BiasFinder: Metamorphic Test Generation to Uncover Bias for Sentiment Analysis Systems

Muhammad Hilmi Asyrofi, Imam Nur Bani Yusuf, Hong Jin Kang, Ferdian Thung, Zhou Yang and David Lo

Abstract—Artificial Intelligence (AI) software systems, such as Sentiment Analysis (SA) systems, typically learn from large amounts of data that may contain human biases. Consequently, the machine learning model in such software systems may exhibit unintended demographic bias based on specific characteristics (e.g., gender, occupation, country-of-origin, etc.). Such biases manifest in an SA system when it predicts a different sentiment for similar texts that differ only in the characteristic of individuals described. Existing studies on revealing bias in SA systems rely on the production of sentences from a small set of predefined templates.

To address this limitation, we present BiasFinder, an approach to discover biased predictions in SA systems via metamorphic testing. A key feature of BiasFinder is the automatic curation of suitable templates based on the pieces of text from a large corpus, using various Natural Language Processing (NLP) techniques to identify words that describe demographic characteristics. Next, BiasFinder instantiates new text from these templates by filling in placeholders with words associated with a class of a characteristic (e.g., gender-specific words such as female names, “she”, “her”). These texts are used to tease out bias in an SA system. BiasFinder identifies a bias-uncovering test case when it detects that the SA system exhibits demographic bias for a pair of texts, i.e., it predicts a different sentiment for texts that differ only in words associated with a different class (e.g., male vs. female) of a target characteristic (e.g., gender).

Our empirical evaluation showed that BiasFinder can effectively create a large number of realistic and diverse test cases that uncover various biases in an SA system with a high true positive rate of up to 95.8%.

Index Terms—sentiment analysis, test case generation, metamorphic testing, bias, fairness bug

1 INTRODUCTION

Many modern software systems comprise of AI systems to make decisions. In AI systems, fairness is considered to be an important non-functional requirement; bias in AI systems, reflecting discriminatory behavior towards unprivileged groups, can lead to real-world harms. To address this requirement, software engineering research communities have proposed for test generation of machine learning systems. While various techniques have been used to detect bias, there have been limited studies on detecting biases in text-based machine learning systems.

Text-based ML systems have numerous applications, for example, NLP techniques have been used for Sentiment Analysis (SA). It is, therefore, important that biases in these systems can be detected before these systems are deployed.

SA systems are used to measure the attitudes and affects in text reviews about an entity, such as a movie or a news article. In this work, we focus on uncovering bias in SA for two reasons.

Firstly, SA has widespread adoption in many domains, including politics, finance, business, education, and healthcare. In the research community, SA continues to be widely studied. In the industry, many companies, such as Microsoft and Google, have developed and provided APIs for software developers to access SA capabilities. This suggests the prevalence of SA in real-life applications.

Secondly, SA has generalizability to other areas of NLP. Some NLP researchers have considered SA to be “mini-NLP”, as research on SA techniques builds on top of a wide range of topics and tasks in the NLP domain.

Modern SA models have outstanding performance on benchmark datasets, which demonstrates their effectiveness. However, there has been a growing understanding in both the Software Engineering and Artificial Intelligence research communities that it is important to study non-functional requirements, such as fairness, which have been overlooked. AI systems learn from data generated by humans. In the case of SA, the training data is typically a dataset of human-written reviews. The training data may reflect human biases. SA systems may, therefore, exhibit biases towards a demographic characteristic, such as gender.

For example, the sentiment predicted by an SA system may differ for a piece of text after a perturbation in the text to replace words that describe a demographic characteristic, e.g., changing “I am an Asian man” into “I am a black woman” may cause a predicted sentiment to change from

---

* M.H. Asyrofi, I.N.B. Yusuf, H.J. Kang, F. Thung, Z. Yang, D. Lo are with the School of Computing and Information Systems, Singapore Management University

E-mail: mhilmia@smu.edu.sg, imamy.2020@phdcs.smu.edu.sg, hjkang.2018@phdcs.smu.edu.sg, ferdianthung@smu.edu.sg, zyang@smu.edu.sg, davidlo@smu.edu.sg

Manuscript received April 19, 2005; revised August 26, 2015.
positive to negative, therefore, showing that the SA system reflects demographic bias.

As SA systems are used in many domains, including sensitive areas such as healthcare, and may be used for business analytics to make critical business decisions, it is important to detect biases in these systems. Early discovery of these biases will help to prevent the perpetuation of human biases, and aid to prevent real-world harms. To do so, SA systems should be tested for fairness (i.e., absence of unintended bias), as existing studies suggest [5], [30]. Prior studies have relied on a small number of templates to generate short texts that may uncover bias. Specifically for SA systems, Kiritchenko and Mohammad [30] propose EEC, which generates test cases produced from 11 handcrafted templates. These test cases help to detect if an SA system predicts a different sentiment given two texts that differ only in a single word associated with a different gender or race.

These test cases are limited in number and may not adequately uncover biases in a system. Very recently, SA researchers [9] have noted that “the templates utilized to create the examples might be too simplistic” and identifying such biases “might be relatively easy”. They suggest that “Future work should design more complex cases that cover a wider range of scenarios.” In this work, our goal is to address these limitations of handcrafted templates by automatically generating test cases to uncover biases.

We propose BiasFinder, a framework that automatically generates test cases to discover biased predictions in SA systems. BiasFinder automatically identifies and curates suitable texts in a large corpus of reviews, and transforms these texts into templates. Each template can be used to produce a large number of mutant texts, by filling in placeholders with concrete values associated with a class (e.g., male vs. female) given a demographic characteristic (e.g., gender). Using these mutant texts, BiasFinder then runs the SA system under test, checking if it predicts the same sentiment for two mutants associated with a different class (e.g. male vs. female) of the given characteristic (e.g. gender). A pair of such mutants are related through a metamorphic relation where they share the same predicted sentiment from a fair SA system.

The key feature of BiasFinder is its automatic identification and transformation of suitable text in a corpus to a template. This allows BiasFinder to produce a large number of test cases that are varied and realistic compared to previous approaches. Identifying suitable texts to transform to a template is challenging. For instance, all references to an entity should be replaced in a consistent way that does not make the text (e.g., a paragraph) incoherent. An example is shown in Figure 1, in which all expressions referring to an entity (“Jake”) need to be updated. The name “Jake” and its references (bolded and underlined) need to be updated together for the text to remain coherent. BiasFinder addresses this challenge through the use of Natural Language Processing (NLP) techniques, such as coreference resolution and named entity recognition, to find all words that require modification.

Our framework, BiasFinder, can be instantiated to identify different kinds of bias. In this work, we show how BiasFinder can be instantiated to uncover bias in three different demographic characteristics: gender, occupation, and country-of-origin. We empirically evaluate the 3 instances of BiasFinder by running it on a SA model based on BERT [32] – a state-of-the-art text analysis engine. By running BiasFinder on an IMDB Dataset of 50K Movie Reviews [33], we produce test cases of texts (paragraphs) resembling movie reviews. We show the effectiveness of BiasFinder in uncovering biases in an SA system; BiasFinder can generate many pairs of texts revealing biases exhibited by the SA system. Additionally, we evaluate whether the pairs of texts are coherent and should have the same sentiment (although predicted to be of different sentiments by the SA system) by performing a user study. We find that BiasFinder’s achieves a reasonable true positive rate of 89%. The contributions of our work are:

- We propose BiasFinder, a framework that uncovers bias in SA systems through the automatic generation of a large number of realistic test cases given a target characteristic. The source code of BiasFinder is publicly available.
- BiasFinder automatically identifies and curates appropriate and realistic texts (of various complexity) and transforms them into templates that can be instantiated to detect bias. Prior work only considers a small set of manually-crafted simple templates.
- We evaluate BiasFinder on IMDB Dataset of 50K Movie Reviews [33], and generate 129,598 bias-uncovering test cases over 3 demographic characteristics, gender, occupation, and country-of-origin.

The rest of this paper is organized as follows. Section 2 introduces the necessary background related to our work. Section 3 presents BiasFinder. Section 4 elaborates GenderBiasFinder, an instantiation of BiasFinder to detect gender bias. Section 5 briefly discusses instantiations of BiasFinder for detecting occupation and country-of-origin bias. Section 6 describes the results of our experiments. Section 7 presents related work. Finally, Section 8 concludes this paper and describes some future work.

2 PRELIMINARIES

This section provides more details of the benchmark dataset that motivates our work (Section 2.1), as well as basic NLP operations that we use as building blocks of our proposed approach (Section 2.2).

2.1 Equity Evaluation Corpus (EEC)
The Equity Evaluation Corpus (EEC) is a benchmark dataset proposed by Kiritchenko and Mohammad [30] to reveal

---

Fig. 1. An example of how all references to the same entity has to be considered when mutating a text to be associated with a different gender.
bias in natural language systems. The EEC consists of 8,640 sentences designed to reveal gender and race bias. These sentences are constructed instantiating placeholders in the templates shown in Table 1. The placeholders in templates 1-7 can be replaced with words to produce sentences that lean towards positive or negative sentiment, while the templates in 8-11 result in sentences with a neutral sentiment.

<table>
<thead>
<tr>
<th>No</th>
<th>Template</th>
<th># Sentence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(person) feels (emotion)</td>
<td>1,200</td>
</tr>
<tr>
<td>2</td>
<td>The situation makes (person) feel (emotion)</td>
<td>1,200</td>
</tr>
<tr>
<td>3</td>
<td>I made (person) feel (emotion)</td>
<td>1,200</td>
</tr>
<tr>
<td>4</td>
<td>(person) made me feel (emotion)</td>
<td>1,200</td>
</tr>
<tr>
<td>5</td>
<td>(person) found himself/herself in a/an (emotion) situation</td>
<td>1,200</td>
</tr>
<tr>
<td>6</td>
<td>(person) told us all about the recent (emotion) events</td>
<td>1,200</td>
</tr>
<tr>
<td>7</td>
<td>The conversation with (person) was (emotion)</td>
<td>1,200</td>
</tr>
<tr>
<td>8</td>
<td>I saw (person) in the market</td>
<td>60</td>
</tr>
<tr>
<td>9</td>
<td>I talked to (person) yesterday</td>
<td>60</td>
</tr>
<tr>
<td>10</td>
<td>(person) goes to the school in our neighborhood</td>
<td>60</td>
</tr>
<tr>
<td>11</td>
<td>(person) has two children</td>
<td>60</td>
</tr>
</tbody>
</table>

Templates in the EEC have two placeholders: (person) and (emotion). Mutant texts are generated by instantiating each placeholder with a predefined value. Predefined values for the placeholder (person) are:

- Common African American female or male first names;
- Common European American female or male first names; taken from Caliskan et al. [34];
- Noun phrases referring to females, such as 'my daughter'; and noun phrases referring to males, such as 'my son'.

The second placeholder, (emotion), corresponds to four basic emotions: anger, fear, joy, and sadness. For each emotion, EEC selects five words from Roget’s Thesaurus with varying intensities.

Although the EEC has successfully revealed bias in NLP systems [30], it is limited only to gender and race bias. It does not explore bias against other demographic information (e.g., occupation, etc.) that may also lead to inappropriate behavior of Sentiment Analysis and other NLP systems. Furthermore, the templates used to create the text dataset may be too short and simplistic as argued by Poria et al. [9]. We suggest that a system that has the capability to automatically create templates to produce more diverse and more complex sentences can aid in better uncovering bias in Sentiment Analysis systems.

2.2 Natural Language Processing (NLP) Techniques

2.2.1 Part-of-speech Tagging

Part-of-speech tagging (PoS-tagging) is the process of identifying the part of speech (e.g. noun, verb) that each word in a text belongs to [35]. An example of PoS-tagging is shown in Figure 2. In the example text, “Maria” is tagged as a proper noun (PROP); “has” and “loves” are tagged as verbs (VERB); and “She” and “him” are tagged as pronouns (PRON).

http://www.gutenberg.org/ebooks/22

2.2.2 Named Entity Recognition

Named entity recognition (NER) automatically identifies named entities in a text and groups them into predefined categories. Examples of named entities are people, organizations, occupations, and geographic locations [36]. An example of NER can be found in Figure 2, where the word “Maria” is assigned to the “PERSON” category. In this work, we are mainly interested with the person (for gender and country-of-origin bias) and occupation (for occupation bias) categories.

2.2.3 Coreference Resolution

Finding all expressions that refer to the same entity in a text is known as coreference resolution [37]. Linking such expressions is useful for many NLP tasks where the correct interpretation of a piece of text has to be derived (e.g. document summarization, question answering). Coreference resolution only links expressions together, and does not identify the types of the referenced entities, which is done through NER. An example of coreference resolution can be found in Figure 2, in which the expressions “Maria” and “She” are linked. Likewise, the expressions “a friend” and “him” are linked as they refer to the same entity.

Given an input text, running a coreference resolution on it will produce n lists of references; each list corresponds to references to a single entity.

![Input Text](http://www.gutenberg.org/ebooks/22)

Maria has a friend. She loves him.

**POS-tagging**

Maria|PROP|Noun|has|VERB|a|DET|friend|Noun|.|PUNCT

She|PRON|loves|VERB|him|PRON|.|PUNCT

**NER**

Maria|PERSON

**Coreferences Resolution**

Maria has a friend. She loves him.

**Coreferences**

Maria, She, a friend, him

2.2.4 Dependency Parsing

The process of assigning a grammatical structure to a piece of text and encoding dependency relationships between words is known as dependency parsing [38], [39]. Encoding such information as a parse tree, words in a text are connected such that words that modify each other are linked. For example, a dependency parse tree connects a verb to its subject and object, and a noun to its adjectives.

Figure 3 shows an example of a parse tree that is output by performing a dependency parsing of an input text: “That guy from Blade Runner also cops a good billing”. The directed, labeled edges between nodes indicate the
relationships between the parent and child nodes. From the parse tree, the root word of a phrase can be identified. For example, the root word of the phrase “That guy from Blade Runner” represented in Figure 3 is “played”, as its node does not have any incoming edges from the nodes of other words in the phrase.

3 BiasFinder

Figure 4 shows the architecture of our proposed approach: BiasFinder. It takes, as input, a collection of texts and a sentiment analysis (SA) system, and produces, as output, a set of bias-uncovering test cases. BiasFinder has three components: (A) template generation engine, (B) mutant generation engine, and (C) failure detection engine. The template generation engine generates bias-targeting templates from a collection of texts. These templates are designed to target bias towards a specific characteristic (e.g., gender). The generated templates are input to the mutant generation engine. This engine generates text variants (mutants) that differ in a target bias characteristic (e.g., two paragraphs, which are otherwise identical, but describe an individual using words associated with a different gender) and should have the same sentiment. These mutants are then input to the failure detection engine. This engine makes use of the metamorphic relation between mutants (i.e., they have the same sentiment as they are generated from the same template) to infer failures (i.e., bias). This engine identifies mutants that uncover bias in the SA system. These mutants are output as the bias-uncovering test cases.

3.1 Template Generation Engine

The template generation engine follows the workflow in Figure 5. It takes a collection of texts as the input and produces bias-targeting templates. Each template is a text unit (e.g., a paragraph) that contains one or more placeholders; the placeholders can be substituted with concrete values to generate different pieces of text that should have the same sentiment.

This engine generates templates for detecting bias in a target characteristic (e.g., gender, occupation, etc.). It extracts linguistic features such as named entities, coreferences, and part-of-speech (Step 1). Using these features, it identifies entities related to the characteristic of the targeted bias (Step 2). If such entities exist in the texts, BiasFinder replaces references to these entities with placeholders. Essentially, the texts are converted to templates which will be used to generate mutant texts for uncovering the targeted bias (Step 3).

3.2 Mutant Generation Engine

To generate mutant texts from a bias-targeting template, this engine replaces template placeholders with concrete values taken from pre-determined lists of possible values. These lists differ based on the target bias under consideration. The engine substitutes the placeholders with concrete values while ensuring that the generated mutants are valid. A mutant is valid if the values that are assigned to the placeholders are in agreement with each other. For example, we do not want to generate the following text: “The man speaks to herself”. The engine ensures this does not occur by picking only values from a single class (e.g., male-related words) to substitute related placeholders to generate a mutant. Each generated mutant is thus associated to a class; and BiasFinder’s goal is to check if an SA discriminates against one of the classes (e.g., male or female) associated with a target characteristic (e.g, gender).

3.3 Failure Detection Engine

The failure detection engine takes as input a set of mutant texts along with their class labels, and produces a set of bias-uncovering test cases. It then feeds the mutants one-by-one to the SA system, which outputs a sentiment label for each mutant. Mutants of
differing classes that are produced from the same template are expected to have the same sentiment. Therefore, if the SA predicts that two mutants of different classes to have different sentiments, they are an evidence of a biased prediction. Such pairs of mutants are output as bias-uncovering test cases.

3.4 Instantiating BiasFinder to Different Biases

BiasFinder can be instantiated in various ways to uncover different kinds of biases. In this work, we investigate 3 instances of BiasFinder that can uncover gender, occupation, and country-of-origin biases of a sentiment analysis (SA) system. To instantiate BiasFinder to a particular target characteristic, we need to customize its three components: template generation engine, mutant generation engine, and failure detection engine. We elaborate on how we create GenderBiasFinder, an instance of BiasFinder targeting gender bias in Section 4 and briefly describe the two other instances of BiasFinder in Section 5.

4 GenderBiasFinder

An SA system exhibits gender bias if it behaves differently for texts that only differ in words that reflect gender. GenderBiasFinder generates mutants by changing words associated with the gender of a person, and uncovers gender bias when the SA system predicts different sentiments for a pair of mutants of different gender classes. In this work, we focus on binary genders: male and female; but our approach can be extended and generalized for non-binary genders. To uncover gender bias, we customize the three main engines of BiasFinder: Template Generation Engine, Mutant Generation Engine, and Failure Detection Engine.

4.1 Template Generation Engine

Algorithm 1: Generating a Template for Detecting Gender Bias

| Input: s: a text, gn: gender nouns |
| Output: t: a template or null |
| t = s; |
| corefs = getCoreferences(s); |
| names = getPersonNamedEntities(s); |
| coref = filter(corefs, names, gn); |
| if coref \neq null then |
| for r \in coref do |
| if isPersonName(r, names) then |
| t = createPlaceholder(t, s, r, names); |
| end |
| else if isGenderPronoun(r) then |
| t = createPlaceholder(t, s, r); |
| end |
| else if hasGenderNoun(r, gn) then |
| t = createPlaceholder(t, s, r, gn); |
| end |
| end |
| if t \neq null then |
| return t == s?null : t |

Next, we filter coreference lists in corefs by performing two checks embedded inside function filter (Line 4): 1) There is only one list in corefs that refers to a person. In this work, we consider any of the following as a reference to a person: (i) a person name, (ii) a gender pronoun (i.e. he, she), or (iii) a phrase containing a gender noun (e.g., “that guy from Blade Runner”). 2) All references in the list identified above must be a reference to a person.

If both conditions are met, filter returns a coreference list coref satisfying the condition; otherwise, it returns null. These checks are done to avoid the generation of unsound templates due to coreference resolution’s limitations, e.g., detecting a set of references to the same entity as two disjoint lists. If there is a coref returned, GenderBiasFinder iterates all its references and creates placeholders depending on the type of each reference r (Lines 7-15). At the end of the iteration, we output a template t generated from the input text s (Line 15). For each iteration, we have three cases depending on the type of each r:

Case 1: The Reference is a Person Name (Line 7)

At line 7, GenderBiasFinder checks whether the reference r is a person’s name in the list of names names extracted using named entity recognition (see Section 2.2.2). If this is the case, GenderBiasFinder generates a template by replacing the person’s name with the (name) placeholder (Line 8). In the example shown in Figure 6 “Drew Barrymore” is a person’s name and is replaced with this placeholder.

Case 2: The Reference is a Gender Pronoun (Line 10)

GenderBiasFinder checks if the reference r is a gender pronoun (Line 10). If so, GenderBiasFinder converts the
gender pronoun into \langle pro-id \rangle (\text{Line } 11), where \text{id} can take several values according to the type of the gender pronoun that the placeholder replaces: (1) spp for subjective personal pronoun (i.e., he and she), (2) opp for objective personal pronoun (i.e., him and her), (3) pp for possessive pronoun (i.e., his and her), and (4) rp for reflexive pronoun (i.e., himself and herself). In the example shown in Figure 6, “she” is converted to \langle pro-spp \rangle placeholder, while “her” is converted to \langle pro-pp \rangle placeholder.

Case 3: The Reference has a Gender Noun (Lines 13, 15)

GenderBiasFinder checks if the root word of the reference \text{r} is a gender noun (Line 15). GenderBiasFinder utilizes dependency parsing (see Section 2.2.4) to find the root word and performs POS-tagging (see Section 2.2.1) to confirm that the root word is a noun. Next, it checks that the word exists in \text{gn}, a collection of gender-related nouns, and if it does, converts the root word to \langle gaw \rangle placeholder (Line 16). In the example shown in Figure 7, the reference is “That guy from “Blade Runner””. By performing dependency parsing and POS-tagging, “guy” is identified as the root word and is a noun. GenderBiasFinder checks whether “guy” exists in \text{gn}. As it does, GenderBiasFinder replaces “guy” to a \langle gaw \rangle placeholder. Some examples of gender nouns are shown in Table 2. In total, we use 22 gender nouns.

<table>
<thead>
<tr>
<th>Name</th>
<th>Gender</th>
<th>Country-of-origin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrosov</td>
<td>Male</td>
<td>Russia</td>
</tr>
<tr>
<td>Kapoor</td>
<td>Female</td>
<td>India</td>
</tr>
<tr>
<td>Andrea</td>
<td>Male</td>
<td>Italy</td>
</tr>
<tr>
<td>Zeynep</td>
<td>Female</td>
<td>Turkey</td>
</tr>
</tbody>
</table>

TABLE 2
Example Names from GenderComputer

TABLE 3
Example of Gender Nouns

<table>
<thead>
<tr>
<th>Male</th>
<th>Female</th>
</tr>
</thead>
<tbody>
<tr>
<td>boy, brother, father, dad, …</td>
<td>girl, sister, mother, mom, …</td>
</tr>
</tbody>
</table>

of the values. Each placeholder can be substituted by a value from a set. We describe the values that each placeholder can be substituted with below:

\langle name \rangle Placeholder: Values to be substituted for this placeholder are taken from the set of names from GenderComputer. GenderComputer provides a database of male and female names from several countries. Each name in the GenderComputer provides information about its gender and its country-of-origin. It is possible that a name may be used by both genders in the same or different countries. Thus, we filter the names to make sure that the selected names are only used for one gender globally. We randomly take \text{2N} names from this filtered set: \text{N} male names and \text{N} female names. By default, \text{N} is set to \text{30}. Examples of names from GenderComputer are shown in Table 2.

\langle pro-id \rangle Placeholder: Values to be substituted for this placeholder depend on the gender class of the mutant and the \text{id}. For male mutant, the values are he for \langle id-spp \rangle, him for \langle id-opp \rangle, his for \langle id-pp \rangle, and himself for \langle id-rp \rangle. For female mutant, the values are she for \langle id-spp \rangle, her for \langle id-opp \rangle, her for \langle id-pp \rangle, and herself for \langle id-rp \rangle.

\langle gaw \rangle Placeholder: Values to be substituted for this placeholder are the set of gender nouns taken from several English resources. Examples of these gender nouns are shown in Table 3.

4.3 Failure Detection Engine

The Failure Detection Engine runs the SA system, using the generated mutants as inputs. It receives, from the SA system, a label for each mutant indicating the predicted sentiment of the mutant. Mutants generated from the same template are expected to have the same predicted sentiment and are grouped together. Each group of mutants is further divided into two classes, depending on the gender associated with the mutant. Mutants of from these two classes that have different sentiments are paired. In other words, the engine find pairs of mutants generated from the same template that differ in both the gender class they are associated with, and the sentiment predicted by the SA system. These pairs of mutants are the bias-uncovering test cases and are the output of GenderBiasFinder.

https://github.com/tue-mdse/genderComputer
http://www.primaryresources.co.uk/english/PC_gen.htm

4.2 Mutant Generation Engine

For each generated template, the mutant generation engine produces multiple mutants by replacing placeholders with concrete values. As our objective in GenderBiasFinder is to create test cases related to gender, each mutant is associated with a gender class (i.e., male or female) and the mutant generation engine is restricted to values associated with the given gender class when filling in all placeholders for one mutant. The engine iterates over all possible combinations

Fig. 7. An Illustrative Example for Case 3 of GenderBiasFinder

POS-tagging of The Reference
That|DET guy|NOUN from|ADP “|PUNCT Blade|PROPN Runner|PROPN “|PUNCT

Generated Template
Even the manic loony who hangs out with the bad guys in “Mad Max” is there. That guy from “Blade Runner” also cops a good billing, although he only turns up at the beginning and the end of the movie.

Coreferences
That guy from “Blade Runner”, he

Dependency Parsing of The Reference
That|DET guy|NOUN from|ADP “|PUNCT Blade|PROPN Runner|PROPN “|PUNCT

Text
Even the manic loony who hangs out with the bad guys in “Mad Max” is there. That guy from “Blade Runner” also cops a good billing, although he only turns up at the beginning and the end of the movie.
5 OTHER INSTANCES OF BiasFinder

In this section, we describe how BiasFinder can be instantiated for occupation and country-of-origin biases.

5.1 Occupation Bias

Occupation bias occurs when an SA system favors an honest (i.e., non-criminal) occupation over another. It can be detected when the SA system produces different sentiment for a pair of mutants that differ only on the occupation referred in the text. We perform these customizations to uncover occupation bias:

Template Generation Engine: We generate occupation templates following Algorithm 2. We first extract the list of occupations occs mentioned in the input text s using named entity recognition (Line 2). We then iterate each occupation occ from occs (Line 3). We then confirm that occ is a noun and check whether occ has adjectives (Lines 4-5). For example, the adjective of "driver" in the "race car driver" noun phrase is "race car". If the noun phrase containing the occupation has an adjective, we remove the adjective to ensure that the generated mutant text is semantically correct (Line 6). Leaving the adjective intact may produce a text that describes a non-existent occupation such as "race car secretary". We then convert occ to (occupation) placeholder (Line 8). We also convert determiner "a" or "an" in front of occ (if it exists) to (det) placeholder to ensure the produced mutant template is grammatically correct. Next, we extract occCorefs (Line 9); occCorefs is the list of references in s that refer to the same entity that occ refers to. We then iterate each reference r from occCorefs (Line 10). We check whether r is a mention of occ (Line 11). For such r, we again create (occupation) and (det) placeholders (if necessary), after removing adjectives (if necessary) (Lines 12-15). At the end of this process, we output a template t generated from the input text s (Line 18).

In the example below, we detect "doctor" and "journalist" as occupations. We only use the first occupation to form a template. As "doctor" is a noun, and it is not preceded by any adjective, we replace it directly to (occupation) placeholder. We then replace its determiner with (det) placeholder. In this case, there are no coreferences of "doctor", so the template generation process ends.

```
Text
The beautiful Jennifer Jones looks the part and gives a wonderful, Oscar nominated performance as a doctor of mixed breed during the advent of Communism in mainland China.

Occupation Named Entity
doctor

Generated Template
The beautiful Jennifer Jones looks the part and gives a wonderful, Oscar nominated performance as (det) (occupation) of mixed breed during the advent of Communism in mainland China.
```

Fig. 8. An Illustrative Example for OccupationBiasFinder

Mutant Generation Engine: To generate occupation mutants, the engine substitutes the (occupation) placeholder with a value from a set of 79 honest (i.e., non-criminal) and gender-neutral occupation names that are taken from [40–42]. The value of (det) is linked with the value of (occupation) placeholder. For example, the values of (det) for "teacher" and "engineer" occupations are "a" and "an", respectively.

Failure Detection Engine: The engine inputs the generated mutants to the SA system. The SA system labels each mutant with a predicted sentiment. Mutants from the same template are grouped together and mutants in the same group that have a different sentiment are paired. By doing so, the engine finds pairs of mutants that differ both in the occupation they mentioned and the sentiment predicted by the SA system. These pairs of mutants are the bias-uncovering test cases for occupation bias.

5.2 Country-of-Origin Bias

Country-of-origin bias occurs when the SA system favors a person who originates from one country over a person originating from another country. This bias is detected when the SA system produces different sentiments for texts differing only in country-of-origin of the person referred in the text. To uncover country-of-origin bias, we customize BiasFinder as follows:

Template Generation Engine: For generating country-of-origin templates, we follow Algorithm 3. We first run coreference resolution to find corefs, which contains references of persons mentioned in the input text s (Line 1). We also run named entity recognition to extract the list of person names mentioned in s (Line 2), which we refer to as names.

Next, we filter coreference lists in corefs by using the same filter function described in Algorithm 1 in Section 4. This is done to avoid the generation of unsound templates due to coreference resolution’s limitations. The filter function
returns either a coreference list coref or null. We stop the template generation process if null is returned.

Otherwise, if the references in coref refer to a consistent gender g (Line 5) – i.e., by checking that there is a gender pronoun in coref and all gender pronouns in it are of the same gender (e.g., he, him, his, himself for male gender) – we iterate each reference r in coref (Lines 7–11). If r is the person name in names, we replace r with a placeholder representing the gender that was detected (Lines 8–10). A ⟨male⟩ or ⟨female⟩ placeholder is created if a male or a female gender was detected, respectively.

In the example below, “Lauren Holly” is detected as a person name and the coreferences consistently refer to female gender. Thus, we replace “Lauren Holly” with ⟨female⟩ placeholder.

Input: s: a text
Output: t: a template or null
1. t = s; corefs = getCoreferences(s);
2. names = getPersonNamedEntities(s);
3. coref = filter(corefs, names);
4. if coref ≠ null then
5. g = inferGender(coref);
6. if g ∈ {Male, Female} then
7. for r ∈ coref do
9. if isPersonName(r, names) then
10. t = createPlaceholder(t, s, r, g);
11. end
12. end
13. return t
14. end
15. return null

Fig. 9. An Example of Text in CountryBiasFinder

Mutant Generation Engine: To generate country-of-origin mutants, the engine substitutes ⟨male⟩ and ⟨female⟩ placeholders with values from a set of people names taken from GenderComputer. GenderComputer provides the country-of-origin and the gender of each name. Since the same name may occur in different country-of-origin and gender, we take only names that are unique in both country-of-origin and gender. We pick only a male name and a female name from each country. In total, we have 52 names taken from 26 countries of origin. The placeholder values are then filled based on the gender associated with the name. Male and female names are used to fill ⟨male⟩ and ⟨female⟩ placeholders, respectively.

Failure Detection Engine: The engine accepts the generated mutants as input and feed them to the SA system, which gives a sentiment label for each mutant. Mutants from the same template that have a different sentiment are then paired. Here, the engine finds pairs of mutants that differ both in the country-of-origin of the person they mentioned and the sentiment predicted by the SA system. These pairs of mutants are the bias-uncovering test cases for country-of-origin bias.

6 Experiments

In this section, we describe our dataset, experimental settings, evaluation metric, and our research questions. Next, we answer the research questions, and mention threats to validity.

6.1 Dataset and Experimental Settings

We focus on a binary sentiment analysis task, i.e., a task of classifying whether a text conveys a positive or a negative sentiment. A popular dataset to evaluate a sentiment analysis system’s performance for this task is the IMDB Dataset of 50K Movie Reviews. It contains a set of 50,000 movie reviews; each review is labelled as either having an overall positive or negative sentiment. Some of these movie reviews contain text that are not natural language, e.g., HTML tags. We remove these text from the movie reviews. Then, we split the 50,000 movie reviews evenly to train and test sets.

We use a Transformer-based model as the sentiment analysis system in our experiments. Transformer-based models have achieved state-of-the-art performances on many NLP tasks in recent years [26, 43, 44]. We pick BERT as a representative model. It is not our goal to evaluate and compare bias over all sentiment analysis systems – we leave such extensive comparative evaluation for future work.

BERT is a pre-trained language model that can be fine-tuned to solve many downstream NLP tasks. We fine-tuned BERT for binary sentiment analysis following Sun et al. work [27] and used the implementation they provided. We fine-tuned the BERT sentiment analysis (SA) model using the train set and use the test set as input for three instances of BiasFinder to generate mutants.

We performed our experiments on a computer running Ubuntu 18.04 with Intel(R) Core(TM) i7-9700K CPU @ 3.60GHz processor, 64GB RAM, and NVIDIA GeForce RTX 2080. For coreference resolution, we use NeuralCoref. We use both SpaCy and Stanford CoreNLP for Part-of-Speech (PoS) Tagging and Named Entity Recognition (NER).

Our objective in the study is to produce test cases that reveal bias. As earlier defined, a bias-uncovering test case is a pair of mutants from a bias-targeting template that should
have the same sentiment, but are labelled with a different sentiment by the SA system. To validate our results, we also performed a user study to ensure that the test cases produced are coherent pieces of text and that the mutants of the same template should, indeed, have the same sentiment label.

6.2 Research Questions

**RQ1.** How many BTCs can BiasFinder generate?
BiasFinder is the first approach to automatically generate templates for uncovering multiple biases. We report the number of generated BTCs produced by each instance of BiasFinder for the BERT-based SA system.

**RQ2.** How many BTCs are true positives?
We count the number of BTCs that are true positive via a user study. The user study involves two participants, both of whom are native English speakers and are not authors of this paper. The participants were asked to label mutant texts from randomly sampled BTCs for the three characteristics. We computed the number of samples required for a statistically representative sample given a margin of error of 5% and 95% confidence for the three populations (of 14,916, 109,386, 5,296 generated BTCs). Therefore, we sampled 400 BTCs for each characteristic, which is a number greater than the representative sample sizes (375, 383, 358). The users were asked to label if the mutant texts involved in each BTC are coherent and to provide sentiment labels. We consider a BTC a true positive if its mutants are labelled as coherent texts and are of the same sentiment.

6.3 Results

**RQ1.** How many BTCs can BiasFinder generate?

Table 4 shows the number of templates generated by and BTCs found by BiasFinder for each instance. BiasFinder discovered a total of 129,598 BTCs across the three characteristics. Among the three characteristics, it generated the highest number of BTCs for occupation bias and the fewest number of BTCs for the country-of-origin bias. Overall, our experimental results suggest that bias of each characteristic occurs in the target SA system. BiasFinder can also find more BTCs for each bias by generating more templates from another corpus. The templates range in their complexity depending on the text in the corpus. Figures 10, 11, 12 show examples of bias-uncovering test cases for gender, occupation, and country-of-origin, respectively.

<table>
<thead>
<tr>
<th>Target Bias</th>
<th># Templates</th>
<th># BTCs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gender</td>
<td>3,057</td>
<td>14,916</td>
</tr>
<tr>
<td>Occupation</td>
<td>8,867</td>
<td>109,386</td>
</tr>
<tr>
<td>Country-of-Origin</td>
<td>2,344</td>
<td>5,296</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>14,268</strong></td>
<td><strong>129,598</strong></td>
</tr>
</tbody>
</table>

Table 4 shows the number of templates generated by and BTCs found by BiasFinder for each instance. BiasFinder discovered a total of 129,598 BTCs across the three characteristics. Among the three characteristics, it generated the highest number of BTCs for occupation bias and the fewest number of BTCs for the country-of-origin bias. Overall, our experimental results suggest that bias of each characteristic occurs in the target SA system. BiasFinder can also find more BTCs for each bias by generating more templates from another corpus. The templates range in their complexity depending on the text in the corpus. Figures 10, 11, 12 show examples of bias-uncovering test cases for gender, occupation, and country-of-origin, respectively.

**RQ2.** How many BTCs are true positives?

We also performed a user study to ensure that the test cases produced are coherent pieces of text and that the mutants of the same template should, indeed, have the same sentiment label.

6.4 Threats to Validity

According to Landis and Koch [48], a Kappa value between 0.40 to 0.6 is moderate agreement, while 0.6 to 0.8 is substantial agreement, and a value above 0.8 represents almost perfect agreement.
Mutant Text - Male Name from Somalia
I consider this movie as one of the most interesting and funny movies of all time ( ... ) Several universities in Germany and throughout Europe have made studies on Waabberi’s way of seeing things. By the way, Waabberi is a very intelligent and sensitive person and of the Jazz musicians in Germany

Mutant Text - Male Name from Iran
I consider this movie as one of the most interesting and funny movies of all time ( ... ) Several universities in Germany and throughout Europe have made studies on Keyghobad’s way of seeing things. By the way, Keyghobad is a very intelligent and sensitive person and of the Jazz musicians in Germany

Fig. 12. BTC Example for Country-of-Origin. (...) is a truncated piece of the original text.

Original Text
I believe the story felt very plain because the director failed to focus on character development

Mutant Text
I believe the story felt very plain because the banker failed to focus on character development

Fig. 13. Example of a false positive. The usage of the “director” word is context specific, and cannot be replaced with another occupation to produce a coherent mutant text.

We have only experimented with an SA system based on BERT and generated templates from the IMDB dataset. The results may not be generalize to other SA systems and datasets. However, BERT is among the top performing models for text classification in recent years [26, 32, 43, 44], and the IMDB dataset is a common dataset used for studying sentiment analysis [28, 49, 50]. Thus, we believe the threats to validity are minimal as we show that the biases occur on mutants of texts from a common dataset and the biases are exhibited by one of the top models.

While we investigated only the IMDB dataset, it is a large dataset and is commonly used to evaluate sentiment analysis techniques. Thus, we believe IMDB is a good representative dataset for investigating biases in sentiment analysis. Furthermore, the IMDB dataset contains high polarity reviews (i.e., reviews with strong positive/negative sentiments) and, as such, there is minimal risk of mislabelling them.

7 Related Work

In this section, we first describe related work on understanding and detecting bias in AI systems (Section 7.1). Next, we describe some of the related work testing AI systems (Section 7.2).

7.1 Bias in AI Systems

The importance of studying bias in AI systems has been described by many researchers [1, 2, 30, 51, 52]. An AI system may perpetuate human biases and perform differently for some demographic groups than others [1]. As such, many existing studies on uncovering bias [1, 2, 30] focus on finding differences in the system’s behavior given a change in a demographic characteristic (aka. attribute). Our approach has the same high-level objective of uncovering differences in behavior when demographic characteristic is modified, however, our approach differs in several ways, which will be described in the following paragraphs.

Themis [1], Aeqitas [2], and FairTest [3] are approaches aiming to generate test cases that detect discrimination in software. Fairway [4] mitigates bias through several strategies, including identifying and removing ethical bias from a model’s training data. Unlike our approach, these strategies do not target NLP systems but focus on systems that take numerical values or images as input, while BiasFinder targets Sentiment Analysis systems which take natural language text as input.

Specific to NLP applications, CheckList [5] has been proposed for creating test cases to evaluate systems on their capabilities beyond their accuracies on test datasets. Fairness is among the capabilities that CheckList tests for, and CheckList relies on a small number of predefined templates for producing test sentences. Our work is complementary to this approach as it can be used to produce test cases without the restriction of predefined templates.

For Sentiment Analysis systems, the EEC [30] has been proposed to uncover bias by detecting differences in predictions of text differing in a single word associated with gender or race. However, as described earlier in Section 2, other researchers [9] have pointed out that the EEC [30] relies on predefined templates that may be too simplistic. We address this limitation as our approach dynamically generates many templates to produce sentences that are varied and realistic. Moreover, our approach uncovers bias through mutating words in text associated with characteristics other than gender and race.

7.2 AI Testing

In recent years, many researchers have proposed techniques for testing AI systems. There are too many of them to mention here. Still, we would like to highlight a few, especially those that are closer to our work. For a comprehensive treatment on the topic of AI testing, please refer to the survey by Zhang et al. [53].

Existing studies have applied metamorphic testing to AI systems [54–57]. Many of these systems focus on finding bugs, for example, in machine translation [54, 57] or autonomous driving systems [55, 56]. Our work is related to these studies as BiasFinder is based on metamorphic testing, but differs in that we focus on finding fairness bugs (gender, occupation, and country-of-origin bias) in Sentiment Analysis systems.

In the NLP domain, some research efforts have developed methods for generating adversarial examples [58, 59], while other researchers have proposed techniques to test robustness to typos and other forms of noise [60], or changes in the names of people mentioned in text [61]. Our work differs from these studies as it focuses on uncovering bias rather than testing the correctness of an NLP system.
8 CONCLUSION AND FUTURE WORK

There is growing use of Artificial Intelligence in software systems, and fairness is an important requirement in Artificial Intelligence systems. Testing is one way to uncover unintended biases [4], [5]. Our research contributes to the body of work on fairness testing and motivates future research to build automatic fairness testing methods for various machine learning tasks, including sentiment analysis (that we consider in this work).

We propose BiasFinder, a metamorphic testing framework for creating test cases to uncover demographic biases in Sentiment Analysis (SA) systems. BiasFinder can be instantiated for different demographic characteristics, such as gender or occupation. Given a target characteristic, BiasFinder curates suitable texts from a corpus to create bias-uncovering templates. From these templates, BiasFinder then produces mutated texts (mutants) that differ only in words associated with different classes (e.g., male vs. female) of the target characteristic (e.g., gender). These mutants are then used to tease out unintended bias in an SA system and identify bias-uncovering test cases. By analyzing a realistic and diverse corpus, BiasFinder can produce realistic and diverse bias-uncovering test cases.

Existing work [30] use only a small set of simple test cases, testing only for ethnicity bias by swapping names of two geographical characteristics, i.e. African American names and European American names, in a small set of templates of simple sentences. Our work generates templates of test cases involving other characteristics, including gender and country-of-origin. Our mutant generation constructs test cases representing names from 30 countries. Together, the template and mutation generation produces test cases that cover a wider range of scenarios.

We empirically evaluated three instances of BiasFinder on an SA model based on BERT. BiasFinder identifies 14,916, 109,386, and 5,296 bias-uncovering test cases for gender, occupation, and country-of-origin bias. Through a user study, we find that the true positive rates of BiasFinder are reasonably high. In other words, BiasFinder produces a high percentage of bias-revealing pairs of texts that a human annotator considers to have the same sentiment and are coherent.

In the future, we plan to instantiate BiasFinder on more biases and expand the experiments (e.g., by considering other text corpora). Moreover, we will evaluate BiasFinder to determine if it generalizes to tasks beyond SA, for example, testing general text classifiers.

References


