Verifying Individual Fairness in Machine Learning Models

Philips George John, Deepak Vijaykeerthy, Diptikalyan Saha*
IBM Research AI
Bengaluru 560 045, India

Abstract

We consider the problem of whether a given decision model, working with structured data, has *individual fairness*. Following the work of Dwork, a model is individually biased (or unfair) if there is a pair of valid inputs which are *close* to each other (according to an appropriate metric) but are treated differently by the model (different class label, or large difference in output), and it is unbiased (or fair) if no such pair exists. Our objective is to construct verifiers for proving individual fairness of a given model, and we do so by considering appropriate relaxations of the problem. We construct verifiers which are sound but not complete for linear classifiers, and kernelized polynomial/radial basis function classifiers. We also report the experimental results of evaluating our proposed algorithms on publicly available datasets.

1 INTRODUCTION

Recent breakthroughs in artificial intelligence, especially machine learning, have lead to AI-based systems assuming a significant role in making real-world decisions — such as decision-making systems for recidivism risk assessments, credit assessments (including loan risk), hiring decisions, content dissemination in social media etc. Many of these systems are trained on, and then evaluate, *structured data* pertaining to individuals (convicts, borrowers, job candidates etc). Unfortunately, studies have already shown that such systems may be prone to discriminating against users/consumers on the basis of characteristics such as race and gender (Angwin et al. 2016), and this has even lead to legal mandates to ensure *fairness* in such systems.

This paper tackles the problem of verifying the absence of individual bias in a given classifier (with *white-box* access) which takes *structured data* as input. The definition of individual fairness/bias that we use in this paper is based on the abstract definition of individual fairness given by (Dwork et al. 2012), which says that a model f is fair if, for any pair of inputs x, x' which are sufficiently *close* (as per an appropriate metric), the model outputs $f(x), f(x')$ are also close (as per another appropriate metric). Since using an ℓ_p-metric for "closeness" does not take into account the structure of the data, we define a more flexible scheme which we feel is appropriate for our modality. We partition the input attributes into subsets, each of which is associated with an appropriate non-negative threshold, and say that two points x, x' are sufficiently close if the coordinate-wise absolute difference, $|x_i - x'_i|$, for each attribute, is at most the threshold ε_j corresponding to the subset S_j to which the attribute index i belongs. Then a model is *fair* if, for any pair of close inputs, the model outputs are also close; for classification models, this means that the model decision does not change (i.e., $f(x) = f(x')$), whereas for regression models, this means that the absolute difference $|f(x) - f(x')|$ is sufficiently small. We note that this scheme is flexible enough to allow for (i) each attribute to have an independent threshold (as one extreme) including a threshold of zero for no perturbations, or (ii) the same threshold for all attributes (the other extreme), which is the same as using the ℓ_∞-metric for closeness (as is usual in adversarial robustness).

Another definition of individual fairness was used in (Aggarwal et al. 2019; Galhotra et al. 2017; Udeshi et al. 2018) in the context of testing rather than verification. It is a simplified and non-probabilistic form of *causal* or *counterfactual fairness* (Kusner et al. 2017), based on the notion of *protected* or *sensitive* attributes (in practical scenarios, these may be gender, race/ethnicity,

*{pgeorg04,deepakvij,diptsaha}@in.ibm.com

Proceedings of the 36th Conference on Uncertainty in Artificial Intelligence (UAI), PMLR volume 124, 2020.

religion etc.). The definition of *fairness* in this context is that any two valid inputs which differ *only on* the protected attribute(s) must always be put in the same class. Our definition subsumes this definition, by considering the threshold for the protected attributes to be sufficiently large (to allow arbitrary perturbations), and the threshold for the non-protected attributes to be zero (to disallow perturbations).

Challenges. The challenges of verifying individual fairness, when compared to the existing work on verifying machine learning models, are two-fold: Firstly, the existing work on verifying bias/fairness in machine learning models considers notions of *group fairness/bias* (Albarghouthi et al. 2017; Bastani et al. 2019). An individual fairness property considers the worst case (fairness *for all* similar input pairs, biased if *there exists* a bad input pair), rather than the average case (with high probability, some notion of parity is maintained between different groups) considered in the group fairness definitions. Hence, *the existing techniques for group fairness cannot be applied to verifying individual fairness*. Secondly, the other work on verification of ML models (which mostly considers the verification of *adversarial robustness* – see (Liu et al. 2019) for a survey) considers a *local robustness* property; the verifier is given a *nominal input* and it verifies robustness in the neighbourhood of that particular input (for example, given a particular image, the verifier either certifies that a small ℓ_∞-norm perturbation of that image does not change the class label, or provides a counter-example). However, verification of individual fairness notions requires us to check a *global robustness* property (i.e. the classifier output does not change for perturbations of *any* input in the domain). This means that *existing approaches to local robustness verification are not directly applicable to our problem*.

Our contributions. To the best of our knowledge, we present the first technique for individual fairness verification (global robustness) for ML models.

We give a meta-algorithm/framework for solving the verification problem, as well as particular algorithms for linear classifiers, and kernelized classifiers with polynomial/rbf kernels. Our algorithms are sound but incomplete (see section 4.1), with the linear classifier case being an exception in that it is exact (both sound and complete) if we allow for worst-case exponential time.

2 RELATED WORK

In recent times, the software engineering community has addressed the problem of testing Individual fairness. THEMIS (Galhotra et al. 2017) used random testing to generate test cases. AEQUITAS (Udeshi et al. 2018) used random testing for global search and performs perturbation close to a sample which showed discrimination. (Aggarwal et al. 2019) used a combination of symbolic execution and model explainability techniques to systematically explore the decision space in a model instead of random testing. None of the above techniques guarantees absence of individual bias.

Previous work on the verification of fairness in machine learning models has considered notions of group fairness such as disparate impact (Albarghouthi et al. 2017; Bastani et al. 2019). There have also been works considering the verification of the adversarial robustness property (and other similar properties) for machine learning models. Robustness does not have any notion of protected attributes. However, from an algorithmic perspective, our work is related to these, albeit using a different metric than the usual ℓ_p-balls, and considering global robustness rather than local (as in the survey (Zhang et al. 2019)).

These verification approaches can be broadly classified into (i) verification using tailor-made satisfiability modulo theory (SMT) or mixed integer linear programming (MILP) based approaches, and (ii) verification using convex relaxations. The former approach leads to sound and complete verification of certain classes of machine learning models — such as linear models, decision trees (including tree ensembles), neural networks with piecewise linear activation functions etc. (Katz et al. 2017; Ehlers 2017; Bunel et al. 2018; Tjeng et al. 2019), but at the cost of a worst-case exponential (or super-exponential) running time. The latter approach (convex relaxations) leads to efficient verifiers for properties such as (local) adversarial robustness for machine learning models (Kolter et al. 2018; Dvijotham et al. 2018; Raghunathan et al. 2018; Gowal et al. 2018; Singh et al. 2018; Wang et al. 2018a; Wang et al. 2018b; Wang et al. 2018c; Zhang et al. 2018; Gehr et al. 2018; Mirman et al. 2018; Qin et al. 2019; Fazlyab et al. 2019; Salman et al. 2019; Singh et al. 2019), but at the cost of sacrificing completeness. Another advantage is that such techniques can also be used for non-linear models, even though much of the existing work focuses on linear and piece-wise linear models.

Perhaps the work most closely related to ours (from a technical perspective) is that of (Raghunathan et al. 2018), who verify the (local) adversarial robustness of a given feed-forward neural network with ReLU activations by relaxing adversarial robustness to a polynomial optimization problem and then using semidefinite relaxations to give lower bounds.

3 PRELIMINARIES

We consider models with a known *prediction function* $f : \mathbb{R}^n \to \mathbb{R}$. We assume that we have white-box access to f; i.e. we have access to all the parameters and hyper-parameters which together give a closed-form expression for f. A regression model uses the output of f directly as the predicted value of the regression variable. A binary classifier $h : \mathbb{R}^n \to \{\pm 1\}$ is of the form $h(x) = \text{sign}(f(x))$. We will often refer to the prediction function f as the classifier itself, with the understanding that the predicted label for x will actually be the sign of $f(x)$. We assume that f is *smooth*, or at least continuously differentiable twice (C^2).

The model f takes as input a real vector with n features, where the domain of feature x_i is $\mathsf{Dom}_i := \{x \in \mathbb{R} \text{ or } \mathbb{Z} \mid l_i \leq x \leq u_i\}$. That is, each feature can be either continuous (in $\mathbb{R}$) or discrete (in $\mathbb{Z}$), and takes values in a fixed interval $[l_i, u_i]$. This characterization of input features is suitable for us since, in this work, we are chiefly considering decision models (classification/regression) on structured data. We say that an input sample is *valid* if the domain constraints for all features are satisfied.

If $f : \mathbb{R}^n \to \mathbb{R}$ is a decision model, the abstract definition of individual fairness, given by (Dwork et al. 2012), is as follows: Given appropriate distance functions — $d(\cdot, \cdot)$ on $\mathbb{R}^n$ (the domain of f) and $D(\cdot, \cdot)$ on $\mathbb{R}$ (the co-domain of f) — as well as thresholds $\varepsilon \geq 0$ and $\delta \geq 0$, the model is individually fair if, for any pair of inputs x, x' such that $d(x, x') \leq \varepsilon$, we have $D(f(x), f(x')) \leq \delta$.

The intuition behind this notion of individual fairness is that *small* or *non-significant* perturbations of a sample x to x' (i.e. the perturbations where $d(x, x') \leq \varepsilon$) must not be treated "differently" by a fair model. The choice of the input distance function $d(\cdot, \cdot)$ identifies the perturbations to be considered non-significant, while the choice of the output distance function $D(\cdot, \cdot)$ limits the changes allowed to the perturbed output in a fair model.

For classification models $f : \mathbb{R}^n \to [k]$, it is appropriate to use the discrete metric $D(y, y') := \mathbb{I}[y = y']$ with the threshold $\delta = 0$ on the model output since, in a fair classification model, we would want to prevent any change in the class label due to small perturbations of the input. For regression models, a simple choice would be the absolute error $D(y, y') := |y - y'|$, with the threshold $\delta > 0$ chosen appropriately based on the scale of the regression variable.

Our notion of closeness in the input domain must take into account the structure of the data, and be general enough to give non-trivial and useful results (fairness certificates/bias instances) for a variety of structured datasets and models. So we proceed as follows. Let the input features be indexed as $[n] := \{1, \ldots, n\}$. We partition $[n]$ into disjoint sets $S_1, \ldots, S_t$, with corresponding thresholds $\varepsilon_1, \ldots, \varepsilon_t \geq 0$ chosen based on domain-specific knowledge of the dataset. A perturbation of $x \in \mathbb{R}^n$ to x' is considered non-significant if for all $j \in [t]$, and for all $i \in S_j$, we have $|x_i - x'_i| \leq \varepsilon_j$.

Note: For notational convenience, we allow the thresholds ε_j to take a special value ∞, which implies that x_i and x'_i can differ arbitrarily for all $i \in S_j$. From a algorithmic perspective, the constraint on $|x_i - x'_i|$ would be removed for all such indices i. Also, if $\varepsilon_j = 0$ for any $j \in [t]$, we can eliminate the variables x'_i for all $i \in S_j$ to reduce the dimensionality of the problem.

We formally define the individual bias of a decision model as follows:

Definition 1 (Individual bias)**.** *A model $f : \mathbb{R}^n \to \mathbb{R}$ is said to be* individually biased *if there exists a pair of valid inputs x and x', with $|f(x) - f(x')| > \delta$, such that $|x_i - x'_i| \leq \varepsilon_j$ for all $i \in S_j$, and for all $j = 1, \ldots, t$. Such a pair (x, x') is called an individual bias instance of the model f.*

A *linear (binary) classifier* is $h(x) = \text{sign}(f(x))$, where f is of the form $f(x) = w^\top x + b$ for some $w \in \mathbb{R}^n$ and $b \in \mathbb{R}$. The decision boundary of a linear classifier is an affine hyperplane. A linear regression model is just $f(x) = w^\top x + b$, without the sign function.

A *kernelized (binary) classifier* is $h(x) = \text{sign}(f(x))$, where f is of the form $f(x) = \sum_{i=1}^{M} w_i y_i K(x_i, x)$, $S := \{(x_i, y_i) : i = 1, \ldots, M\}$ is a subset of the training set, w_i is the weight assigned to the sample $(x_i, y_i) \in S$, and $K(\cdot, \cdot)$ is the Kernel function.

One of the commonly used kernels is the degree-d polynomial kernel, where $K(x, y) = (ax^\top y + b)^d$ for some $a, b \in \mathbb{R}$. The prediction function of such a classifier $f(x) = \sum_{i=1}^{M} w_i y_i (ax_i^\top x + b)^d$ can then be written as a degree-d polynomial in the variables $x_1, \ldots, x_n$. A polynomial kernel allows for a "curved" non-linear decision boundary rather than the "straight" hyperplane boundary of a linear classifier.

Another commonly used kernel function is the radial basis function (RBF, also known as gaussian) kernel, with $K(x, y) = exp(-\gamma \cdot \|x - y\|^2)$. In an RBF kernelized classifier, high confidence negative decision regions can be seen around the negative data points, whereas high confidence positive decision regions can be seen around the positive data points.

A set $S \subseteq \mathbb{R}^n$ is said to be a *convex set* if for any two points $x, y \in S$ and any $0 \leq \lambda \leq 1$, the point $\lambda x + (1 -$

$\lambda)y \in S$. If S is a convex set, a function $f : S \to \mathbb{R}$ is said to be a *convex function* if for any $x, y \in S$ and $0 \leq \lambda \leq 1$, $f(\lambda x+(1-\lambda)y) \leq \lambda f(x)+(1-\lambda)f(y)$. On the other hand, a function is said to be *concave* if we have the inequality $f(\lambda x + (1-\lambda)y) \geq \lambda f(x) + (1-\lambda)f(y)$ for all $x, y \in S$ and $0 \leq \lambda \leq 1$.

Notation $[n]$ denotes the set $\{1, 2, \ldots, n\}$. If $x \in \mathbb{R}^n$, x_{-i} denotes the tuple $(x_j)_{j \neq i} \in \mathbb{R}^{n-1}$ and for $S \subseteq [n]$, x_S is shorthand for the tuple $(x_i)_{i \in S}$. If S is a set, then $\binom{S}{k}$ denotes the set of all subsets $T \subseteq S$ with $|T| = k$. If $|S| = n$, then $\left|\binom{S}{k}\right| = \binom{n}{k} = \Theta(n^k)$. We use $\mathbb{R}^{m \times n}$ to denote the space of all $m \times n$ real matrices, and $\mathbb{S}^n(\mathbb{R})$ to denote the space of all real symmetric $n \times n$ matrices. If f is a function, $f \succeq 0$ denotes that f is *non-negative* or positive semi-definite (p.s.d); that is, $f(x) \geq 0$ for all x in the domain of f. If A is a matrix, then $A \succeq 0$ denotes that the matrix is positive semi-definite (p.s.d). We use $\mathbb{N}^n_d := \{\alpha \in \mathbb{N}^n \mid |\alpha| := \alpha_1 + \ldots + \alpha_n \leq d\}$ to denote the set of multi-indices corresponding to the exponents of n-variate monomials with degree $\leq d$.

4 VERIFYING INDIVIDUAL BIAS

4.1 A META-ALGORITHM

Ideally, a *verifier* for the individual bias property would solve the following decision problem (after fixing the attribute domains $\mathsf{Dom}_1, \ldots, \mathsf{Dom}_n$, the feature partitions $S_1, \ldots, S_t$, and the thresholds $\varepsilon_1, \ldots, \varepsilon_t$ and δ): "For a given model f, does there exist an individual bias instance?" (YES/NO). This problem would be *undecidable*, by Rice's theorem (Hopcroft et al. 1990), if we allow f to be an arbitrary partial recursive function. Even if we restrict the choice of f to common non-linear function classes, the problem would still likely be NP-hard (since we show that it is NP-hard when f is a polynomial with $\deg(f) \geq 2$).

We focus on the following guarantees for the verifier $\mathcal{V}$ (with our desired property for the model f being individual fairness/no bias):

1. **Soundness:** If the verifier $\mathcal{V}$ outputs NO BIAS, then the model f is actually unbiased.
2. **Completeness:** If the verifier $\mathcal{V}$ outputs a bias instance, then the model f is actually biased. Also, the verifier $\mathcal{V}$ will *always* terminate with either NO BIAS or a valid bias instance.

To circumvent the hardness of exact (sound and complete) verification, we try to get verifiers with *soundness*, but *not completeness*. That is, an output of NO BIAS by the verifier V will always be *correct* (f will actually be unbiased). But $\mathcal{V}$ *may* fail to terminate within finite time with the correct output NO BIAS or a bias instance, even when the classifier is actually unbiased or a valid bias instance exists, respectively. It may also keep finding only spurious bias instances, which we *do not* output.

For developing our algorithms, we formulate the individual bias verification problem as a (non-convex) optimization problem, and use provably-correct global optimization approaches (such as mixed integer linear programming) to perform the verification.

The rationale behind this approach is as follows: If we wish to find a bias instance x, x', the required "closeness" constraints $|x_i - x'_i| \leq \varepsilon_j$ (for all $i \in S_j$, for all $j \in [t]$) on x and x' are *linear constraints* (since $|z| \leq \varepsilon \iff -\varepsilon \leq z \leq \varepsilon$), which are easy to handle in an optimization framework (perhaps the only easier form of constraint is an interval constraint $z \in [\alpha, \beta]$). The domain constraints on each x_i are of the form $x_i, x'_i \in [l_i, u_i] \cap (\mathbb{R} \text{ or } \mathbb{Z})$. A problem is that the integrality constraint for discrete attributes is computationally expensive, but this can be mitigated to some extent by the judicious use of relaxations.

We can then say that the model f is individually biased if and only if an input pair x^*, x'^* with $|f(x^*) - f(x'^*)| > \delta$ belongs to the set of pairs x, x' constrained as above. In fact, since x and x' are interchangeable, we can assume $f(x^*) - f(x'^*) < -\delta$ for a bias instance. With all these considerations, we can formulate an optimization problem for individual bias verification.

$$\begin{aligned} D^* := \min\ & f(x) - f(x') \\ \text{s.t. } & |x_i - x'_i| \leq \varepsilon_j,\ \forall i \in S_j,\ \forall j \in [t] \\ & x_i, x'_i \in [l_i, u_i] \cap (\mathbb{R} \text{ or } \mathbb{Z}),\ \forall i \in [n] \end{aligned} \tag{1}$$

If the verifier solves the optimization problem (1) and finds a solution (x^*, x'^*) with objective function value $D^* < -\delta$, then that solution will be a valid individual bias instance that the verifier can output. On the other hand, if the verifier finds a *certifiable lower bound* which implies $D^* \geq -\delta$, then the verifier can correctly output NO BIAS. However, the requirement of certifiable lower bounds precludes the use of many common optimization techniques based on gradient descent or interpolation.

Relaxation To avoid the hardness of the integral constraints, it is possible to relax *some* of the categorical features to allow fractional values. This makes sense especially for features such as *age*; which take a large ordered set of values, and where a fractional value is "interpretable". It is easy to see that this relaxation preserves the NO BIAS certification; that is, a classification model is actually unbiased w.r.t. the original attribute do-

mains if it is unbiased w.r.t. the relaxed domains, since we are only expanding the set of valid inputs for bias instances. But this approach may generate spurious counterexamples which we can only reject, and then continue to try to find valid counterexamples. In addition to the domain relaxations, we may also choose to use relaxations of the optimization problem that end up yielding non-tight lower bounds (without valid counterexamples). In general, we may end up doing this indefinitely without actually finding a valid counterexample (even if it exists) — thus, a verifier that uses a relaxation will not be complete. The specific relaxations that we use in each case will be discussed in later sections.

The key idea of our work is to demonstrate that we can use the optimization approach described above (with appropriate relaxations and solution methods) to solve the individual bias verification problem for some interesting and useful classes of models, under reasonable assumptions. The details of this, as well as specific techniques and methods, will be given in subsequent sections for each type of classifier that we consider, i.e. linear, kernelized polynomial, and RBF. We first give a general meta-algorithm for the individual bias verification problem (see Algorithm 1).

The intuition behind this meta-algorithm is just combining the optimization problem formulation (1) and domain relaxations. We allow the user to choose a *subset* D of the discrete attributes, which should take small sets of values (e.g. boolean attributes). The optimization procedure will fix x_D and x'_D to specific value combinations using equality constraints, and repeat for all such feasible value combinations. The discrete attributes which are not in D *may be (not necessarily)* relaxed to take fractional values. In this way, we can find a set of lower bounds ℓ for $f(x) - f(x')$ which we examine to see if there is any possibility of bias ($\ell < -\delta$). The details of the optimization procedure will vary depending on the type of f (no optimization procedure can solve such a problem in a certifiably optimal way for general f), and will be described in later sections.

Note that if no domain relaxations are used *and* the optimization procedure is guaranteed to always find tight bounds (e.g. mixed integer linear programming), then the resulting verifier will be sound and complete. Otherwise, the resulting verifier will be sound but incomplete (e.g. sum-of-squares).

Algorithm 1 A meta-algorithm for individual bias verification.

procedure VERIFY-INDIVIDUAL-BIAS(f, $(S_j)_{[t]}$ $(\varepsilon_j)_{[t]}$, δ, D, $(\mathsf{Dom}_i)_{[n]}$)
  Input: Classification model f (white-box), discrete attributes D, feature partitioning $S_1, \ldots, S_t$, thresholds $\varepsilon_1, \ldots, \varepsilon_t$ and δ, attribute domains $\mathsf{Dom}_1, \ldots, \mathsf{Dom}_n$.
  Output: Either (i) Valid bias instance (x, x') or (ii) NO BIAS.
  Let $L = \emptyset$.
  Construct the set V_p:

$$V_p := \{(v, v') \mid v, v' \text{ are feasible for } x_D,\ x'_D \text{ and } |v_i - v'_i| \leq \varepsilon_j\ \forall i \in D \cap S_j\ \forall j \in [t]\}$$

  for all $(v, v') \in V_p$ **do**
    Let:

$$\begin{aligned} D^* := \min\ & f(x) - f(x') \\ \text{s.t. } & |x_i - x'_i| \leq \varepsilon_j,\ \forall i \in S_j \cap \overline{D},\ \forall j \in [t] \\ & x_i, x'_i \in [l_i, u_i],\ \forall i \notin D \\ & x_D = v \text{ and } x'_D = v' \end{aligned}$$

    Find a lower bound $\ell \leq D^*$.
    If $\ell < -\delta$, try to find a certificate x^*, x'^* for the lower bound ℓ (i.e. $f(x^*) - f(x'^*) = \ell$), which may not always exist.
    Add (ℓ, x^*, x'^*) to L.
  if $\ell \geq -\delta$ for all lower bounds in L **then**
    Output NO BIAS.
  else if There exists a lower bound $\ell < -\delta$ with a valid certificate in L **then**
    Output bias instance (x^*, x'^*).

4.2 LINEAR AND POLYNOMIAL MODELS

4.2.1 Linear Models

We first give an elementary instantiation of the meta-algorithm for linear models, $f(x) = w^\top x + b$. This is the only case where the optimization problem involved is actually convex, and thus the relaxed problem (with continuous attributes) can be solved with soundness and completeness in polynomial-time. In all other cases, the optimization problem is non-convex.

If f is a linear (affine) regression model of the form $f(x) = w^\top x + b$, the objective function of the problem (1), $f(x) - f(x') = w^\top x - w^\top x'$, is linear. The constraints are also linear ($|x_i - x'_i| \leq \varepsilon_j$ can be replaced by the pair of linear constraints $x_i - x'_i \leq \varepsilon_j$, $x'_i - x_i \leq \varepsilon_j$), with integrality constraints for the categorical features. Hence the problem (1) can be solved as a mixed-integer linear program (MILP). MILP solvers can solve the problem *exactly*, modulo computational issues, with worst-case exponential time, and are also fairly efficient in

practice for reasonable problem dimensions ($= 2n$ in this case). Thus, we get a sound and complete verifier.

Suppose $f(x) = \text{sign}(g(x))$, $g(x) = w^\top x + b$, is a linear classification model. Now, the objective function is no longer linear, but since we use $\delta = 0$ for classification, we can take advantage of the fact that $f(x) - f(x') = 0 \iff g(x) \cdot g(x') \geq 0$ to get a quadratic objective function for minimization. Since $g(x) \cdot g(x') = x^\top w w^\top x' + b \cdot (w^\top x + w^\top x') + b^2$, we can rewrite it in the form $g(x) \cdot g(x') = (x\ x')^\top Q (x\ x') + b \cdot (w\ w)^\top (x\ x') + b^2$, where Q is a positive semi-definite quadratic form in $2n$ variables $(x\ x')$. Hence the problem can be solved *exactly* by mixed-integer quadratic programming (MIQP) solvers, again practically efficient but with worst-case exponential time, to yield a sound and complete verifier.

4.2.2 Kernelized Polynomial Models

In a kernelized classification/regression model with a *polynomial kernel*, the kernel function used is of the form $K(x, y) = (a\, x^\top y + b)^d$, where $a, b \in \mathbb{R}$ are constants and d is the degree of the polynomial kernel. Then,

$$f(x) = \sum_{i=1}^{M} w_i y_i\, K(x_i, x) = \sum_{i=1}^{M} w_i y_i\, (a\, x_i^\top x + b)^d$$

That is, the model f can be viewed as a degree-d polynomial in the variables $x_1, \ldots, x_n$. Thus, the function we wish to lower bound, $f(x) - f(x')$, is also a degree-d polynomial in $2n$ variables. Let $g(x, x') := f(x) - f(x')$. Minimizing g over a linear constraint set ($l_i \leq x_i, x_i' \leq u_i$ for all i, $x_i = v_i, x_i' = v_i'$ for all $i \in D$, $-\varepsilon_j \leq x_i - x_i' \leq \varepsilon_j$ for all $i \in S_j$ and for all j), as in Algorithm 1, is a polynomial optimization problem over a basic closed semi-algebraic set. If we have discrete attributes which we do not fix during optimization (i.e. not in D), but which we do not want to relax, this can be done using polynomial constraints as well. A constraint $x(x_i - 1) \cdots (x_i - k) = 0$ would ensure that x_i takes values in $\{0, 1, \ldots, k\}$. The drawback is that such constraints will be very expensive computationally unless k is very small (e.g. $k = 2$ for boolean values).

We can find lower bounds for such polynomial optimization problems using various methods, including sum-of-squares relaxations, geometric programming etc. In this paper, we consider the method of finding lower bounds for polynomial optimization problems using sum-of-squares relaxations (proposed independently by Lasserre and Parrilo, based on the earlier work of Shor (Shor 1987)). We use the particular semidefinite programming (SDP) relaxation from (Lasserre 2015) to solve our optimization problem. To the best of our knowledge, this is the first work where sum-of-squares is applied to verify global robustness properties for ML models.

We now give some intuition about the sum-of-squares method for polynomial optimization. The problem is to minimize the given polynomial function g subject to polynomial inequality constraints. The s.o.s algorithm in this case tries to find the largest real number γ such that the shifted polynomial $g - \gamma$ can be written as a specific type of polynomial (a polynomial in the *quadratic module* (Lasserre 2015) generated by the constraint polynomials). Finding this polynomial can be thought of as finding a vector of monomial coefficients which satisfies appropriate semi-definite constraints. To keep the optimization problem finite-dimensional (i.e. the number of monomials is finite), we have to put an upper-bound d on the degree of this polynomial. This results in the degree-d sum of squares relaxation. With some additional assumptions (always satisfied in our setting), a non-trivial theorem in real algebraic geometry (Putinar's Positivstellensatz (Lasserre 2015)) then guarantees that γ is a lower bound for the polynomial g on the feasible set.

Suppose g^* is the actual tight lower bound for g (i.e. it is achieved by some point which is our certificate/minimizer). However, when the chosen relaxation degree d is too small, we might not be able to represent $g - g^*$ as a polynomial of the required form, and the best degree $\leq d$ representation obtained by the optimizer will give a worse (smaller) lower bound. In this case, the verification algorithm must increase d successively to find better lower bounds. It is known that for all such polynomial optimization problems $\mathcal{P}$ and any $\varepsilon > 0$, there will be some finite $d_{\mathcal{P},\varepsilon}$ such that the degree-$d_{\mathcal{P},\varepsilon}$ relaxation lower bound will be ε-close to the actual lower bound (convergence of s.o.s), but there are no known upper bounds for $d_{\mathcal{P},\varepsilon}$ (to the best of our knowledge). Hence an s.o.s-based verification algorithm remains incomplete (unlike the MILP/MIQP approaches), even in cases where all integrality constraints are applied (no domain relaxations), due to this non-zero gap in the s.o.s lower bounds.

We omit a full description of the sum-of-squares relaxations, but the book by Lasserre (Lasserre 2015) is a good reference which includes all the details and proofs. A brief technical description of the relaxations (*sans* proofs) is given in the supplementary material.

Using the sum-of-squares relaxations, and the fact that semidefinite programs can be solved up to exponential accuracy in polynomial time (w.r.t the number of variables and constraints of the SDP), we get the following:

Theorem 1. *There is a polynomial-time algorithm (which runs in time $n^{O(2d)}$) that outputs $\ell \pm O(1/2^n)$, where ℓ is a lower bound for $g(x, x') := f(x) - f(x')$,*

subject to a linear constraint set as in Algorithm 1.

The $\pm O(1/2^n)$ error comes from the error in solving semi-definite programs, which cannot be avoided even when using exact arithmetic (since SDPs with rational coefficients need not have rational solutions).

This implies that we have a sound but incomplete algorithm to solve the relaxed individual bias verification problem for polynomial kernel classifiers, by plugging in the sum-of-squares relaxation algorithm from Theorem 1 into the meta-algorithm (Algorithm 1) from Section 4.1.

The salient points of our technique for polynomial kernelized classifiers are give below:

- We use the sum-of-squares relaxation technique to find a lower-bound approximation of $g(x, x')$.
- The lower-bound approximation ensures that when our verifier is always correct when it says NO BIAS. However, it can yield spurious counter-examples / loose lower bounds at any particular level (of relaxation degree).
- The approximations are refined by taking sum-of-squares relaxations of higher degree, which makes the s.o.s lower bound closer to the actual lower bound of $g(x, x')$.
- This refinement process may not terminate (only a convergence result is known), yielding an incomplete verifier.

4.3 RBF Kernelized Classifiers

The Radial Basis Function (RBF) kernel is of the form $K(x, y) = \exp(-\gamma\|x - y\|_2^2)$, for a fixed parameter γ. So the kernelized classifier is of the form

$$f(x) = \sum_{i \in \mathcal{S}^+} \varphi_i(x) - \sum_{i \in \mathcal{S}^-} \varphi_i(x)$$

where $\varphi_i(x) := w_i \exp(-\gamma\|x - x_i\|_2^2)$. We use $\mathcal{S}^+$ to denote the subset of indices i with $y_i = 1$, and $\mathcal{S}^-$ to denote those with $y_i = -1$. We will abuse the notation to write $x_i \in \mathcal{S}^+$ and $i \in \mathcal{S}^+$ as appropriate, and similarly for $\mathcal{S}^-$. Suppose that all the non-zero model weights satisfy $0 < c < w_i < C$ for some bounds c and C. Let $g(x, x') := f(x) - f(x')$.

Let $\epsilon > 0$ be a very small constant (compared to c). Suppose that, when searching for bias instances, we only want to find pairs x, x' where $g(x, x') < -2\varepsilon$. Then, we can completely avoid looking at regions with $|f(x)| \leq \varepsilon$ and $|f(x')| \leq \varepsilon$ (by triangle inequality). We can directly exploit the above fact when $\delta > 0$ (with $\varepsilon = \delta/2$) since our desired condition for bias is $g(x, x') < -\delta$. Things are not as straightforward when $\delta = 0$ (with classification models), but we argue that, under reasonable assumptions, we can fix a sufficiently small (but non-zero) ε, say $\varepsilon = 10^{-8}$, such that finding a lower bound $g(x, x') \geq -2\varepsilon$ rather than $g(x, x') \geq 0$ does not exclude any valid and *interesting* bias instances. Specifically, the assumption is that we consider the non-zero model weights ($w_i > c$) to not be too small compared to ε, and we are not interested in bias instances where the attribute values have small positive magnitude of the order of ε (such attribute values are unlikely to occur in real world data), and based on the bounds that we specify, we are not considering points x which are very far from all the support vectors. Hence we can argue that, for a very small $\varepsilon > 0$, we have $|f(x)| \geq \varepsilon$ for all valid x which we wish to consider as a bias instance.

Theorem 2. *Given a kernelized classifier (with the RBF kernel) $f : \mathbb{R}^n \to \mathbb{R}$ with M support vectors, and for a fixed, sufficiently small, $\varepsilon > 0$, there is an algorithm* FIND-BIAS-RBF *which runs in time* $\mathsf{poly}(n, M)$ *that either (i) returns a bias instance x, x' with $g(x, x') := f(x) - f(x') < -2\varepsilon$, or (ii) returns a lower-bound $L \geq -2\varepsilon$ such that, for all valid data points [1], we have $g(x, x') \geq L$.*

We mention that we chose $\varepsilon = 10^{-8}$ in our experiments based on the above considerations, because the datasets we used had $\approx 10 - 20$ features, and we decided (rather unilaterally) that bias instances with feature values $\ll 10^{-3}$ were not interesting.

We now give a sketch of both the algorithm and the proof of correctness. A detailed pseudocode is left for the supplementary material. We note that this algorithm (FIND-BIAS-RBF) is slightly different compared to the meta-algorithm in Section 4.1. However, it can still be plugged into the meta-algorithm to verify individual bias of a given kernelized RBF classifier. The main differences are: (i) In the case of finding a valid bias instance (x, x'), the algorithm FIND-BIAS-RBF does not give an exact lower bound ℓ as in the meta-algorithm (we can only say that $\ell < -2\varepsilon$), and (ii) The case where the FIND-BIAS-RBF algorithm gives a lower bound $\geq -2\varepsilon$ is treated as the NO BIAS case (which differs from the meta-algorithm when $\delta = 0$). But we still get a correct output in case (i), and have already justified that a very small ε will make sure that all valid examples are covered in case (ii); with only minor assumptions on the minimum magnitudes of the model weights and the attribute values.

The intuition behind the FIND-BIAS-RBF algorithm is as follows. A kernelized RBF classifier f is a linear combination of n-dimensional gaussian densities, where for each support vector x_i (with label y_i), you have a gaus-

[1] Assuming that we do not have valid feature values as small as $\approx n\sqrt{\varepsilon}$.

sian with mean (i.e. centered at) x_i with covariance matrix $\frac{1}{2\gamma}I$. This gaussian is scaled by a factor $0 < w_i < C$, which is equivalent to scaling the variance of each (i.i.d.) co-ordinate by $w_i^2 < C^2$. Then, we can write

$$\varphi_i(x) = \mathcal{N}\left(x_i, \frac{w_i^2}{2\gamma}I\right)(x)$$

where we again abuse the notation and use $\mathcal{N}(\mu, \Sigma)$ to denote the gaussian probability density function (pdf) with mean μ and covariance matrix Σ.

Now, for any x, x' such that $f(x) < -\varepsilon$ and $f(x') > \varepsilon$ (which would imply that $g(x, x') < -2\varepsilon$), there must exist a support vector $x_r \in \mathcal{S}^+$ such that $\varphi_r(x') \geq \varepsilon/M$, and a support vector $x_s \in \mathcal{S}^-$ such that $\varphi_s(x) \geq \varepsilon/M$. That is, any individual bias instance (pair of inputs) that we care about (w.r.t. our fixed ε) must be in the intersection of $\mathcal{B}_{\ell_2}(x_r, D_r)$ and $\mathcal{B}_{\ell_2}(x_s, D_s)$ for some support vectors $x_r \in \mathcal{S}^+$ and $x_s \in \mathcal{S}^-$, where $D_r := \sqrt{\frac{1}{\gamma}\log\left(\frac{Mw_r}{\varepsilon}\right)}$ (and D_s is defined similarly), which means that it must also be in the intersection of $\mathcal{B}_{\ell_\infty}(x_r, D_r)$ and $\mathcal{B}_{\ell_\infty}(x_s, D_s)$. We then minimize the objective function $P(x, x') = \frac{1}{2}\big(\sum_{u\in\mathcal{S}^+} w_u\|x' - x_u\|^2 + \sum_{v\in\mathcal{S}^-} w_v\|x - x_v\|^2\big)$, subject to the linear constraints that $x, x' \in \mathcal{B}_{\ell_\infty}(x_r, D_r) \cap \mathcal{B}_{\ell_\infty}(x_s, D_s)$.

It is easy to see that minimizing $P(x, x')$ subject to these constraints (for a particular pair x_r and x_s) is a convex quadratic program, which can be solved in $\mathsf{poly}(n)$ time. This has to be done for all x_r and x_s. This requires $\leq M^2$ iterations, and so the entire algorithm runs in $\mathsf{poly}(n, M)$ time. If no appropriate bias instance is found in these iterations, we output the lower bound $L \geq -2\varepsilon$, which is the smallest value of $g(x^*, x'^*)$ among those found in each iteration.

5 EXPERIMENTAL RESULTS

5.1 SETUP

All our experiments are carried out on a cloud virtual machine with 32 Intel Xeon E5-2683 v4 (2.10 GHz) processors, 128 GB RAM and no dedicated GPU. The machine runs Ubuntu 16.04, and has Python 3.6.8 (Anaconda) installed, along with all the default Anaconda packages. Each experiment is run with at most 32 parallel jobs (`python` processes) on this machine.

Tools We use the `cplex` and `quadprog` Python packages to solve the quadratic programs, and `SDPA` to solve the sum-of-squares relaxation SDPs.

Table 1: Experimental results of the proposed algorithms for the relaxed problem.

DS	Model	Accuracy		Bias	Time taken
		Train	Test		
GC	GC_Linear1	0.767	0.76	Yes	226.2s
	GC_Linear2	0.756	0.748	No	229.0s
	GC_Poly1	0.650	0.704	No	78.4m
	GC_Poly2	0.658	0.7	No	77.3m
	GC_Rbf1	0.992	0.664	Yes	9.5s
	GC_Rbf2	1.0	0.7	No	27.7m
AD	AD_Linear1	0.822	0.821	Yes	47.3s
	AD_Linear2	0.823	0.821	No	48.7s
	AD_Poly1	0.826	0.824	Possible	10.4s
	AD_Poly2	0.826	0.823	Possible	10.9s
	AD_Rbf1	0.828	0.827	Yes	114.5s
	AD_Rbf2	0.905	0.811	Yes	546.8s
FD	FD_Linear1	0.660	0.654	Yes	0.163s
	FD_Linear2	0.660	0.654	No	0.101s
	FD_Poly1	0.602	0.610	Possible	18.1s
	FD_Poly2	0.590	0.607	Possible	17.1s
	FD_Rbf1	0.928	0.669	Yes	16.1s
	FD_Rbf2	1.0	0.665	Yes	32.7s
CR	CR_Linear1	0.943	0.86	Yes	106.8s
	CR_Linear2	0.953	0.88	No	79.1s
	CR_Poly1	0.906	0.93	No Bias	33.8s
	CR_Poly2	0.863	0.9	No Bias	31.8s
	CR_Rbf1	1.0	0.85	Yes	9.2s
	CR_Rbf2	1.0	0.56	No	389.4s

Table 2: Experimental results of testing bias using random sampling.

Model	Time Taken	Result
GC_Lin1	1.812 mins	Not found
GC_Lin2	2.088 mins	Not found
GC_Poly1	1.756 mins	Not found
GC_Poly2	1.646 mins	Not found
GC_Rbf1	4.039 mins	Not found
GC_Rbf2	4.737 mins	Not found

5.2 BENCHMARKS

5.2.1 Datasets

We use four publicly available datasets to benchmark our algorithms, listed in Table 3. Note that the Adult dataset which we use is a publicly available dimension-reduced variant (*Modified Adult Dataset (DiCE)*) of the full dataset. In all cases, we report the time taken for the individual bias verification step on an already trained model. The times which we report are obtained from the Python `time.perf_counter()` function.

In all the datasets, we replace textual categorical attributes by appropriate integer values as appropriate. We also we do a $75\% - 25\%$ stratified split on each dataset before using it for training.

Table 3: The datasets used for experiments

Dataset	# Features	# Rows (Train)
German Credit (GC)	20	750
Adult (AD)	8	24420
Fraud Detection (FD)	9	825
Credit ISLR (CR)	10	300

For the perturbation bounds, we use a counterfactual formulation where a subset of the attributes in each dataset are selected as protected/sensitive, with arbitrary perturbations allowed ($\varepsilon_1 = \infty$), and the rest of the attributes are fixed ($\varepsilon_2 = 0$). The protected attributes are: sex-marital-status for German Credit, race for Adult, ethnicity for Fraud Detection, and gender, ethnicity for Credit.

5.2.2 Models

All the models are trained using the `scikit-learn` framework. The linear models are trained using scikit-learn logistic regression, with L_2 regularization and the default parameters. The DD_Linear2 models are trained after setting the protected attribute values to 0 throughout the training data (masking). The rbf kernelized models are trained using the support vector machine classifier (`sklearn.svm.SVC`) with the rbf kernel. The DD_Rbf1 models are trained with $C = 1000$ and $\gamma = 10^{-4}$. The DD_Rbf2 models are trained with $C = 1$ and $\gamma = 0.5$, and after masking the protected attribute. The polynomial kernelized models are trained using `SVC` with the degree-2 polynomial kernel. The DD_Poly1 models are trained with $C = 1.$, $\gamma = 0.001$, and $r = 0$. The DD_Poly2 models are trained with the same hyperparameters, but after masking the protected attribute.

5.3 EVALUATION OF VERIFICATION

We show the results of evaluating our proposed verification algorithms on the models described above and show the results in Table 1. It can be seen that the time taken for verification — even for reasonably complex models on real-world datasets — is within fairly acceptable limits, even with the worst case exponential time. It can also be seen that the sum-of-squares relaxations scale quite badly as the dimension of the data increases, which is as expected.

Wherever Table 1 shows Bias = Yes, it indicates that the verifier gave a valid bias instance as the output. Bias = No indicates that the verifier proved No Bias. Bias = Possible indicates that the verifier did not prove a lower-bound ≥ 0, but neither did it find a valid bias instance. An example of a bias instance found for the model AD_Lin1 is (age: 60, work: Private, edu: Bachelors, marital-status: Married, occupation: Professional, race: White, sex: Male, hours-per-wk: 8). The model predicts the income as $\geq 50k$. Changing race from "White" to "Other" flips the income prediction to $< 50k$.

5.4 COMPARISON WITH TESTING

We performed an experiment by running the random testing algorithm (THEMIS) with 50,000 samples and verification algorithm to compare the time taken by the verification algorithm and testing for finding counter-example and determine that our algorithm provides NO BIAS before the testing algorithm exhaust the test case generation in a specific time. The result is presented in Table 2. The result shows that random testing can take comparable or worse time than our verification algorithm, even without generating a single counterexample (bias instance).

6 CONCLUSION AND DISCUSSION

We have considered a notion of individual fairness for structured data, and the problem of verifying the lack of individual bias in a given decision model. We have given a meta-algorithm for solving this problem, as well as specific algorithms for linear models and kernelized models with polynomial/RBF kernels. To the best of our knowledge, this is the first work that considers the verification of individual fairness for ML models.

Analysis of Model Bias Our algorithms output either a no-bias certificate or a bias instance (input pair), but this may be insufficient by itself in real-world investigations of model bias. To further the analysis, our solution offers two possibilities out-of-the-box. One possibility is to tighten the constraints on the input features, invoking the verifier every time, to find different *input regions* where the model is fair. Another possibility is to impose additional linear constraints (domain-knowledge-based) on the bias instances, which does not affect the optimization formulations. *Note that we are not reporting any experimental results for the above analyses*. Another important problem is to find regions where bias exists for *every* input in that region, but this is not possible using our verifier alone.

Future Work In future, we plan to extend this work in the following dimensions — 1) verifying wider classes of ML models, 2) extending our techniques to work with other individual & group fairness definitions, and 3) exploring different abstraction-refinement schemes such as counter-example driven refinement.

Acknowledgements The authors would like to thank Dinesh Garg and Rishi Saket for helpful discussions.

References

[1] Aniya Aggarwal et al. “Black Box Fairness Testing of Machine Learning Models”. In: *Proc. ESEC/FSE*. ACM, 2019, pp. 625–635.

[2] Aws Albarghouthi et al. “FairSquare: probabilistic verification of program fairness”. In: *PACMPL* 1.OOPSLA (2017), 80:1–80:30.

[3] Julia Angwin et al. *Machine Bias: There’s software used across the country to predict future criminals. And it’s biased against blacks*. 2016.

[4] Osbert Bastani et al. “Probabilistic Verification of Fairness Properties via Concentration”. In: *Proc. ACM Program. Lang.* OOPSLA (2019).

[5] Rudy R Bunel et al. “A Unified View of Piecewise Linear Neural Network Verification”. In: *Proc. NeurIPS*. 2018, pp. 4790–4799.

[6] Krishnamurthy Dvijotham et al. “A Dual Approach to Scalable Verification of Deep Networks”. In: *Proc. UAI*. 2018, pp. 162–171.

[7] Cynthia Dwork et al. “Fairness Through Awareness”. In: *Proc. ITCS*. ACM, 2012, pp. 214–226.

[8] Rüdiger Ehlers. “Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks”. In: *CoRR* abs/1705.01320 (2017).

[9] Mahyar Fazlyab et al. “Safety Verification and Robustness Analysis of Neural Networks via Quadratic Constraints and Semidefinite Programming”. In: *CoRR* abs/1903.01287 (2019).

[10] Sainyam Galhotra et al. “Fairness Testing: Testing Software for Discrimination”. In: *Proc. ESEC/FSE*. ACM, 2017, pp. 498–510.

[11] Timon Gehr et al. “AI2: Safety and robustness certification of neural networks with abstract interpretation”. In: *2018 IEEE Security and Privacy (SP)*. IEEE, 2018, pp. 3–18.

[12] Sven Gowal et al. “On the effectiveness of interval bound propagation for training verifiably robust models”. In: *CoRR* abs/1810.12715 (2018).

[13] John E. Hopcroft et al. *Introduction To Automata Theory, Languages, And Computation*. 1st. Addison-Wesley, 1990.

[14] Guy Katz et al. “Reluplex: An efficient SMT solver for verifying deep neural networks”. In: *CAV 2017*. Springer, 2017, pp. 97–117.

[15] J. Zico Kolter et al. “Provable defenses against adversarial examples via the convex outer adversarial polytope”. In: *Proc. ICML*. Vol. 80. PMLR, 2018, pp. 5283–5292.

[16] Matt J Kusner et al. “Counterfactual Fairness”. In: *Proc. NeurIPS*. 2017, pp. 4066–4076.

[17] Jean Bernard Lasserre. *An introduction to polynomial and semi-algebraic optimization*. CUP, 2015.

[18] Changliu Liu et al. “Algorithms for Verifying Deep Neural Networks”. In: *CoRR* abs/1903.06758 (2019).

[19] Matthew Mirman et al. “Differentiable abstract interpretation for provably robust neural networks”. In: *Proc. ICML*. Vol. 80. PMLR, 2018, pp. 3575–3583.

[20] Ramaravind K. Mothilal et al. *Modified Adult Dataset (DiCE)*. `https://github.com/microsoft/DiCE/`.

[21] Chongli Qin et al. “Verification of Non-Linear Specifications for Neural Networks”. In: *Proc. ICLR*. OpenReview.net, 2019.

[22] Aditi Raghunathan et al. “Semidefinite relaxations for certifying robustness to adversarial examples”. In: *Proc. NeurIPS*. 2018, pp. 10877–10887.

[23] Hadi Salman et al. “A Convex Relaxation Barrier to Tight Robustness Verification of Neural Networks”. In: *CoRR* abs/1902.08722 (2019).

[24] Naum Z Shor. “Class of global minimum bounds of polynomial functions”. In: *Cybernetics and Systems Analysis* 23.6 (1987), pp. 731–734.

[25] Gagandeep Singh et al. “An Abstract Domain for Certifying Neural Networks”. In: *Proc. ACM Program. Lang.* 3.POPL (Jan. 2019), 41:1–41:30.

[26] Gagandeep Singh et al. “Fast and Effective Robustness Certification”. In: *Proc. NeurIPS*. 2018, pp. 10802–10813.

[27] Vincent Tjeng et al. “Evaluating Robustness of Neural Networks with Mixed Integer Programming”. In: *Proc. ICLR*. OpenReview.net, 2019.

[28] Sakshi Udeshi et al. “Automated Directed Fairness Testing”. In: *Proc. ASE*. ACM, 2018, pp. 98–108.

[29] Shiqi Wang et al. “Efficient Formal Safety Analysis of Neural Networks”. In: *Proc. NeurIPS*. 2018, pp. 6367–6377.

[30] Shiqi Wang et al. “Formal Security Analysis of Neural Networks Using Symbolic Intervals”. In: *Proc. 27th USENIX Conference on Security Symposium*. 2018, pp. 1599–1614.

[31] Shiqi Wang et al. “MixTrain: Scalable Training of Formally Robust Neural Networks”. In: *CoRR* abs/1811.02625 (2018).

[32] Huan Zhang et al. “Efficient Neural Network Robustness Certification with General Activation Functions”. In: *Proc. NeurIPS*. 2018, pp. 4939–4948.

[33] Jie M. Zhang et al. “Machine Learning Testing: Survey, Landscapes and Horizons”. In: *CoRR* abs/1906.10742 (2019).