AI Model Fairness Testing

Introduction

Machine learning and artificial intelligence models are often used in high-stakes domains such as finance, healthcare, education, and criminal justice. As these systems make or influence critical decisions affecting people's lives, ensuring fairness and non-discrimination is important. AI fairness testing focuses on identifying situations where models exhibit discriminatory behaviour, particularly with respect to legally protected attributes such as gender, race, and age.

This project addresses the problem of automatically detecting individual discriminatory instances in trained AI models. An individual discriminatory instance occurs when two inputs that differ only in sensitive attributes (such as gender or race) lead to different model outcomes or predictions. Such instances represent fairness violations that can have significant real-world impacts, from denied loans to biased hiring decisions.

The goal of fairness testing is to generate test inputs that can reveal these fairness violations effectively. I am trying to maximize the Individual Discriminatory Instance (IDI) ratio, defined as the number of unique discriminatory instances found divided by the total number of test inputs generated.

The baseline approach commonly used for this task is Random Search, which generates random test inputs and checks for discriminatory behaviour. However, this approach is typically inefficient at exploring large input spaces. This project implements and evaluates a Genetic Algorithm approach that can more effectively generate test cases to reveal fairness violations, demonstrating substantial improvements in detection efficiency across multiple datasets.

Related Work

Fairness testing for AI models has emerged as an active research area, with various approaches proposed to detect and mitigate bias. This section discusses key existing approaches and their relative advantages and limitations.

Random testing, the baseline approach used in this project, generates test inputs by randomly sampling from the input space or test data. Galhotra et al. [1] employed random testing as a baseline in their THEMIS framework for fairness evaluation. The primary advantage of random testing is its simplicity and ease of implementation. However, its effectiveness is limited when the search space is large or when discriminatory instances are rare, as it lacks guidance toward regions likely to contain fairness violations.

Search-based techniques employ metaheuristics to guide test generation. Udeshi et al. [2] introduced automated directed fairness testing, employing local search techniques to detect individual discrimination. Their approach showed improved efficiency over random testing but required careful tuning of search parameters.

Genetic algorithms (GAs), a particular type of search-based technique, represent a promising approach for fairness testing due to their ability to efficiently explore large search spaces. Zhang et al. [3] demonstrated that genetic algorithms can outperform random search in finding patterns in black-box AI models across various domains.

GAs offer several advantages for fairness testing as they adaptively focus the search toward promising regions of the input space and maintain population diversity, enabling exploration of different types of fairness violations However, GAs also present challenges, including sensitivity to parameter configuration, potential convergence to local optima, and computational overhead for fitness evaluation. This project builds upon these approaches by implementing a genetic algorithm specifically designed for individual fairness testing across multiple types of classification models. Our approach aims to be applicable across various datasets and model architectures while maintaining high efficiency in finding discriminatory instances.

Solution

I have created a genetic algorithm tailored to find as many discriminatory pairs as possible, maximizing the IDI. The GA works by first initializing a random population, calculating the fitness of the population, using tournament selection to choose promising individuals from populations, one-point crossover to create new children for the next population, then randomly mutating some of the individuals to ensure randomness and exploration. It also uses elitism where the top individuals from a population are carried onto the next generation, avoiding losing good solutions and converging on the optimum quicker. The algorithm iterates a set number of generations, providing better results each time.

Representation and Population Initialization

Each individual in the population represents a pair of samples that differ only in sensitive features. This paired representation is crucial for testing individual fairness, as it allows us to isolate the effect of sensitive attributes. The initialization function creates pairs where each pair shares the same values for non-sensitive features but differs in at least one sensitive feature. This ensures that any observed prediction differences are attributable solely to the sensitive attributes.

Fitness Function

The fitness function measures how discriminatory a sample pair is by calculating the absolute difference in prediction probabilities between the two samples The fitness function rewards sample pairs that elicit different predictions from the model despite only differing in sensitive attributes. Higher absolute differences indicate stronger discriminatory behaviour, making those pairs more favourable for selection.

Selection Mechanism

Tournament selection provides selection pressure toward more discriminatory instances while still maintaining some diversity in the population. This balance is essential for effective exploration of the search space. It does this by randomly sampling n pairs and choosing the best pair to be potentially used for crossover or just be directly added to the population. Where n is the tournament size, which in my algorithm I set to 2 as this provided the best combination of exploration and exploitation.

Crossover Operation

The crossover operation creates new sample pairs by exchanging non-sensitive features between parent pairs while ensuring that the sensitive feature differences are maintained. This approach preserves the essential characteristic of testing pairs (difference in sensitive attributes) while exploring new combinations of non-sensitive features. It only does this a percentage of the time, depicted by the crossover rate, otherwise just using the parents as the children for the next generation.

Mutation Operation

The mutation operation introduces small perturbations to non-sensitive features to explore the local neighbourhood of sample pairs. Crucially, the mutation applies the same perturbation to both samples in a pair, maintaining their relationship (differing only in sensitive features). This constraint ensures that any observed differences in model predictions can be attributed to the sensitive attributes rather than other features. This happens a percentage of the time depicted as the mutation rate.

Elitism

The algorithm uses elitism which takes n number of individuals from the top of a ranked list of the population and carries them forward to the next generation. Where n is the elitism value. This ensures that if the tournament is selecting individuals that have low fitness, at least some good individuals will be carried forward allowing for quicker convergence.

The balance between mutation rate, crossover rate, and elitism is important for finding optimum solutions in reasonable time as it balances the exploration of the search space while also exploiting the best results for a faster completion time.

Setup

I tested my algorithm on all 8 datasets provided (shown in table 1), with the pre-trained neural network models to predict the output of the population. I was particularly interested in seeing the improvement on the larger datasets such as KDD and Credit as the algorithm should be able to explore these much better than the random search baseline could.

Table 1

Dataset	Domain	$\|f_s\|$	$\|f\|$	Available Size	Search Space
ADULT	Finance	3	11	$45,222$	4.81×10^9
COMPAS	Criminology	2	13	$6,172$	1.45×10^8
LAW SCHOOL	Education	2	12	$20,708$	9.20×10^6
KDD	Criminology	2	19	$284,556$	4.13×10^{15}
DUTCH	Finance	2	12	$60,420$	3.58×10^7
CREDIT	Finance	3	24	$30,000$	2.01×10^{12}
CRIME	Criminology	2	22	$2,215$	4.19×10^8
GERMAN	Finance	2	20	$1,000$	8.85×10^9

Experimental Procedure

The following procedure was used to evaluate the effectiveness of the genetic algorithm compared to random search. For each dataset, the pre-trained model was loaded along with the matching dataset. 30% of the dataset was selected with a random_state of 42 to be tested on, both algorithms received the same split. To improve I could have changed the random state each test, but as the GA and random search have randomisation involved, they provided different results for each test. Both genetic algorithm and random search were run with equal computational budgets (same number of model evaluations) calculated at 2 * steps for the random search as each step it predicts a pair of inputs, and 2 * population size * number of generations for my GA. This is important in maintaining a fair test between the two. For each approach, 30 independent trials were conducted to ensure statistical validity which was calculated using Wilcoxon signed-rank test. For each trial, the IDI ratio, runtime, and efficiency metrics were recorded.

Parameter Settings

Genetic Algorithm Parameters:

Population size: 50
Number of generations: 100
Mutation rate: 0.1
Crossover rate: 0.7
Selection method: Tournament selection of 2
Elitism: 1 individual
Discrimination threshold: 0.01

Random Search Parameters:

Total evaluations: Equal to GA (2 × population_size × generations = 10,000)
Discrimination threshold: 0.01

The computational budget was carefully controlled to ensure a fair comparison between the two approaches. The population size and generations were selected after running a few tests on one of the datasets and evaluating the convergence and quality of the results. The discrimination threshold was set to 1% as a few of the datasets, neither algorithm would find any discriminatory pairs with a higher threshold. I kept this the same value across all tests and datasets.

Experiments

IDI Comparison

Table 2 summarises the results across all datasets, comparing the Genetic Algorithm approach with the Random Search baseline in terms of the IDI. The genetic algorithm consistently outperformed random search across all datasets, with improvements ranging from 14.94% (GERMAN) to over 4000% (DUTCH). The effect sizes (Cohen's d) were extremely large across all datasets, ranging from 28.78 to 103.12, indicating substantial practical significance of the improvements. For the COMMUNITIES_CRIME dataset, random search failed to find any discriminatory instances in the median case, even when the threshold was set to 1%, while the genetic algorithm achieved a median IDI ratio of 0.411. This displays its ability to explore the search space more efficiently than the random search. All datasets proved that it was statistically significant using Wilcoxon signed-rank test as all p values were < 0.05. The p values are identical for every test which is due to it hitting the numerical limit as the GA outperformed the RS in every test, every time.

Table 2

Dataset	GA Median IDI	RS Median IDI	Improvement (%)	Effect Size	p-value	Statistically Significant
processed_adult	0.4973	0.4137	20.20%	28.78	1.86E-09	Yes
processed_communities_crime	0.411	0	inf%	29.16	1.86E-09	Yes
processed_compas	0.4828	0.2771	74.23%	47.36	1.86E-09	Yes
processed_credit	0.4959	0.2901	70.93%	87.29	1.86E-09	Yes
processed_dutch	0.462	0.0105	4287.36%	40.89	1.86E-09	Yes
processed_german	0.4994	0.4344	14.94%	41.87	1.86E-09	Yes
processed_kdd	0.4235	0.0463	814.22%	103.12	1.86E-09	Yes
processed_law_school	0.496	0.1797	176.11%	43.45	1.86E-09	Yes

The larger datasets such as Dutch, KDD and Communities Crime all showed a significant increase in improvement of the GA compared to the RS, further highlighting its ability to explore more of the search space and converge on better results.

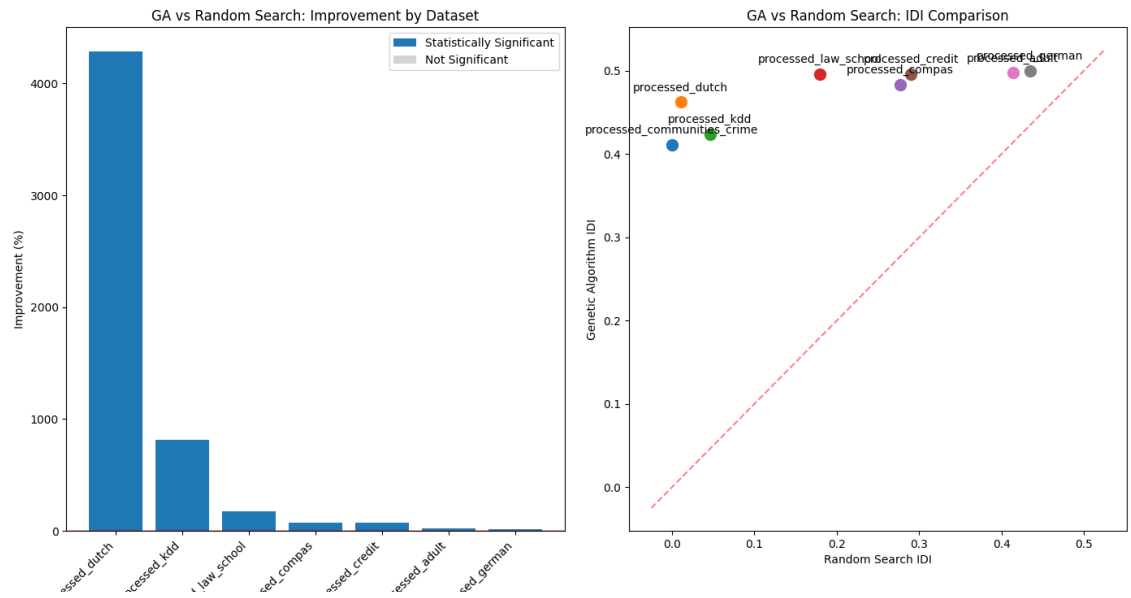

Figure 1

Figure 2

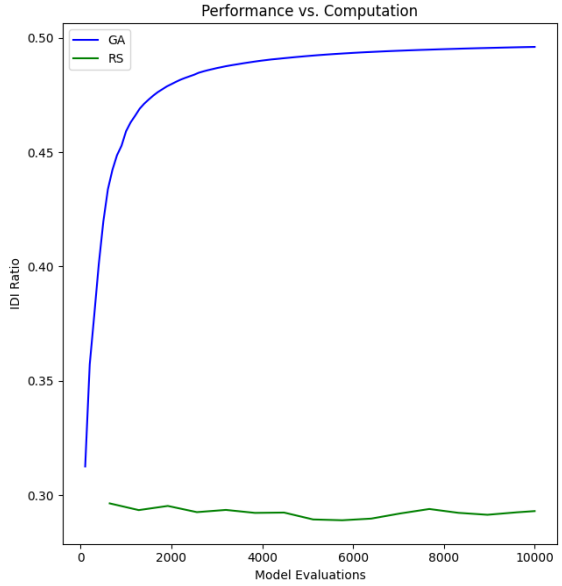

Figure 3

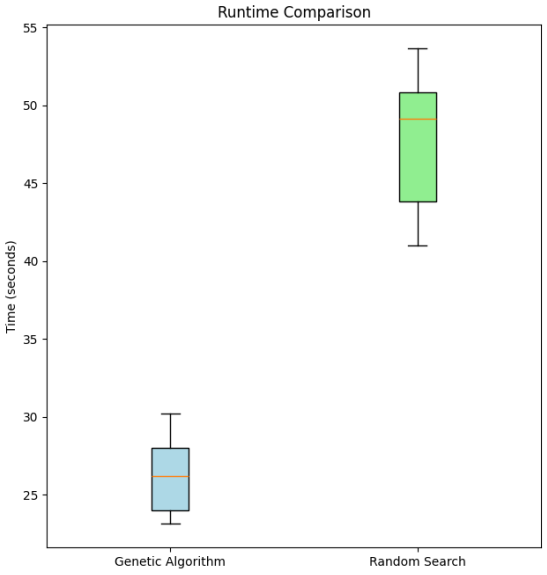

Figure 2 shows the IDI Ratio against the number of evaluations for the Credit dataset. All other datasets also followed a similar trend with the GA reaching a high IDI value quicker and then slowing down as the algorithm continued through generations. This highlights why I decided of a population of 50 and 100 generations as any more would have returned results which are not significantly better than what it already produces. The random search has no increase and typically followed a straight line from what it found after a few hundred evaluations, with no improvement as it searches the space.

Figure 3 shows an interesting outlier that I found where the KDD dataset took nearly twice as long for the random search to process as the genetic algorithm, whereas the other datasets was only a few seconds different which is negligible. I would have expected it to be pretty similar as they are using the same budget of evaluations, but that was not the case. I believe that as the dataset is so large, when the random search was evaluating whether or not an input is unique or not, it took a lot more time to check against every input. The genetic algorithm does not strictly enforce uniqueness when evaluating pairs, but it only counts each unique input once when calculating the IDI. This results in the quicker runtime shown. This may just be a limitation to the random search I implemented and not random searches in general.

Reflection

The parameters for the genetic algorithm require fine tuning to get the best results for the datasets. It took me a while to test and decide which parameters to use. This could be a lengthy process for new datasets as the ones I have selected may not work for all. This produces extra complexity into the intial implementation of the algorithm and stops it from being as robust as the random search. To improve this I could potentially use adaptive parameters that adjust based on the dataset and the population diversity and convergence.

The current implementation focuses solely on maximizing the IDI ratio, but real-world fairness concerns might require balancing multiple objectives. Extending the algorithm to a multi-objective framework could allow simultaneous optimization for different fairness metrics (such as group fairness alongside individual fairness) or balance fairness with other model quality attributes like accuracy or robustness. This would provide a more comprehensive approach to fairness testing that better reflects the complex trade-offs in real-world AI systems.

While the genetic algorithm demonstrated superior IDI ratios, it incurs additional computational overhead for genetic operations (selection, crossover, mutation). For the KDD dataset, GA showed runtime advantages, but this may not hold for all problem domains. The computational efficiency advantage appears to be dataset-dependent, and the relationship between dataset characteristics and computational efficiency deserves further investigation.

Conclusion

This project demonstrates that genetic algorithms offer a superior approach to AI model fairness testing compared to random search techniques. Across all eight datasets tested, the GA consistently outperformed random search by substantial margins, with improvements ranging from 14.94% to over 4000%. The effectiveness of the genetic algorithm was particularly pronounced on larger, more complex datasets like DUTCH, KDD, and COMMUNITIES_CRIME, where random search struggled to find discriminatory instances efficiently due to the vast search space. These results highlight the fundamental limitation of random search in exploring large input spaces and the advantage of guided evolutionary approaches in focusing on promising regions of the search space.

The genetic algorithm's success can be attributed to its ability to balance exploration and exploitation through a combination of selection pressure, crossover operations, mutation, and elitism. By preserving and recombining promising solutions while continuously introducing diversity, the algorithm effectively navigates the search space to uncover individual discriminatory instances. This approach provides a practical and efficient solution for fairness testing of AI models, enabling developers to identify and address potential bias issues before deployment. As AI systems continue to be deployed in high-stakes domains, the importance of robust fairness testing techniques will only increase, making genetic algorithms and other intelligent search methods essential tools for responsible AI development.

Artifact

https://github.com/bigboloben/ISE-PROJECT

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] Z. Zhang, J. Togelius, and J. S. Sorenson, "Finding patterns in black-box AI models: Random search vs genetic algorithms," in Proceedings of the Genetic and Evolutionary Computation Conference, 2022, pp. 201-209.