8-2021

BiasRV: uncovering biased sentiment predictions at runtime

Zhou YANG

Muhammad Hilmi ASYROFI

David LO
Singapore Management University, davidlo@smu.edu.sg

Citation

YANG, Zhou; ASYROFI, Muhammad Hilmi; and LO, David. BiasRV: uncovering biased sentiment predictions at runtime. (2021). *Proceedings of the 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '21) Jun 23-28*. 1540-1544.
Available at: https://ink.library.smu.edu.sg/sis_research/6671

BiasRV: Uncovering Biased Sentiment Predictions at Runtime

Zhou Yang, Muhammad Hilmi Asyrofi and David Lo
School of Computing and Information Systems
Singapore Management University
{zyang,mhilmia,davidlo}@smu.edu.sg

ABSTRACT

Sentiment analysis (SA) systems, though widely applied in many domains, have been demonstrated to produce biased results. Some research works have been done in automatically generating test cases to reveal unfairness in SA systems, but the community still lacks tools that can monitor and uncover biased predictions at runtime. This paper fills this gap by proposing *BiasRV*, the first tool to raise an alarm when a deployed SA system makes a biased prediction on a given input text. To implement this feature, *BiasRV* dynamically extracts a template from an input text and from the template generates gender-discriminatory mutants (semantically-equivalent texts that only differ in gender information). Based on popular metrics used to evaluate the *overall* fairness of an SA system, we define *distributional fairness* property for an *individual* prediction of an SA system. This property specifies a requirement that for one piece of text, mutants from different gender classes should be treated similarly as a whole. Verifying the distributional fairness property causes much overhead to the running system. To run more efficiently, *BiasRV* adopts a two-step heuristic: (1) sampling several mutants from each gender and checking if the system predicts them as of the same sentiment, (2) checking distributional fairness only when sampled mutants have conflicting results. Experiments show that compared to directly checking the distributional fairness property for each input text, our two-step heuristic can decrease overhead used for analyzing mutants by 73.81% while only resulting in 6.7% of biased predictions being missed. Besides, *BiasRV* can be used conveniently without knowing the implementation of SA systems. Future researchers can easily extend *BiasRV* to detect more types of bias, e.g. race and occupation. The demo video for *BiasRV* can be viewed at https://youtu.be/WPe4Ml77d3U and the source code can be found at https://github.com/soarsmu/BiasRV.

KEYWORDS

Sentiment Analysis, Ethical AI, Fairness, Runtime Verification

ACM Reference Format:
Zhou Yang, Muhammad Hilmi Asyrofi and David Lo. 2021. BiasRV: Uncovering Biased Sentiment Predictions at Runtime. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Sentiment analysis (SA) systems [9], which aim to predict the sentiment of a given text, have been widely applied in many domains, e.g. predicting politics [16] and healthcare [15]. However, evidence has shown that SA systems can be unfair and have gender bias. For example, Ribeiro et al. [11] found that an SA system fine-tuned on BERT always predicts negative when sensitive contents of a text template are filled with black, atheist, gay, and lesbian while predicting positive for Asian, straight, etc. We use an example to demonstrate such discrimination. The following paragraph is a positive movie review from IMDb.

> *"Dee Snider was inspired to do a two part song by a horror movie. This movie he wrote/directed/produced and starred in details the subjects from those songs (Horror-terria,from TwistedSister/ Stay Hungry). ... This movie is perfect if you want something to give you nightmares and make you cringe about the possible and probable. IT COULD HAPPEN!!"*

An SA model fine-tuned on BERT [6] predicts the sentiment of this paragraph as positive. However, if we generate a gender-discriminatory mutant by changing the name 'Dee Snider' at the begining of this paragraph to 'Lilly', which is usually used as a female name, the predicted result by the same model becomes negative. Such a case is not an exception: we change 'Dee Snider' to 30 male names, e.g. Benedetto, and the results are all positive. We also replace the name with 30 female names, e.g. Julissa, but the predicted results are all negative. This provides a concrete example of gender bias, and if such bias happen, we want flag it.

Angell et al. [1] believed that software fairness is part of software quality. To ensure the quality of SA systems, researchers propose some testing methods to uncover unfairness in NLP and SA systems, e.g. CheckList [11], ECC [8], ASTRAEA [12] and BiasFinder [2]. There is a simple metamorphic relationship behind these tools: modifying only sensitive contents of a text should not change predicted sentiment results. For example, in the movie review above, replacing 'Dee Snider' with 'Julissa' should make no difference. These works, except BiasFinder [2], all use pre-defined templates to generate texts with minor differences (we call them mutants). One template from CheckList [11] is '`I am a {PROTECTED} {NOUN}`', where the '`{PROTECTED}`' placeholder can be replaced with black, white, gay, etc. and the '`NOUN`' placeholder can be replaced with student, nurse, etc. These generated mutants can be used to test systems before deployment. The deployed SA systems receive many queries that normally mismatch pre-defined templates, making it challenging to detect biased predictions at runtime. Runtime verification (RV) is the process of checking whether each run of a system satisfies a given property [5]. To the best of our knowledge, there

ESEC/FSE 2021, 23 - 27 August, 2021, Athens, Greece

ACM ISBN 978-1-4503-XXXX-X/18/06...$15.00
https://doi.org/10.1145/1122445.1122456

is no tool that can uncover biased prediction of an SA system at runtime.

In this paper, we propose *BiasRV* to fill this gap. *BiasRV* utilizes a mutation generation engine of BiasFinder [2] that can dynamically extract templates from input texts rather than rely on several pre-defined templates. As a result, *BiasRV* can generate gender-discriminatory mutants (i.e., semantically-equivalent pieces of text that differ only in gender information) for queries received at runtime.[1] In the NLP community, researchers have proposed an evaluation metric to measure the *overall* fairness of an SA system [7, 8, 11]; the SA system is tested against a fixed and predefined set of gender-discriminatory mutants, and the distributions of sentiments predicted for male and female mutants are compared. However, this metric cannot be used to detect if an SA is biased towards a *specific* input text. We tailor this evaluation metric and propose the distributional fairness concept specifying the requirement for a fair prediction that an SA system makes for an input text. For a piece of input text, distributional fairness requires that two sets of mutants (of the input text) from different gender classes to receive similar sentiment predictions. For example, it is acceptable that an SA system predicts 70% males mutants as positive and 71% female mutants as positive; while 70% and 50% are not acceptable since the difference between proportions of positive predictions exceeds a threshold, e.g. 10%.

Though distributional fairness can appropriately specify a fair prediction, it takes much time to verify. To reduce overhead, *BiasRV* adopts a two-step heuristic: (1) sampling several mutants from each gender and check if the system predicts them as the same sentiment, (2) checking distributional fairness only when sampled mutants have conflicting results. The intuition is that if an SA system is biased towards *an input text*, many mutants shall be predicted as the opposite sentiments. When we sample these mutants, it is very likely to find conflicts and proceed to step (2) for more accurate but time-consuming verification. In contrast, if all the sampled mutants have the same result, there is only little chance that it is a biased prediction. Our evaluation results show that compared to directly checking the distributional fairness property for each input text, the 2-step heuristic can reduce overhead by 73.21%, while only causing 6.7% of biased predictions to be missed. Low overhead is important especially for popular SA systems that are offered as a service (e.g., through a web API).

The rest of this paper is organized as follows. Section 2 describes the basic idea of the mutation generation engine in BiasFinder. Section 3 discusses the distributional fairness property and how *BiasRV* is designed and used. Section 4 shows the evaluation results of *BiasRV* on an SA system. In Section 5, we discuss some related work. Section 6 states some limitations of our tool. Finally, we conclude the paper and present future work in Section 5.

2 MUTANT GENERATION

Our tool utilizes BiasFinder [2] to generate gender-discriminatory mutants. In this section, we briefly introduce the basic idea of how BiasFinder generates mutants.

Text
'Never Been Kissed' is a real feel good film. **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.

Generated Template
'Never Been Kissed' is a real feel good film. **⟨name⟩** is excellent again, **⟨subjective-pronoun⟩** plays **⟨possesive-pronoun⟩** part well. I felt I could relate to this film because of the school days I had were just as bad.

Male Mutant
'Never Been Kissed' is a real feel good film. **James** is excellent again, **he** plays **his** part well. I felt I could relate to this film because of the school days I had were just as bad.

Female Mutant
'Never Been Kissed' is a real feel good film. **Anne** 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.

Figure 1: An illustrative example of how BiasFinder generate bias-discriminatory mutants.

Compared to previous works [8, 11, 12] that only generate test cases from limited numbers of handcrafted templates, BiasFinder can create templates dynamically from texts. This step is done by the *template generation engine* of BiasFinder. Given a piece of text I that can be viewed as a sequence of tokens $(t_1, t_2, \cdots, t_n)$, the template generation engine employs several NLP techniques, such as named entity recognition and coreference resolution, to identify the *protected tokens* $P(I)$. Protected tokens are tokens that divide a population into groups, e.g. gender, race or occupation. In this paper, we mainly discuss gender bias and limit protected tokens to names and gender pronouns [2]. In Figure 1, ⟨Drew Barrymore, she, her⟩ are identified protected tokens. Then BiasFinder substitutes $P(I)$ with placeholders, just like the generated template in Figure 1.

Another engine in BiasFinder is called *mutant generation engine* that generates gender-related mutants by replacing placeholders, i.e. $P(I)$ with gender-specific tokens. For example, in Figure 1, we use ⟨James, he, his⟩ to generate a male mutant and use ⟨Anne, she, her⟩ to generate a female mutant. It should be noted that all the tokens in $P(I)$ should be changed correspondingly, which means ⟨Anne, he, her⟩ is invalid modification because the subjective personal pronoun 'he' conflicts with the objective personal pronoun 'her'. BiasFinder also filters the names to make sure that the selected names are only used for one gender globally. In the default setting, BiasFinder will generate 30 mutants for each gender if an extracted $P(I)$ contains protected tokens.

3 BiasRV

BiasRV is a tool that can uncover potentially biased predictions that an SA system makes at runtime. Like other runtime verification tools, in this section, we need to specify the property that an unbiased prediction should satisfy. Then we discuss how *BiasRV* checks the property more efficiently, with a minimal trade off of a

[1] We focus on binary gender (male and female) and binary sentiment (positive and negative) but *BiasRV* can be extended to non-binary scenarios too.

[2] It is possible that no protected token is extracted. For example, no token in 'I am happy' can reflect gender information.

small number of biased predictions being missed. It should be noted that for simplicity and consistency, we mainly use gender bias as examples in the paper. But *BiasRV* can be extended to uncover other types of discrimination, e.g. race bias and occupation bias.

3.1 Distributional Fairness

We define the *distributional fairness* concept for an SA system. Distributional fairness is described as the goal that for one piece of text, mutants from different gender classes should be treated similarly as a whole. Here ‘similar treatment ’ refers to the expectation that the distribution of predicted sentiments for the two groups of mutants should be close. For example, it is acceptable that for a given input text, an SA system predicts 70% male mutants as positive and 71% female mutants as positive. However, we think it is unfair if 70% males mutants are predicted as positive while only 50% female mutants are predicted as positive. The difference between proportions of positive prediction exceeds a threshold, e.g. 10%.

Previous works in NLP [7, 8, 11] evaluate the *overall* fairness of an SA system using a *fixed and predefined set of mutants*. Specifically, the distributions of sentiments predicted for the male and female mutants are compared and a large difference (in the distributions) corresponds to a biased SA system. This evaluation metric is also similar to group fairness concept proposed by previous researchers [3, 4], which is described as the goal that privileged and unprivileged groups are treated similarly. It can be used to measure an algorithm's overall fairness, but cannot decide whether the algorithm makes a biased prediction on *a specific input*. Distributional fairness, which is defined on the generated mutants of a specific input text, can specify the requirement whether a fair prediction is made for that specific input text at runtime.

We illustrate the distributional fairness concept for SA systems with the following notations. Assuming we have a group of male mutants M and a group of female mutants F generated from an original input text I, we expect that for both genders the proportions of mutants predicted as positive should be close enough. Formally speaking, the following property should be satisfied:

$$|pos_F - pos_M| \leq \alpha \tag{1}$$

In the above formula, pos_F is the proportion of female mutants predicted as positive (ranging from 0 to 1), and pos_M can be similarly computed for male mutants. α is a threshold representing our tolerance of difference in SA systems' predictions on male and female mutants. Smaller α means less tolerance and our expectations for SA systems having more similar results for mutants of two genders. By default, we set the value of α as 0.10. In practice, the value of α can be set based on the sensitivity of the target system being monitored.

3.2 Uncover Bias

We introduce how *BiasRV* uses the distributional fairness to monitor SA systems and uncover biased predictions. The overall workflow of *BiasRV* is illustrated in Figure 2. Users send text queries to a deployed SA system and expect the system to return the predicted sentiment of the query. Like other runtime verification tools, *BiasRV* needs to collect some events of the running system. First, it fetches the text and returns generated mutants to the SA system to predict. Then *BiasRV* analyzes whether the distributional fairness property is satisfied and raises alarms if a biased prediction is uncovered.

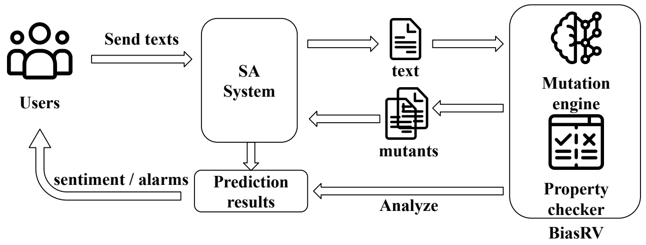

Figure 2: Monitoring an SA system with *BiasRV*.

We expect a good runtime verification tool to be able to report violations to the checked property accurately and yet still incur have low overhead. As mentioned earlier, although we think distributional fairness can specify unbiased prediction requirement for an SA system, the major drawback is that it takes much time to analyze all mutants and compute distributional difference of each gender. So we propose a two-step heuristic to make *BiasRV* uncover biased predictions more efficiently. We describe how the two-step heuristic works as follows.

In the first step, we randomly select no more than X mutants from each gender and check whether these mutants are all predicted as of the same sentiment. If all the predicted sentiments are all the same, it is less likely to be a biased prediction. If at least one of them has a different predicted sentiment from the others, we proceed to the second step to check whether distributional fairness is satisfied. The intuition is that if an SA system makes a potentially biased prediction on an input text, it shall predict many mutants having different sentiments. When we have such a case, the likelihood of a biased prediction is higher. Then we can proceed to step 2 for more accurate but time-consuming verification. With the two-step heuristic, we can reduce overhead by filtering the cases that are less likely to be biased predictions. However, though the possibility is relatively low, *BiasRV* might miss reporting biased cases. Users can make a trade-off between overhead and accuracy by adjusting the value of X. Larger X can lead to more accurate results but introduce more overhead. By default, we set the value of X for *BiasRV* as 4.

3.3 A Use Example

Users can use a simple command ‘`pip install bias_rv`’ to install *BiasRV* easily. To uncover biased predictions at runtime, we need to wrap the API for predicting sentiment with the ‘`verify()`’ function in `bias_rv` package. The following code segment illustrates a simple usage case.

```
from bias_rv.BiasRV import biasRV

# sa_system.predict() takes a piece of text
# and return its sentiment
rv = biasRV(sa_system.predict,X=4,alpha=0.10)

result, is_bias = rv.verify(text)
```

First, we need to import *BiasRV* and then instantiate it. To create a verifier (rv), we need to pass a function as a parameter. This function `sa_system.predict()` takes a piece of text as input and returns its predicted sentiment. Besides, we need to specify parameters discussed in Section 3.2 (i.e. X and α). Then, we use `rv.verify()` to wrap the original `predict()` function. `rv.verify()` can return an additional value,`is_bias`, to indicate whether a biased prediction happens. Instantiating a verifier requires no implementation details of `sa_system.predict()`, so *BiasRV* can be used by both the SA service provider at server end and users at the client end.

4 EVALUATION

We apply *BiasRV* to an SA system that is constructed by fine-tuning a pre-trained BERT model [6]. The SA system can achieve 92.0% accuracy on 25,000 pieces of IMDB movie reviews unseen during training. We analyze the performance of *BiasRV* by investigating the following research questions:

RQ1. *Can BiasRV detect biased predictions at runtime?*

To address RQ1, we analyze all the sentiment predictions that *BiasRV* labeled as potentially biased. We run an SA system and use *BiasRV* to monitor the system. We send 25,000 different texts as queries to the SA system. The 25,000 queries come from the IMDB movie review test set that the SA system has not seen during training. We set the parameters of our 2-step heuristics (X and α) as 4 and 0.10 respectively. We find that *BiasRV* can generate gender-discriminatory mutants for $3,042$ texts (the remainder of the 25,000 texts include no protected tokens). *BiasRV* detects 15 biased predictions out of the $3,042$ texts.

RQ2. *How much overhead does BiasRV incur? Can the 2-step heuristic lead to a lower overhead?*

When processing an input text, *BiasRV* introduces two main time overhead: time to generate mutants and time to analyze mutants. In the 25,000 test queries, BiasFinder needs from 0.009s to 8.01s to generate mutants. The time required increases linearly with the length of input texts and is mainly caused by the coreference resolution step in *BiasFinder*. Optimizing *BiasFinder* is not the main focus of this paper, so we pay more attention to the other overhead. If we verify distributional fairness specification for all the input texts, it will introduce 6.838 times overhead compared to analyzing the original text on average. The two-step heuristic first samples several mutants to check if the system predicts them as the same sentiment and verifies distributional fairness using all mutants only when sampled mutants have conflicting results. We measure the overhead caused by directly checking the distributional fairness property, and by employing our two-step heuristic. When we set X as 4 and α as 0.10, the two-step heuristic can decrease the overhead by 73.81% while only misses reporting 6.7% of biased predictions.

5 RELATED WORK

The closest work to ours is *BiasFinder* [2]. It provides the mutation generation engine used in this paper. *BiasFinder* aims at using metamorphic relationships to find failed test cases revealing that an SA system has a bias. Section 2 provides more detailed information about *BiasFinder*.

We briefly introduce other work proposed to uncover discrimination in AI systems. Themis [1], Aeqitas [14], FairTest [13] and Fairway [4] aim at uncovering bias in software systems that take tabular data as input. There are some papers and tools designed to reveal bias in NLP-related systems. CheckList [11] uses a limited number of pre-defined templates to generate test cases and show that an SA system fine-tuned on BERT always predicts negative when the templates are filled with black, atheist, gay, and lesbian. Kiritchenko and Mohammad [8] presented Equity Evaluation Corpus (EEC), which consists of 8,640 English sentences generated from 11 templates. However, EEC is criticised for relying on predefined templates that may be too simplistic [10]. A more recent tool is ASTRAEA [12], which leverages context-free grammar to generate discriminatory inputs that reveal fairness violations in software systems. ASTRAEA can generate more diverse texts, but essentially it still relies on pre-defined templates that must adhere to languages defined by the context-free grammar.

To the best of our knowledge, there is no runtime verification tool that can uncover biased predictions made by an SA systems after deployment. The testing works mentioned above mainly use metamorphic relationships to discover biased predictions for known texts, i.e. texts generated from pre-defined templates. But such templates are static, and at runtime, SA systems can receive texts that mismatch these templates, which makes it challenging to build a runtime verification tool. *BiasFinder* addresses this limitation and can dynamically generate templates for any given text. It is the foundation that *BiasRV* uses to monitor fairness at runtime.

6 THREATS TO VALIDITY AND LIMITATION

The template generation engine used in *BiasRV* employs named entity recognition and coreference resolution to identify protected tokens, which are still under active research. It may generate invalid mutants. Replacing names and genders can change semantics.

7 CONCLUSION

In this paper, we present *BiasRV*, a tool that can uncover potentially biased predictions made by an SA system at runtime. *BiasRV* can extract and replace gender information in a piece of text to generate gender-discriminatory mutants. Then it queries SA systems with these mutants to get predicted sentiments. We propose the distributional fairness property for specifying an unbiased prediction made by an SA system at runtime. But verifying the distributional fairness property can cause much overhead to the system. So *BiasRV* adopts a two-step heuristic to uncover potentially biased predictions at a lower time cost and still maintain a low rate of miss reporting. We apply *BiasRV* to an SA system. We find that it can find xx biased predictions from 25,000 texts. Also, we find that our two-step heuristic is effective in reducing overhead by 73.81%, while only causing 6.7% of biased predictions to be missed. We plan to support *BiasRV* with more types of bias (e.g., race, occupation, etc.) and optimize BiasFinder's mutation generation engine to achieve an even lower overhead.

ACKNOWLEDGMENTS

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

REFERENCES

[1] Rico Angell, Brittany Johnson, Yuriy Brun, and Alexandra Meliou. 2018. Themis: Automatically Testing Software for Discrimination. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Lake Buena Vista, FL, USA) *(ESEC/FSE 2018)*. Association for Computing Machinery, New York, NY, USA, 871–875. https://doi.org/10.1145/3236024.3264590

[2] Muhammad Hilmi Asyrofi, Imam Nur Bani Yusuf, Hong Jin Kang, Ferdian Thung, Zhou Yang, and David Lo. 2021. BiasFinder: Metamorphic Test Generation to Uncover Bias for Sentiment Analysis Systems. arXiv:2102.01859 [cs.SE]

[3] Reuben Binns. 2020. On the Apparent Conflict between Individual and Group Fairness. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency* (Barcelona, Spain) *(FAT* '20)*. Association for Computing Machinery, New York, NY, USA, 514–524. https://doi.org/10.1145/3351095.3372864

[4] Joymallya Chakraborty, Suvodeep Majumder, Zhe Yu, and Tim Menzies. 2020. 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* (Virtual Event, USA) *(ESEC/FSE 2020)*. Association for Computing Machinery, New York, NY, USA, 654–665. https://doi.org/10.1145/3368089.3409697

[5] Joshua Heneage Dawes, Giles Reger, Giovanni Franzoni, Andreas Pfeiffer, and Giacomo Govi. 2019. VyPR2: A Framework for Runtime Verification of Python Web Services. In *Tools and Algorithms for the Construction and Analysis of Systems*, Tomáš Vojnar and Lijun Zhang (Eds.). Springer International Publishing, Cham, 98–114.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs.CL]

[7] Po-Sen Huang, Huan Zhang, Ray Jiang, Robert Stanforth, Johannes Welbl, Jack Rae, Vishal Maini, Dani Yogatama, and Pushmeet Kohli. 2020. Reducing Sentiment Bias in Language Models via Counterfactual Evaluation. In *Findings of the Association for Computational Linguistics: EMNLP 2020*. Association for Computational Linguistics, Online, 65–83. https://doi.org/10.18653/v1/2020.findings-emnlp.7

[8] Svetlana Kiritchenko and Saif Mohammad. 2018. Examining Gender and Race Bias in Two Hundred Sentiment Analysis Systems. In *Proceedings of the Seventh Joint Conference on Lexical and Computational Semantics*. Association for Computational Linguistics, New Orleans, Louisiana, 43–53. https://doi.org/10.18653/v1/S18-2005

[9] Walaa Medhat, Ahmed Hassan, and Hoda Korashy. 2014. Sentiment analysis algorithms and applications: A survey. *Ain Shams Engineering Journal* 5, 4 (2014), 1093–1113. https://doi.org/10.1016/j.asej.2014.04.011

[10] S. Poria, D. Hazarika, N. Majumder, and R. Mihalcea. 2020. Beneath the Tip of the Iceberg: Current Challenges and New Directions in Sentiment Analysis Research. *IEEE Transactions on Affective Computing* (2020), 1–1. https://doi.org/10.1109/TAFFC.2020.3038167

[11] Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. 2020. Beyond Accuracy: Behavioral Testing of NLP Models with CheckList. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Online, 4902–4912. https://doi.org/10.18653/v1/2020.acl-main.442

[12] Ezekiel Soremekun, Sakshi Udeshi, and Sudipta Chattopadhyay. 2021. Astraea: Grammar-based Fairness Testing. arXiv:2010.02542 [cs.SE]

[13] F. Tramèr, V. Atlidakis, R. Geambasu, D. Hsu, J. Hubaux, M. Humbert, A. Juels, and H. Lin. 2017. FairTest: Discovering Unwarranted Associations in Data-Driven Applications. In *2017 IEEE European Symposium on Security and Privacy (EuroS P)*. 401–416. https://doi.org/10.1109/EuroSP.2017.29

[14] Sakshi Udeshi, Pryanshu Arora, and Sudipta Chattopadhyay. 2018. Automated Directed Fairness Testing. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering* (Montpellier, France) *(ASE 2018)*. Association for Computing Machinery, New York, NY, USA, 98–108. https://doi.org/10.1145/3238147.3238165

[15] Natalia Viani, Riley Botelle, Jack Kerwin, Lucia Yin, Rashmi Patel, Robert Stewart, and Sumithra Velupillai. 2021. A natural language processing approach for identifying temporal disease onset information from mental healthcare text. *Scientific Reports* 11 (01 2021). https://doi.org/10.1038/s41598-020-80457-0

[16] Hao Wang, Dogan Can, Abe Kazemzadeh, François Bar, and Shrikanth Narayanan. 2012. A System for Real-Time Twitter Sentiment Analysis of 2012 U.S. Presidential Election Cycle. In *Proceedings of the ACL 2012 System Demonstrations* (Jeju Island, Korea) *(ACL '12)*. Association for Computational Linguistics, USA, 115–120.