietzsche
mobilization
poets
interclub
lawabiding
greekmuslim
misusing
deviates
unofficially
practicing
ubiquitous
abyssinian
survivors
commented
newschannel
exploited
guarantees
contractor
oficial
handknotted
misconduct
ideals
allegeddly
variations
misquoted
sometimesheated
resumption
zoosexuality
piratable

makkah
ends
highrating
immmigrant
djs
counterattack
violating
ubisoft
munich
unbased
stormtroopers
attests
defenseless
proudly
secretly
suggestion
northamericans
routine
unece
youngster
dokdo
longcoate
propagandized
pseudochristian
bradshaw
benyovszky
involves
somosexuality
hatians
humanizing
ungodly
concieved
spans
predominantly
electrostatically
patriarch
designated
alledge
exuse
pseudocatholic