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

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

Abstract—Artificial intelligence systems, such as Sentiment Analysis (SA) systems, typically learn from large amounts of data that may reflect human bias. Consequently, such systems may exhibit unintended demographic bias against specific characteristics (e.g., gender, occupation, country-of-origin, etc.). Such bias manifests in an SA system when it predicts different sentiments for similar texts that differ only in the characteristic of individuals described. To automatically uncover bias in SA systems, this paper presents BiasFinder, an approach that can discover biased predictions in SA systems via metamorphic testing. A key feature of BiasFinder is the automatic curation of suitable templates from any given text inputs, using various Natural Language Processing (NLP) techniques to identify words that describe demographic characteristics. Next, BiasFinder generates new texts from these templates by mutating words associated with a class of a characteristic (e.g., gender-specific words such as female names, "she", "her"). These texts are then used to tease out bias in an SA system. BiasFinder identifies a bias-uncovering test case (BTC) when an SA system predicts different sentiments for texts that differ only in words associated with a different class (e.g., male vs. female) of a target characteristic (e.g., gender). We evaluate BiasFinder on 10 SA systems and 2 large scale datasets, and the results show that BiasFinder can create more BTCs than two popular baselines. We also conduct an annotation study and find that human annotators consistently think that test cases generated by BiasFinder are more fluent than the two baselines.

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

1 Introduction

Many modern software systems employ artificial intelligence (AI) to make decisions. In AI systems, fairness is considered to be an important non-functional requirement as bias in AI systems, reflecting discriminatory behavior towards unprivileged groups, can lead to real-world harms. To address this requirement, software engineering research techniques, such as test case generation, have been applied to detect bias [1]–[5]. This paper investigates sentiment analysis (SA) systems that can be used to measure the attitudes and affects in text reviews about an entity, such as a movie or a news article [6], [7]. We focus on uncovering bias in SA systems for three reasons:

Firstly, SA has widespread adoption in many domains [8], [9], including politics [10], [11], finance [12]–[15], business [16], education [17]–[19], and healthcare [20]–[22]. In the research community, SA continues to be widely studied [23]–[28]. In the industry, many companies, such as Microsoft[1] and Google[2], have developed and provided APIs for software developers to access SA capabilities. This suggests the prevalence of SA in real-life applications. As a result, bias in SA systems can have a big impact on society.

Secondly, SA has generalizability to other areas of NLP. Some researchers have considered SA to be "mini-NLP" [9], as research on SA techniques builds on top of a wide range of topics and tasks in the NLP domain. Cambria et al. [29] argue that SA is a problem with a composite nature, requiring 15 more fundamental NLP problems to be addressed at the same time. Therefore, we believe that tackling bias in SA is a suitable first step that could lead to a more general approach to detect bias in textual data.

Thirdly, due to the importance of SA systems, there are many recent research works [5], [30]–[34] that focus solely on the fairness issues in SA systems. Although these works are not guaranteed to be fully generalizable to all kinds of NLP systems, the importance and wide applicability of SA justify the need of fairness studies that focus on it.

Modern SA systems have outstanding performance on benchmark datasets, which demonstrates their effectiveness. In the case of SA, the training data is typically a dataset of human-written texts that may reflect human bias. SA systems may, therefore, exhibit bias towards a demographic characteristic, such as gender [30], [32]. 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 positive to negative, therefore, showing that the SA system reflects demographic bias.

Early discovery of bias in SA systems will help to prevent the perpetuation of human bias, and aid to prevent real-world harms. As a result, existing studies suggest [5], [30] that SA systems should be tested for fairness (i.e., exposing unintended bias). Researchers define the *metamorphic relationship* that a fair SA system should have and utilize

• *M.H. Asyrofi, Z. Yang, I.N.B. Yusuf, H.J. Kang, F. Thung, D. Lo are with the School of Computing and Information Systems, Singapore Management University*
E-mail: mhilmia@smu.edu.sg, zyang@smu.edu.sg, imamy.2020@phdcs.smu.edu.sg, hjkang.2018@phdcs.smu.edu.sg, ferdianthung@smu.edu.sg, davidlo@smu.edu.sg

Manuscript received April 19, 2005; revised August 26, 2015.

[1]https://azure.microsoft.com/en-us/services/cognitive-services/text-analytics/

[2]https://cloud.google.com/natural-language/docs/analyzing-sentiment

metamorphic testing to uncover bias. More specifically, a fair SA system should produce the same result for two pieces of texts that only differ in sensitive features (e.g., demographic characteristics like gender). Prior studies typically rely on a small number of templates to generate short texts and check whether the above metamorphic relationship is satisfied. For example, 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.

However, these test cases are limited in number and may not adequately uncover bias in SA systems. Very recently, SA researchers [9] have noted that "the templates utilized to create the examples might be too simplistic" and identifying such bias "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 templates that can be used to uncover bias.

In this paper, we present BiasFinder, a framework that automatically generates templates and test cases to discover biased predictions in SA systems. BiasFinder automatically identifies the words associated to demographic characteristics in given texts and then replaces identified words with placeholders to transform these texts into templates. Each template can be used to produce a large number of text mutants, 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 the metamorphic relationship is satisfied (i.e., an SA system predicts the same sentiment for two mutants with different demographic characteristics.)

The key feature of BiasFinder is its automatic identification and transformation of suitable texts in a corpus to templates, which allows BiasFinder to produce a large number of test cases that are varied and realistic compared to previous approaches [30]. Identifying the suitable texts to be transformed to templates 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) not fluent. 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 fluent. 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 uncover bias in three different demographic characteristics: gender, occupation, and country-of-origin. We obtained 10 SA models by fine-tuning 5 Transformer-based models on two popular sentiment analysis datasets: the IMDB movie review and Twitter Sentiment140 (the IMDB and Twitter datasets for short in the following parts of the paper). We compare BiasFinder with two baselines (EEC [30] and MT-NLP [31]) on the two datasets. We evaluate the effectiveness of BiasFinder in uncovering bias by measuring the number of bias-uncovering test cases (BTCs) found. A BTC is a pair of two text mutants that only differ in sensitive information (e.g., gender) but are predicted as different sentiments by an SA system. Our experiments showed that BiasFinder could uncover more BTCs than two baselines (EEC [30] and MT-NLP [31]) on two datasets. Additionally, we evaluate whether the generated texts are fluent by performing a manual annotation study. The results demonstrate that participants consistently consider texts generated by BiasFinder to be more fluent than texts generated by MT-NLP. The contributions of our work are as follows:

Original Text
It seems that **Jake** with all **his** knowledge of the great outdoors didn't realize the danger! **He** enters a mine shaft that's leaking with dangerous gas!

Mutated Text
It seems that **Julia** with all **her** knowledge of the great outdoors didn't realize the danger! **She** enters a mine shaft that's leaking with dangerous gas!

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.

- 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[3].
- BiasFinder automatically identifies and curates appropriate and realistic texts (of various complexity) and transforms them into templates that can be instantiated to detect different types of bias. Prior works only consider a small set of manually-crafted simple templates or focus on detecting one type of bias.
- We compare BiasFinder with two baselines on IMDB and Twitter datasets. The results show that BiasFinder can generate more BTCs, and an annotation study demonstrates that human annotators consistently consider that BiasFinder can generate more fluent text mutants.

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, threats to validity and potential usages of BiasFinder. Section 7 presents related work. Finally, Section 8 concludes this paper and describes future work.

2 PRELIMINARIES

This section provides more details of metamorphic testing for revealing fairness issues (Section 2.1), as well as basic NLP operations that we use as building blocks of our proposed approach (Section 2.2).

2.1 Metamorphic Testing for Fairness

Counterfactual fairness is a widely adopted fairness concept [35]–[38], which is introduced by Kusner et al. who specify

[3]https://github.com/soarsmu/BiasFinder

TABLE 1
Templates in EEC.

No	Template	# Sentence
Sentences with emotion words:		
1	$\langle person \rangle$ feels $\langle emotion \rangle$	1,200
2	The situation makes $\langle person \rangle$ feel $\langle emotion \rangle$	1,200
3	I made $\langle person \rangle$ feel $\langle emotion \rangle$	1,200
4	$\langle person \rangle$ made me feel $\langle emotion \rangle$	1,200
5	$\langle person \rangle$ found himself/herself in a/an $\langle emotion \rangle$ situation	1,200
6	$\langle person \rangle$ told us all about the recent $\langle emotion \rangle$ events	1,200
7	The conversation with $\langle person \rangle$ was $\langle emotion \rangle$	1,200
Sentences with no emotion words:		
8	I saw $\langle person \rangle$ in the market	60
9	I talked to $\langle person \rangle$ yesterday	60
10	$\langle person \rangle$ goes to the school in our neighborhood	60
11	$\langle person \rangle$ has two children	60

that "a decision is fair towards an individual if it is the same in (a) the actual world and (b) a counterfactual world where the individual belonged to a different demographic group" [35]. We formalize this counterfactual fairness specification as a metamorphic relationship.

We first introduce the definitions of fairness and the formalisation of metamorphic testing for uncovering fairness issues in SA systems. An SA system can be abstracted as a function $f : X \rightarrow Y$, which takes a text $x \in X$ as input and produces the sentiment $y = f(x)$ reflected in the input. We expect a fair SA system not to make predictions that are based on emotionally irrelevant but sensitive information (e.g., gender, ethnic groups, countries of origin, etc), which are called **protected features**. We use $p(x)$ to denote the protected features of an input x and $n(x)$ to denote the **non-protected features**. The above expectation for a fair SA system can be formally specified with a metamorphic relationship:

$$\forall x_i, x_j, n(x_i) = n(x_j) \wedge p(x_i) \neq p(x_j) \rightarrow f(x_i) = f(x_j)$$

where x_i and x_j are two inputs that share the same non-protected features (i.e., $n(x_i) = n(x_j)$) but differ in sensitive features (i.e., $p(x_i) \neq p(x_j)$). A fair SA system should make same prediction for the two inputs, i.e., $f(x_i) = f(x_j)$. Pairs of x_i and x_j that violate the metamorphic relationship are referred as *bias-uncovering test case* (BTC).

The Equity Evaluation Corpus (EEC) is a benchmark [30] that leverages the metamorphic relationship to reveal bias in SA systems. The EEC consists of 8,640 sentences designed to reveal gender and race bias. These sentences are constructed by 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.

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

- Common African American female or male first names; Common European American female or male first names; taken from Caliskan et al. [39]
- Noun phrases referring to females, such as "my daughter"; and noun phrases referring to males, such as "my son".

The second placeholder, $\langle emotion \rangle$, corresponds to four basic emotions: anger, fear, joy, and sadness. For each emotion, EEC selects five words from Roget's Thesaurus[4] 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 SA 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 complex sentences can aid in better uncovering bias in SA systems.

2.2 Natural Language Processing (NLP) Techniques

We introduce several NLP techniques used in BiasFinder to identifier sensitive features and build templates.

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 [40]. An example of PoS-tagging is shown in Figure 2. In the example, "Maria" is tagged as a proper noun (PROPN); "has" and "loves" are tagged as verbs (VERB); and "She" and "him" are tagged as pronouns (PRON).

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 [41]. 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 in 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 [42]. 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.

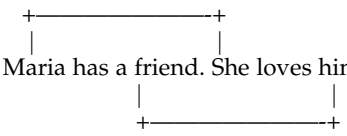
[4] http://www.gutenberg.org/ebooks/22

Input Text
Maria has a friend. She loves him.

POS-tagging
Maria|PROPN has|VERB a|DET friend|NOUN .|PUNCT
She|PRON loves|VERB him|PRON .|PUNCT

NER
Maria | PERSON

Coreferences Resolution



Coreferences
Maria, She
a friend, him

Fig. 2. An example of POS-tagging, NER, and coreference resolution. There are two entities identified by the coreference resolution, "Maria" and "a friend", and the expressions referring to these entities are linked.

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 [43], [44]. 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.

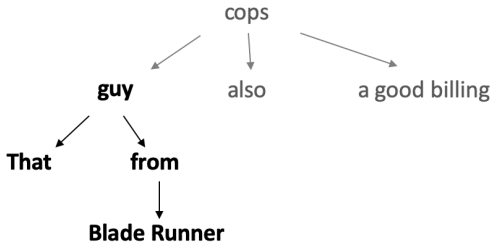


Fig. 3. Example of a dependency parse tree for the sentence "That guy from Blade Runner also cops a good billing". The root word of the phrase "That guy from Blade Runner" (bolded in the above image) is "guy".

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 "guy", 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 inputs, a collection of texts and a sentiment analysis (SA) system, and produces, as outputs, 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 (BTCs).

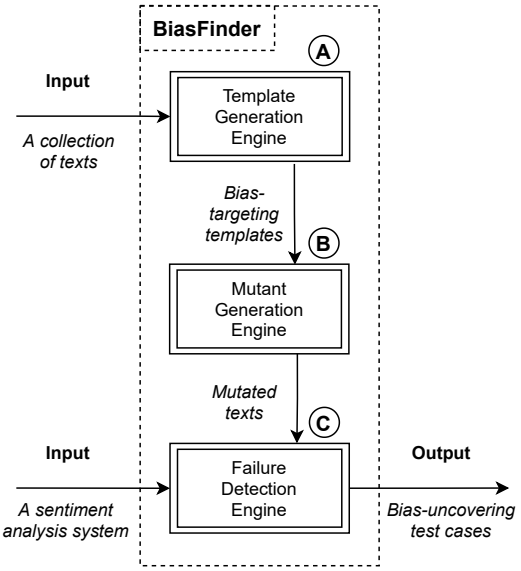


Fig. 4. The architecture of BiasFinder.

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 mutants to uncover the targeted bias (*Step 3*).

3.2 Mutant Generation Engine

To generate mutants from a bias-targeting template, this engine replaces template placeholders with concrete values

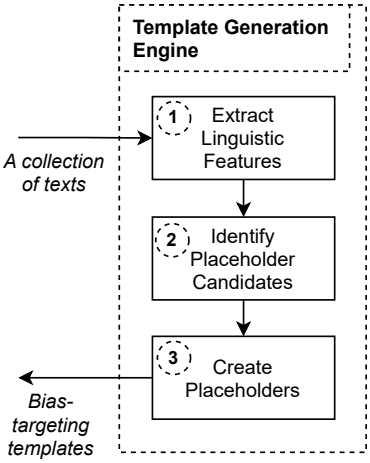


Fig. 5. The workflow of Template Generation Engine.

taken from pre-determined lists of possible values. These lists differ based on the types of bias under consideration. The engine substitutes the placeholders with concrete values while ensuring that the generated mutants are valid. A mutant is valid if and only 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 text mutants, and produces a set of bias-uncovering test cases. It 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 have different sentiments, they is the evidence of a biased prediction. Such pairs of mutants are output as *bias-uncovering test cases*.

3.4 Instantiating BiasFinder to Different Bias

BiasFinder can be instantiated in various ways to uncover different kinds of bias. In this work, we investigate 3 instances of BiasFinder that can uncover gender, occupation, and country-of-origin bias in SA systems. 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.

Algorithm 1: Generating a Template for Detecting Gender Bias.

```
   Input: s: a text, gn: gender nouns
   Output: t : a template or null
1  t = s;
2  corefs = getCoreferences(s);
3  names = getPersonNamedEntities(s);
4  coref = filter(corefs, names, gn);
5  if coref ≠ null then
6  |  for r ∈ coref do
7  |  |  if isPersonName(r, names) then
8  |  |  |  t = createPlaceholder(t, s, r, names);
9  |  |  end
10 |  |  else if isGenderPronoun(r) then
11 |  |  |  t = createPlaceholder(t, s, r);
12 |  |  end
13 |  |  else if hasGenderNoun(r, gn) then
14 |  |  |  t = createPlaceholder(t, s, r, gn);
15 |  |  end
16 |  end
17 end
18 return t == s?null : t
```

GenderBiasFinder generates mutants by changing words associated with the gender of a person, and uncovers gender bias when the SA system predicts differing 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 follow the below descriptions to customize the three main engines of BiasFinder: Template Generation Engine, Mutant Generation Engine, and Failure Detection Engine.

4.1 Template Generation Engine

Algorithm 1 shows the process for generating templates for uncovering gender bias. Given an input text, GenderBiasFinder extracts linguistic features in the form of parts-of-speech, named entities referring to person names, and coreferences. GenderBiasFinder uses coreference resolution (see Section 2.2.3) to find references of entities in the text (Line 2). References to a unique entity are grouped together in a list. The output of the coreference resolution is n lists where n is the total number of entities mentioned in the text, which we refer to as *corefs*. We also run named entity recognition (see Section 2.2.2) to identify person named entities (e.g., person names) in the text (Line 3).

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 18). For each iteration, we have three cases depending on the type of each r:

Case 1: *The Reference is a Person Name (Line 7-9)*

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 $\langle name \rangle$ placeholder (Line 8). In the example shown in Figure 6, "Drew Barrymore" is a person's name and is replaced with this placeholder.

Text
"Never Been Kissed" is a real feel good film. If you haven't seen it yet, then rent it out. I am going to buy it when its released because I loved it. **Drew Barrymore** is excellent again, **she** plays **her** part well. I felt I could relate to this film because of the school days I had were just as bad.

Coreferences
Drew Barrymore, she, her

Person Named Entity
Drew Barrymore

Generated Template
"Never Been Kissed" is a real feel good film. If you haven't seen it yet, then rent it out. I am going to buy it when its released because I loved it. ⟨**name**⟩ is excellent again, ⟨**pro-spp**⟩ plays ⟨**pro-pp**⟩ part well. I felt I could relate to this film because of the school days I had were just as bad.

Fig. 6. An illustrative example for Case 1 and 2 of GenderBiasFinder.

Case 2: *The Reference is a Gender Pronoun (Lines 10-12)*

GenderBiasFinder checks if the reference r is a gender pronoun (Line 10). If so, GenderBiasFinder converts the gender pronoun into $\langle pro\text{-}id \rangle$ (Line 11), where 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 possesive 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 $\langle pro\text{-}spp \rangle$ placeholder, while "her" is converted to $\langle pro\text{-}pp \rangle$ placeholder.

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

GenderBiasFinder checks if the *root word* of the reference r is a gender noun (Line 13). 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 gn, a collection of gender-related nouns, and if it does, converts the root word to $\langle gaw \rangle$ placeholder (Line 14). 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 gn. As it does, GenderBiasFinder replaces "guy" to a $\langle gaw \rangle$ placeholder. Some examples of gender nouns are shown in Table 3. In total, we use 22 gender nouns.

TABLE 2
Examples of names from GenderComputer.

Name	Gender	Country-of-origin
Felipe	Male	Brazil
Abhishek	Male	India
Barbora	Female	Czech
Zeynep	Female	Turkey

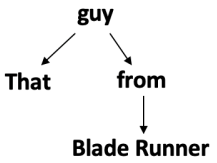
TABLE 3
Examples of nouns reflecting gender information.

Male	Female
boy, brother, father, dad, . . .	girl, sister, mother, mom, ...

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.

Coreferences
That guy from "Blade Runner", he

Dependency Parsing of The Reference



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 ⟨**gaw**⟩ from "Blade Runner" also cops a good billing, although ⟨**pro-spp**⟩ only turns up at the beginning and the end of the movie.

Fig. 7. An illustrative example for case 3 of GenderBiasFinder.

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 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 Gender-

Computer[5]. 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. Examples of names from GenderComputer are shown in Table 2. 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. To avoid the results being affected by other types of bias, we only pick person names originating from a single country, i.e. the USA. To do so, we only choose the person names from the USA category in the GenderComputer database. The USA names from GenderComputer are accompanied by frequency information (a number indicating how frequently a name is used). We select N male names and N female names of the highest frequency. By default, N is set to 30.

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

$\langle$***gaw***$\rangle$ **Placeholder:** Values to be substituted for this placeholder are the set of gender nouns taken from several English resources[6,7,8] 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 from these two classes that have different sentiments are paired. In other words, the engine finds 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.

5 Other Instances of BiasFinder

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

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 differing sentiment for a pair of mutants that differ only on the occupation referred in the text. We perform these customizations to uncover occupation bias:

Algorithm 2: Generating a Template for Detecting Occupation Bias

```
   Input: s: a text
   Output: t : a template or null
 1 t = s;
 2 occs = getOccNamedEntities(s);
 3 for occ ∈ occs do
 4     if isNoun(occ) then
 5         if hasAdjective(t, occ) then
 6             t = removeAdjective(t, occ);
 7         end
 8         t = createPlaceholders(t, occ);
 9         occCorefs = getCoreferencesOf(s, occ);
10         for r ∈ occCorefs do
11             if isRefContainsOcc(r, occ) then
12                 if hasAdjective(t, r) then
13                     t = removeAdjective(t, r);
14                 end
15                 t = createPlaceholders(t, r);
16             end
17         end
18         return t;
19     end
20 end
21 return null
```

Template Generation Engine: BiasFinder generates occupation templates by following Algorithm 2. BiasFinder first extracts the list of occupations $occs$ mentioned in the input text s using named entity recognition (Line 2). BiasFinder then iterates each occupation occ from $occs$ (Line 3). BiasFinder then confirms 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, BiasFinder removes 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". BiasFinder then converts occ to $\langle occupation \rangle$ placeholder (Line 8). BiasFinder also converts determiner "a" or "an" in front of occ (if it exists) to $\langle det \rangle$ placeholder to ensure the produced mutant template is grammatically correct. Next, BiasFinder extracts $occCorefs$ (Line 9); $occCorefs$ is the list of references in s that refers to the same entity that occ refers to. BiasFinder then iterates each reference r from $occRefs$ (Line 10). BiasFinder checks whether r is a mention of occ (Line 11). For such r, BiasFinder again creates $\langle occupation \rangle$ and $\langle det \rangle$ placeholders (if necessary), after removing adjectives (if necessary) (Lines 12-15). At the end of this process, BiasFinder outputs a template t generated from the input text s (Line 18).

In the example shown in Figure 8, BiasFinder detects "doctor" and "journalist" as occupations. BiasFinder only uses the first occupation to form a template. As "doctor" is a noun, and it is not preceded by any adjective, BiasFinder replaces it directly to $\langle occupation \rangle$ placeholder. BiasFinder then replaces its determiner with $\langle det \rangle$ placeholder. In this case, there are no coreferences of "doctor", so the template generation process ends.

[5]https://github.com/tue-mdse/genderComputer
[6]https://7esl.com/gender-of-nouns/
[7]http://www.primaryresources.co.uk/english/PC_gen.htm
[8]https://ielts.com.au/articles/grammar-101-feminine-and-masculine-words-in-english/

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. William Holden never looked better playing a romantic lead as a journalist covering war torn regions in the world.

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. William Holden never looked better playing a romantic lead as a journalist covering war torn regions in the world.

Fig. 8. An illustrative example for OccupationBiasFinder.

Algorithm 3: Generating a Template for Detecting Country-of-Origin Bias

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* $\neq$ null **then**
5 g = inferGender(*coref*);
6 **if** $g \in \{Male, Female\}$ **then**
7 **for** $r \in$ *coref* **do**
8 **if** isPersonName(*r, names*) **then**
9 t = *createPlaceholder*(*t, s, r, g*);
10 **end**
11 **end**
12 **return** t
13 **end**
14 **end**
15 **return** $null$

Mutant Generation Engine: To generate occupation mutants, the engine substitutes the $\langle occupation \rangle$ placeholder with a value from a set of 79 honest (i.e., non-criminal) and gender-neutral occupation names that are taken from [45]–[47]. The value of $\langle det \rangle$ is linked with the value of $\langle occupation \rangle$ placeholder. For example, the values of $\langle det \rangle$ 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, BiasFinder follows Algorithm 3. BiasFinder first runs coreference resolution to find *corefs*, which contains references of persons mentioned in the input text s (Line 1). BiasFinder also runs named entity recognition to extract the list of person names mentioned in s (Line 2), which BiasFinder refers to as $names$.

Next, BiasFinder filters 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*. BiasFinder stops the template generation process if *null* is returned.

Otherwise, if the references in $coref$ refer to a consistent gender g (Line 6) – 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) – BiasFinder iterates each reference r in $coref$ (Lines 7-11). If r is the person name in $names$, BiasFinder replaces r with a placeholder representing the gender that was detected (Lines 8-10). A $\langle male \rangle$ or $\langle female \rangle$ placeholder is created if a male or a female gender was detected, respectively.

In the example shown in Figure 9, "Lauren Holly" is detected as a person name and the coreferences consistently refer to the female gender. Thus, BiasFinder replaces "Lauren Holly" with $\langle female \rangle$ placeholder.

Text
I loved this movie, it was cute and funny! **Lauren Holly** was wonderful, she's funny and very believable in her role.

Coreferences
Lauren Holly, she, her

Person Named Entity
Lauren Holly

Generated Template
I loved this movie, it was cute and funny! ⟨**female**⟩ was wonderful, she's funny and very believable in her role.

Fig. 9. An illustrative example for CountryBiasFinder.

Mutant Generation Engine: To generate country-of-origin mutants, the engine substitutes $\langle male \rangle$ and $\langle female \rangle$ placeholders with values from a set of people names taken from GenderComputer[9]. 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 unique names 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. The placeholder values are then filled in based on the gender associated with the name. Male and female names are used to fill $\langle male \rangle$ and $\langle female \rangle$ placeholders, respectively.

[9]https://github.com/tue-mdse/genderComputer

Failure Detection Engine: The engine accepts the generated mutants as input and feeds 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.

TABLE 4
The performance of SA models on the datasets.

Model	Accuracy	
	IMDB	Twitter S140
BERT-base-cased	89.12%	83.54%
RoBERTa-base	92.84%	86.24%
ALBERT-base-v2	89.45%	82.96%
Muppet-RoBERTa-base	95.29%	85.69%
ELECTRA-base	91.20%	84.39%

6 Experiments

In this section, we describe our dataset, experimental settings, evaluation metrics, and research questions. Then, we answer the research questions, analyze threats to validity, and discuss potential usage of BiasFinder.

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 is the IMDB dataset [48], which 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 texts that are not a natural language, e.g., HTML tags. We remove these texts from the movie reviews. Then, we split the 50,000 movie reviews evenly to train and test sets. In addition to the IMDB dataset, we also use the Twitter Sentiment140 dataset [49]–[51], which contains 1.6 million texts associated with either positive or negative sentiments. We randomly pick 400,000 texts as the train set and 100,000 texts as the test set. The selected train set and the test set are mutually exclusive.

We use fine-tuned five Transformer-based models on two datasets to obtain the SA systems in our experiments. Transformer-based models have achieved state-of-the-art performances on many NLP tasks (including sentiment analysis) in recent years [26], [52], [53]. In this work, we use the implementations available in HuggingFace[10] to fine-tune a number of recently proposed Transformer models (including Google BERT [54], Facebook RoBERTa [55], Google ALBERT [56], Google ELECTRA [57], and Facebook Muppet [58]) on the IMDB and Twitter datasets to obtain SA models. We report the accuracy of our SA models on the test set of each dataset in Table 4. The models' performance on the IMDB dataset is high and comparable to the accuracy reported in a recent work that also fine-tunes BERT for sentiment analysis [27]. The performance of our models on randomly sampled data from the Twitter dataset is higher than the performance reported in the recent models presented by Tay et al. [49].

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[11]. We use both SpaCy[12] and Stanford CoreNLP[13] for Part-of-Speech (PoS) Tagging and Named Entity Recognition (NER). The models used for these NLP tasks can be leveraged directly without any further fine-tuning.

In our experiment, we compare BiasFinder with two baselines: EEC [30] that uses 11 static templates to generate texts of various features and MT-NLP [31] that performs *analogy mutation* and *active mutation* to produce discriminatory inputs. For example, MT-NLP changes "actor" (indicating male) to its *analogy* noun "actress" (indicating female). The *active mutation* is operated by inserting an adjective word in front of nouns that represent humans, e.g. inserting "male" before "soldiers" to get "male soldiers". MT-NLP focuses on gender bias, while BiasFinder provides a general framework that targets various types of bias.

Our objective in the study is to produce test cases that reveal bias. As defined in Section 2.1, a bias-uncovering test case (BTC) is a pair of texts that only differ in protected features (e.g., gender), but are predicted as different sentiments by an SA system. The authors of MT-NLP [31] used the number of *fairness violations* found as a metric to evaluate the ability of a fairness testing tool in uncovering bias in SA systems. The *fairness violation* concept is equivalent to BTC in this paper.

We also perform an annotation study to evaluate whether the mutants generated by BiasFinder and MT-NLP are fluent since an SA system may change its prediction because a text mutant is not fluent rather than because of actual bias. Fluency is one of the gold-standard human evaluation metrics to evaluate the linguistic quality of generated texts [59]. Fluency is defined as the quality of individual sentences. A fluent sentence should have no formatting problems, capitalization errors or obvious grammatical issues that make the text difficult to read [60]. We consider a mutant to be fluent if the changed parts from the sentences: (1) look natural, i.e., no formatting problems, capitalization errors or obvious grammatical issues, (2) are consistent with each other and the other words, and (3) do not introduce redundant words.

6.2 Research Questions

RQ1. *How many BTCs can BiasFinder generate? How does it compare with EEC and MT-NLP?*

BiasFinder is the first approach to automatically generate templates and text mutants to uncover multiple types of bias. We report the number of BTCs produced by the 3

[10]https://huggingface.co/
[11]https://github.com/huggingface/neuralcoref
[12]https://spacy.io/
[13]https://stanfordnlp.github.io/CoreNLP/

TABLE 5
The number of templates and mutants generated by BiasFinder, EEC and MT-NLP on the IMDB and Twitter datasets.

Type	Tool	IMDB		Twitter	
		template	mutant	template	mutant
Gender	BiasFinder	3,015	153,866	1,769	63,104
	MT-NLP	95,219	285,976	15,150	59,462
	EEC	140	8,400	140	8,400
Country	BiasFinder	2,828	70,700	959	23,975
Occupation	BiasFinder	14,319	1,131,201	202	15,958

instances of BiasFinder for each SA system. Since EEC and MT-NLP only target at gender bias, we only compare them with GenderBiasFinder to make the comparison fair. We also report the performance of the other two instances of BiasFinder (for occupation and country-of-origin bias).

RQ2. *How fluent are the generated mutants?*

We evaluate the fluency of the generated mutants via an annotation study. The annotation study involves two participants, both of whom are native English speakers and are not authors of this paper. The participants were asked to rate the fluency of each mutant using a Likert scale of 1 to 3. Score 1 indicates a non-fluent text (a mutated part looks absurd or inconsistent), 2 indicates a somewhat fluent text (a mutated part looks natural but it is not fully consistent with the text or contains some redundant terms), and 3 indicates a fluent text (all mutated parts look natural, are consistent with each other and the remaining text, and do not contain redundant terms). We consider that the fluency of the mutants to be passable if the average ratings given to the mutants are at least in the middle of the Likert scale (i.e., 2).

The participants were asked to label randomly sampled mutants generated by BiasFinder (for gender, country-of-origin, and occupation bias) and MT-NLP (for gender bias). The participants do not know which tools generated which mutants. BiasFinder generates 216,970, 1,147,159, and 94,675 mutants for gender, occupation and country-of-origin bias, respectively. MT-NLP generates 345,438 mutants for gender bias. We want to analyze statistically representative samples of those mutants to investigate their quality. To compute the size of statistically significant sample size, we use a popular online sample size calculator [61], which implements the Cochran's formula [62]. This online sample size calculator is also used in a number of prior SE works, e.g., [63]–[65]. We specify 95% confidence level and 5% confidence interval, which are the same or a stricter setting than those used in the prior works [63]–[65]; this setting gives our findings a confidence level of 95% with a margin of error of 5%. Running the online sample size calculator on each of the 4 mutant populations returns us either 383 or 384. To standardize, we sample 384 mutants from each mutant population. Thus, each of our annotation study participants needs to rate 384 $\times$ 4 mutants = 1,536 mutants. To determine the level of agreement between the two participants in the annotation study, we computed Cohen's Kappa [45] and obtained a value of 0.55 – usually interpreted as moderate agreement [46], [47].

TABLE 6
The number of BTCs uncovered for gender bias.

Model	Tool	# BTC	
		IMDB	Twitter
BERT-base-cased	BiasFinder	7,723	14,373
	MT-NLP	1,447	793
	EEC	5,674	238
RoBERTa-base	BiasFinder	8,051	29,167
	MT-NLP	779	886
	EEC	4,560	186
AlBERT-base-v2	BiasFinder	14,966	15,735
	MT-NLP	993	705
	EEC	2,678	420
Muppet-RoBERTa-base	BiasFinder	5,747	51,569
	MT-NLP	834	861
	EEC	4,694	360
ELECTRA-base	BiasFinder	5,862	13,573
	MT-NLP	477	783
	EEC	5,336	906
Average	BiasFinder	8,469.8	24,883.4
	MT-NLP	906	805.6
	EEC	4,588.4	422

6.3 Results

RQ1. *How many BTCs can BiasFinder generate?*

Table 6 shows the numbers of gender BTCs found by BiasFinder, MT-NLP, and EEC for the 5 SA models investigated in our experiments. On the IMDB dataset, BiasFinder reveals the highest number of gender BTCs for all SA models (8,469.8 on average), while MT-NLP and EEC can only uncover 906 and 4,588.4 gender BTCs. On the Twitter dataset, BiasFinder also reveals the highest number of BTCs for each SA model (24,883.4 on average). On the other hand, MT-NLP and EEC only find 805.6 and 422 BTCs, both of which are two orders of magnitude lower than the numbers of gender BTCs found by BiasFinder. The comparison results on the two datasets highlight the superior capability of BiasFinder in exposing gender bias.

Table 7 shows the numbers of country-of-origin BTCs found by BiasFinder on the IMDB and Twitter datasets. On average, BiasFinder finds 3,363.2 country-of-origin BTCs on the IMDB dataset and 5,166.4 country-of-origin BTCs on the Twitter dataset. Table 8 shows the numbers of occupation BTCs found by BiasFinder on the IMDB and Twitter datasets. We can observe that the average number of occupation BTCs found on the IMDB dataset is 144,683.2 and the average number of occupation BTCs found on the Twitter dataset is 19,021.2.

TABLE 7
The number of BTCs found by BiasFinder for country-of-origin bias.

Model	# BTC	
	IMDB	Twitter
BERT-base-cased	4,794	4,380
RoBERTa-base	2,620	6,810
AlBERT-base-v2	3,566	4,096
Muppet-RoBERTa-base	2,832	5,554
ELECTRA-base	3,004	4,992
Average	3,363.2	5,166.4

Figure 10, 11 and 12 show examples of BTCs for gender,

occupation, and country-of-origin, respectively.

TABLE 8
The number of BTCs found by BiasFinder for occupation bias.

Model	# BTC IMDB	Twitter
BERT-base-cased	200,400	16,938
RoBERTa-base	134,926	25,656
AlBERT-base-v2	184,496	16,956
Muppet-RoBERTa-base	85,256	20,098
ELECTRA-base	118,338	15,458
Average	144,683.2	19,021.2

Mutated Text - using a uniquely male name
What is **he** supposed to be? **He** was a kid in the past, ... and the future? This movie had a lot of problems. Is **he** a ghost, or just a strong kid. Man, ... what a piece of crap. I'm still confused. Also, is **he** supposed to be an abortion? Strange. Very strange. This movie will mess with your mind, ... and it's not very scary, ... just confusing. Why was **he** , ... Where did, ... What was the, ... oh, who cares, ... **Benedetto** isn't worth it, ... My score: 10

Mutated Text - using a uniquely female name
What is **she** supposed to be? **She** was a kid in the past, ... and the future? This movie had a lot of problems. Is **she** a ghost, or just a strong kid. Man, ... what a piece of crap. I'm still confused. Also, is **she** supposed to be an abortion? Strange. Very strange. This movie will mess with your mind, ... and it's not very scary, ... just confusing. Why was **she** , ... Where did, ... What was the, ... oh, who cares, ... **Elaisha** isn't worth it, ... My score: 10

Fig. 10. An example of BTC for uncovering gender bias.

Mutated Text - Housekeeper
Great underrated movie great action good actors and a wonderful story line. Wesley is verry good and the **housekeeper** the bad guy is wonderful The girl plays a nice role and the comedy mixed with blakness!

Mutated Text - Programmer
Great underrated movie great action good actors and a wonderful story line. Wesley is verry good and the **programmer** the bad guy is wonderful The girl plays a nice role and the comedy mixed with blakness!

Fig. 11. An example of BTC for uncovering occupation bias.

RQ2. *How fluent are the generated mutants?*
Table 9 shows the result of the annotation study. We find that the average fluency ratings of mutants range from 1.77 to 3. For gender bias, the average fluency rating of mutants generated by BiasFinder (2.61 out of 3) is 28.57%[14] higher than those generated by MT-NLP (2.03 out of 3), which highlights the better linguistic quality of the BiasFinder mutants. For country-of-origin bias, both participants gave the maximum

[14] $(2.61 - 2.03)/2.03 \times 100\%$

Mutated Text - using a 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 on of the Jazz musicians in Germany

Mutated Text - using a 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 on of the Jazz musicians in Germany

Fig. 12. An example of BTC for uncovering country-of-origin bias. (...) is a truncated piece of the original text.

TABLE 9
User-annotated fluency scores of mutants generated by BiasFinder and MT-NLP [31]. Mutants with higher scores are more fluent.

Type	Tool	Fluency Participant 1	Participant 2	All
Gender	BiasFinder	2.32	2.89	2.61
	MT-NLP	1.78	2.28	2.03
Country	BiasFinder	3	3	3
Occupation	BiasFinder	1.46	2.08	1.77

fluency rating (3 out of 3) for all mutants. For occupation bias, the average fluency rating of the mutants is 1.77. This is still higher than the halfway of the Likert scale (2) and thus we consider it to be passable. Still, the generated occupation mutants are not as fluent as the mutants generated for the other bias.

We investigated the non-fluent mutants produced by BiasFinder for occupation bias and we show an example in Figure 13. For that example, although BiasFinder successfully identified the word "driver" as an occupation and can generate a placeholder for it, replacing the placeholder with another occupation results in a non-fluent text. The usage of the word "driver" is specific to the context described in the text, and it cannot be replaced with many other occupations without losing fluency.

Original Text
Boris Leskin as Alex's grandfather and **driver** of the tour car makes a valuable contribution to the film, as well as Laryssa Lauret, who is seen in the last part of the movie.

Mutated Text
Boris Leskin as Alex's grandfather and **secretary** of the tour car makes a valuable contribution to the film, as well as Laryssa Lauret, who is seen in the last part of the movie.

Fig. 13. An example of a non-fluent mutant. The usage of the "driver" word is context specific, and cannot be replaced with another occupation (i.e., "secretary").

6.4 Threats to Validity

We have only experimented with SA models fine-tuned on 5 Transformer-based models and generated templates

on the IMDB and Twitter datasets. The results may not generalize to other SA systems and datasets. However, Transformer-based models are among the top performing models for text classification in recent years [26], [52]–[54]. The IMDB and Twitter datasets are commonly used datasets for studying sentiment analysis [28], [66], [67]. Another threat to validity is that some of the found BTCs may be caused by actual errors instead of bias, especially when the SA models under investigation have poor performance. To minimize this threat, we evaluate BiasFinder and baselines on SA models constructed by fine-tuning state-of-the-art Transformer-based models. Table 4 shows that they all have high prediction accuracies on SA tasks.

The names used by BiasFinder are gathered from the GenderComputer database. Although it is claimed in its documentation that GenderComputer provides lists of male and female first names for different countries, some last names are found in their database. These last names are gender independent and may affect BiasFinder's ability to identify BTCs if last names are used to generate mutants. To mitigate this threat, we manually check the selected names (30 male names and 30 female males) used in our experiments to ensure they are all first names.

6.5 Potential Usage

In this paper, BiasFinder mainly serves as a fairness testing tool for SA systems. We believe that the mutants generated by BiasFinder can be utilized in other parts of the SA systems life cycle, including model training, deployment, and repair. When training an SA model, BiasFinder can be used to augment the training set with texts of diverse gender information to mitigate bias. At the deployment stage, the BiasFinder's idea can be transferred to detect biased predictions at runtime. Since BiasFinder can dynamically find templates for any input text, a biased prediction can be detected at runtime by comparing the prediction from the input text with the predictions from mutated input texts. After detecting biased predictions, one can heal unfairness by leveraging prediction results for these mutants (e.g., using the majority predictions for the mutants as the final result). We have recently demonstrated the usage of BiasFinder in two downstream tasks: runtime verification of bias [68] and automatic healing of bias [69]. BiasRV [68] is built on top of BiasFinder to identify unfair predictions made by a deployed sentiment analysis system on the fly. Specifically, BiasRV defines a *distributional fairness* concept, a fairness property that can be verified at runtime by analyzing the output distributions of mutants generated by BiasFinder. BiasHeal [69] uses several rules to analyze an SA system's predictions on mutants generated by BiasFinder and shows that unfair results can be healed on the fly without sacrificing accuracy. In addition to the above two use cases, BiasFinder can also be potentially utilized as a data augmentation technique to enhance model fairness by retraining models on BTCs. The above strategies can help to improve the fairness of sentiment analysis systems, and they can potentially be extended to cover other text analytics or natural language processing systems to ensure that they are free from different types of bias.

Besides, BiasFinder can also be potentially used in a wider range of applications beyond sentiment analysis. In general, sentiment analysis can be viewed as a specialized text classification with class labels corresponding to the sentiment polarities of the text. Conceptually, BiasFinder can be applicable to detect fairness issues in other text classification tasks. There are many text classification tasks where the main difference with sentiment analysis is in the class labels e.g., spam vs. non-spam, fraudulent claim vs. legitimate claim, fake news vs. real news, etc. Moreover, for many of these tasks, fairness issues are also highly relevant. Text classification is also a building block of many other NLP solutions, e.g., chatbot.

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 related works in 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], [70], [71]. An AI system may perpetuate human bias and perform differently for some demographic groups than others [1], [32], [70], [71]. As such, many existing studies on uncovering bias [1]–[3], [5], [30] focus on finding differences in the system's behavior given a change in a demographic characteristic (aka. sensitive attribute). Our approach has the same high-level objective of uncovering differences in behavior when the 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 SA systems that 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 SA systems, Diaz et al. [32] manually identify and replace words that explicitly or implicitly encode age information in input texts to uncover age-related bias. 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 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.

Compared to these prior works, our work is "wider" in two aspects: First, many of them require extensive manual steps (e.g., creating limited numbers of templates manually) while our work is fully automated. Second, many of them focus on only one kind of fairness issue (e.g., gender bias only), while we have shown that our approach can be generalized across multiple fairness issues (i.e., gender bias, country-of-origin bias, occupation bias).

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. [72].

Existing studies have applied metamorphic testing to AI systems [73]–[76]. Many of these systems focus on finding bugs, for example, in machine translation [73], [76] , autonomous driving systems [74], [75], or automatic speech recognition [77]–[79]. 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 SA systems.

In the NLP domain, some research efforts have developed methods for generating adversarial examples [80], [81], while other researchers have proposed techniques to test robustness to typos and other forms of noise [82], or changes in the names of people mentioned in text [83]. 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 bias. 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 bias in Sentiment Analysis (SA) systems. BiasFinder can be instantiated for different demographic characteristics, such as gender or occupation. Given a characteristic of interest, 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 SA systems and identify bias-uncovering test cases. By analyzing a realistic and diverse corpus, BiasFinder can produce realistic and diverse bias-uncovering test cases. BiasFinder generates templates of test cases involving other characteristics, including gender, occupation and country-of-origin. Together, the template and mutation generation produces test cases that cover a wider range of scenarios.

We empirically evaluated BiasFinder against two prior works. For gender bias, BiasFinder can uncover more BTCs than both EEC and MT-NLP on all SA systems under investigation. BiasFinder can also find additional BTCs for occupation and country-of-origin bias. Through a manual annotation study, we show that human annotators consistently consider mutants generated by BiasFinder are more fluent than mutants generated by MT-NLP.

In the future, we plan to instantiate BiasFinder for more types of bias and extend the experiments (e.g., by considering other text corpora). Moreover, we will evaluate BiasFinder to determine if it generalizes to other NLP tasks beyond sentiment analysis, for example, testing general text classifiers.

Acknowledgment

This research was supported by the Singapore Ministry of Education (MOE) Academic Research Fund (AcRF) Tier 1 grant.

References

[1] S. Galhotra, Y. Brun, and A. Meliou, "Fairness testing: testing software for discrimination," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 2017, pp. 498–510.

[2] S. Udeshi, P. Arora, and S. Chattopadhyay, "Automated directed fairness testing," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 98–108.

[3] F. Tramer, V. Atlidakis, R. Geambasu, D. Hsu, J.-P. Hubaux, M. Humbert, A. Juels, and H. Lin, "Fairtest: Discovering unwarranted associations in data-driven applications," in *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2017, pp. 401–416.

[4] J. Chakraborty, S. Majumder, Z. Yu, and T. Menzies, "Fairway: a way to build fair ml software," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 654–665.

[5] M. T. Ribeiro, T. Wu, C. Guestrin, and S. Singh, "Beyond accuracy: Behavioral testing of nlp models with checklist," *Association for Computational Linguistics (ACL 2020*, 2020.

[6] B. Pang, L. Lee, and S. Vaithyanathan, "Thumbs up? sentiment classification using machine learning techniques," in *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10*, ser. EMNLP '02. USA: Association for Computational Linguistics, 2002, p. 79–86. [Online]. Available: https://doi.org/10.3115/1118693.1118704

[7] P. 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*, ser. ACL '02. USA: Association for Computational Linguistics, 2002, p. 417–424. [Online]. Available: https://doi.org/10.3115/1073083.1073153

[8] W. Medhat, A. Hassan, and H. Korashy, "Sentiment analysis algorithms and applications: A survey," *Ain Shams engineering journal*, vol. 5, no. 4, pp. 1093–1113, 2014.

[9] S. Poria, D. Hazarika, N. Majumder, and R. Mihalcea, "Beneath the tip of the iceberg: Current challenges and new directions in sentiment analysis research," *IEEE Transactions on Affective Computing*, 2020, accepted (early access at: https://ieeexplore.ieee.org/document/9260964).

[10] M. Haselmayer and M. Jenny, "Sentiment analysis of political communication: combining a dictionary approach with crowdcoding," *Quality & Quantity*, vol. 51, pp. 2623 – 2646, 2017.

[11] J. A. Caetano, H. S. Lima, M. F. Santos, and H. T. Marques-Neto, "Using sentiment analysis to define twitter political users' classes and their homophily during the 2016 american presidential election," *Journal of Internet Services and Applications*, vol. 9, pp. 1–15, 2018.

[12] S. Krishnamoorthy, "Sentiment analysis of financial news articles using performance indicators," *Knowl. Inf. Syst.*, vol. 56, no. 2, p. 373–394, Aug. 2018. [Online]. Available: https://doi.org/10.1007/s10115-017-1134-1

[13] T. Renault, "Sentiment analysis and machine learning in finance: a comparison of methods and models on one million messages," *Digital Finance*, 09 2019.

[14] M. Day and C. Lee, "Deep learning for financial sentiment analysis on finance news providers," in *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, 2016, pp. 1127–1134.

[15] S. Sohangir, D. Wang, A. Pomeranets, and T. M. Khoshgoftaar, "Big data: Deep learning for financial sentiment analysis," *Journal of Big Data*, vol. 5, pp. 1–25, 2017.

[16] M. Rambocas, "Marketing research: The role of sentiment analysis," *FEP WORKING PAPER SERIES*, 04 2013.

[17] S. Rani and P. Kumar, "A sentiment analysis system to improve teaching and learning," *Computer*, vol. 50, no. 05, pp. 36–43, may 2017.

[18] N. Altrabsheh, M. Gaber, and E. Haig, "Sa-e: Sentiment analysis for education," in *Frontiers in Artificial Intelligence and Applications*, vol. 255, 06 2013.

[19] F. S. Dolianiti, D. Iakovakis, S. B. Dias, S. Hadjileontiadou, J. A. Diniz, and L. Hadjileontiadis, "Sentiment analysis techniques and applications in education: A survey," in *Technology and Innovation in Learning, Teaching and Education*, M. Tsitouridou, J. A. Diniz, and T. A. Mikropoulos, Eds. Cham: Springer International Publishing, 2019, pp. 412–427.

[20] V. S. Gupta and S. Kohli, "Twitter sentiment analysis in healthcare using hadoop and r," in *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, 2016, pp. 3766–3772.

[21] O. Oyebode, F. Alqahtani, and R. Orji, "Using machine learning and thematic analysis methods to evaluate mental health apps based on user reviews," *IEEE Access*, vol. 8, pp. 111 141–111 158, 2020.

[22] S. Yadav, A. Ekbal, S. Saha, and P. Bhattacharyya, "Medical sentiment analysis using social media: Towards building a patient assisted system," in *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*. Miyazaki, Japan: European Language Resources Association (ELRA), May 2018. [Online]. Available: https://www.aclweb.org/anthology/L18-1442

[23] M. Chen, "Efficient vector representation for documents through corruption," *arXiv preprint arXiv:1707.02377*, 2017.

[24] Y. Zhang, Q. Liu, and L. Song, "Sentence-state lstm for text representation," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2018, pp. 317–327.

[25] J. Gong, X. Qiu, S. Wang, and X. Huang, "Information aggregation via dynamic routing for sequence encoding," *arXiv preprint arXiv:1806.01501*, 2018.

[26] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," in *Advances in neural information processing systems*, 2019, pp. 5753–5763.

[27] C. Sun, X. Qiu, Y. Xu, and X. Huang, "How to fine-tune bert for text classification?" in *China National Conference on Chinese Computational Linguistics*. Springer, 2019, pp. 194–206.

[28] J. Howard and S. Ruder, "Universal language model fine-tuning for text classification," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2018, pp. 328–339.

[29] E. Cambria, S. Poria, A. Gelbukh, and M. Thelwall, "Sentiment analysis is a big suitcase," *IEEE Intelligent Systems*, vol. 32, no. 6, pp. 74–80, 2017.

[30] S. Kiritchenko and S. Mohammad, "Examining gender and race bias in two hundred sentiment analysis systems," in *Proceedings of the Seventh Joint Conference on Lexical and Computational Semantics*, 2018, pp. 43–53.

[31] P. Ma, S. Wang, and J. Liu, "Metamorphic testing and certified mitigation of fairness violations in nlp models," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, C. Bessiere, Ed. International Joint Conferences on Artificial Intelligence Organization, 7 2020, pp. 458–465, main track. [Online]. Available: https://doi.org/10.24963/ijcai.2020/64

[32] M. Díaz, I. Johnson, A. Lazar, A. M. Piper, and D. Gergle, "Addressing age-related bias in sentiment analysis," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 2018, pp. 1–14.

[33] J. Bhaskaran and I. Bhallamudi, "Good secretaries, bad truck drivers? occupational gender stereotypes in sentiment analysis," in *Proceedings of the First Workshop on Gender Bias in Natural Language Processing*. Florence, Italy: Association for Computational Linguistics, Aug. 2019, pp. 62–68. [Online]. Available: https://aclanthology.org/W19-3809

[34] E. Soremekun, S. Udeshi, and S. Chattopadhyay, "Astraea: Grammar-based fairness testing," 2021.

[35] M. J. Kusner, J. Loftus, C. Russell, and R. Silva, "Counterfactual fairness," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/a486cd07e4ac3d270571622f4f316ec5-Paper.pdf

[36] Y. Wu, L. Zhang, and X. Wu, "Counterfactual fairness: Unidentification, bound and algorithm," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, 7 2019, pp. 1438–1444. [Online]. Available: https://doi.org/10.24963/ijcai.2019/199

[37] S. Chiappa, "Path-specific counterfactual fairness," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, pp. 7801–7808, Jul. 2019. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/4777

[38] S. Garg, V. Perot, N. Limtiaco, A. Taly, E. H. Chi, and A. Beutel, "Counterfactual fairness in text classification through robustness," in *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, 2019, pp. 219–226.

[39] A. Caliskan, J. Bryson, and A. Narayanan, "Semantics derived automatically from language corpora contain human-like biases," *Science*, vol. 356, pp. 183–186, 04 2017.

[40] E. Brill, "Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging," *Comput. Linguist.*, vol. 21, no. 4, p. 543–565, Dec. 1995.

[41] D. Nadeau and S. Sekine, "A survey of named entity recognition and classification," *Lingvisticae Investigationes*, vol. 30, pp. 3–26, 2007.

[42] W. M. Soon, H. T. Ng, and D. C. Y. Lim, "A machine learning approach to coreference resolution of noun phrases," *Computational Linguistics*, vol. 27, no. 4, pp. 521–544, 2001. [Online]. Available: https://www.aclweb.org/anthology/J01-4004

[43] J. Nivre and S. Kübler, "Dependency parsing," *Synthesis Lectures on Human Language Technologies*, vol. 2, 01 2009.

[44] D. Chen and C. Manning, "A fast and accurate dependency parser using neural networks," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 740–750. [Online]. Available: https://www.aclweb.org/anthology/D14-1082

[45] T. Bolukbasi, K.-W. Chang, J. Zou, V. Saligrama, and A. Kalai, "Man is to computer programmer as woman is to homemaker? debiasing word embeddings," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, ser. NIPS'16. Red Hook, NY, USA: Curran Associates Inc., 2016, p. 4356–4364.

[46] A. Caliskan-Islam, J. Bryson, and A. Narayanan, "Semantics derived automatically from language corpora necessarily contain human biases," *Science*, vol. 356, 08 2016.

[47] J. Zhao, T. Wang, M. Yatskar, V. Ordonez, and K.-W. Chang, "Gender bias in coreference resolution: Evaluation and debiasing methods," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, Jun. 2018, pp. 15–20. [Online]. Available: https://www.aclweb.org/anthology/N18-2003

[48] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, June 2011, pp. 142–150. [Online]. Available: http://www.aclweb.org/anthology/P11-1015

[49] Y. Tay, M. Dehghani, J. P. Gupta, V. K. Aribandi, D. Bahri, Z. Qin, and D. Metzler, "Are pretrained convolutions better than pretrained transformers?" in *ACL 2021*, 2021.

[50] A. Abbasi, A. Hassan, and M. Dhar, "Benchmarking Twitter sentiment analysis tools," in *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*. Reykjavik, Iceland: European Language Resources Association

(ELRA), May 2014, pp. 823–829. [Online]. Available: http://www.lrec-conf.org/proceedings/lrec2014/pdf/483_Paper.pdf
[51] A. Go, R. Bhayani, and L. Huang, "Twitter sentiment classification using distant supervision," pp. 1–12, 2009. [Online]. Available: http://www.stanford.edu/~alecmgo/papers/TwitterDistantSupervision09.pdf
[52] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," vol. 33, pp. 1877–1901, 2020. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf
[53] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of Machine Learning Research*, vol. 21, no. 140, pp. 1–67, 2020. [Online]. Available: http://jmlr.org/papers/v21/20-074.html
[54] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 2019, pp. 4171–4186.
[55] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," 2019.
[56] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=H1eA7AEtvS
[57] K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning, "Electra: Pre-training text encoders as discriminators rather than generators," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=r1xMH1BtvB
[58] A. Aghajanyan, A. Gupta, A. Shrivastava, X. Chen, L. Zettlemoyer, and S. Gupta, "Muppet: Massive multi-task representations with pre-finetuning," 2021.
[59] W. Yuan, G. Neubig, and P. Liu, "BARTScore: Evaluating generated text as text generation," in *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021. [Online]. Available: https://openreview.net/forum?id=5Ya8PbvpZ9
[60] A. R. Fabbri, W. Kryscinski, B. McCann, R. Socher, and D. Radev, "Summeval: Re-evaluating summarization evaluation," *Transactions of the Association for Computational Linguistics*, vol. 9, pp. 391–409, 2021.
[61] "Sample size calculator," https://www.surveysystem.com/sscalc.htm, accessed: 2021-10-01.
[62] W. G. Cochran, *Sampling Techniques, 3rd Edition*. John Wiley, 1977.
[63] J. E. Montandon, C. Politowski, L. L. Silva, M. T. Valente, F. Petrillo, and Y.-G. Guéhéneuc, "What skills do it companies look for in new developers? a study with stack overflow jobs," *Information and Software Technology*, vol. 129, p. 106429, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584920301877
[64] M. Aniche, C. Treude, I. Steinmacher, I. Wiese, G. Pinto, M.-A. Storey, and M. A. Gerosa, "How modern news aggregators help development communities shape and share knowledge," in *Proceedings of the 40th International Conference on Software Engineering*, ser. ICSE '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 499–510. [Online]. Available: https://doi.org/10.1145/3180155.3180180
[65] J. Wang, L. Li, and A. Zeller, "Restoring execution environments of jupyter notebooks," in *Proceedings of the 43rd International Conference on Software Engineering*, ser. ICSE '21. IEEE Press, 2021, p. 1622–1633. [Online]. Available: https://doi.org/10.1109/ICSE43902.2021.00144
[66] T. Thongtan and T. Phienthrakul, "Sentiment classification using document embeddings trained with cosine similarity," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: Student Research Workshop*, 2019, pp. 407–414.
[67] D. S. Sachan, M. Zaheer, and R. Salakhutdinov, "Revisiting lstm networks for semi-supervised text classification via mixed objective function," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 6940–6948.
[68] Z. Yang, M. H. Asyrofi, and D. Lo, "Biasrv: Uncovering biased sentiment predictions at runtime," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2021. New York, NY, USA: Association for Computing Machinery, 2021, p. 1540–1544. [Online]. Available: https://doi.org/10.1145/3468264.3473117
[69] Z. Yang, H. Jain, J. Shi, M. Asyrofi, and D. Lo, "Biasheal: On-the-fly black-box healing of bias in sentiment analysis systems," in *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. Los Alamitos, CA, USA: IEEE Computer Society, oct 2021, pp. 644–648. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ICSME52107.2021.00073
[70] L. Dixon, J. Li, J. Sorensen, N. Thain, and L. Vasserman, "Measuring and mitigating unintended bias in text classification," in *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*, 2018, pp. 67–73.
[71] M. Hardt, E. Price, and N. Srebro, "Equality of opportunity in supervised learning," in *Advances in neural information processing systems*, 2016, pp. 3315–3323.
[72] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, "Machine learning testing: Survey, landscapes and horizons," *IEEE Transactions on Software Engineering*, no. 01, pp. 1–1, feb 5555. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/TSE.2019.2962027
[73] Z. Sun, J. M. Zhang, M. Harman, M. Papadakis, and L. Zhang, "Automatic testing and improvement of machine translation," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ser. ICSE '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 974–985. [Online]. Available: https://doi.org/10.1145/3377811.3380420
[74] Z. Q. Zhou and L. Sun, "Metamorphic testing of driverless cars," *Communications of the ACM*, vol. 62, no. 3, pp. 61–67, 2019.
[75] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid, "Deeproad: Gan-based metamorphic testing and input validation framework for autonomous driving systems," in *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2018, pp. 132–142.
[76] L. Sun and Z. Q. Zhou, "Metamorphic testing for machine translations: Mt4mt," in *2018 25th Australasian Software Engineering Conference (ASWEC)*. IEEE, 2018, pp. 96–100.
[77] M. H. Asyrofi, F. Thung, D. Lo, and L. Jiang, "Crossasr: Efficient differential testing of automatic speech recognition via text-to-speech," in *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2020, pp. 640–650.
[78] M. H. Asyrofi, Z. Yang, and D. Lo, "Crossasr++: A modular differential testing framework for automatic speech recognition," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2021. New York, NY, USA: Association for Computing Machinery, 2021, p. 1575–1579. [Online]. Available: https://doi.org/10.1145/3468264.3473124
[79] M. Asyrofi, Z. Yang, J. Shi, C. Quan, and D. Lo, "Can differential testing improve automatic speech recognition systems?" in *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. Los Alamitos, CA, USA: IEEE Computer Society, oct 2021, pp. 674–678. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ICSME52107.2021.00079
[80] Z. Zhao, D. Dua, and S. Singh, "Generating natural adversarial examples," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=H1BLjgZCb
[81] M. Iyyer, J. Wieting, K. Gimpel, and L. Zettlemoyer, "Adversarial example generation with syntactically controlled paraphrase networks," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, 2018, pp. 1875–1885.
[82] M. T. Ribeiro, S. Singh, and C. Guestrin, "Semantically equivalent adversarial rules for debugging nlp models," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2018, pp. 856–865.
[83] V. Prabhakaran, B. Hutchinson, and M. Mitchell, "Perturbation sensitivity analysis to detect unintended model biases," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019, pp. 5744–5749.