Use Privacy in Data-Driven Systems

Theory and Experiments with Machine Learnt Programs

Anupam Datta
Carnegie Mellon University

Matt Fredrikson
Carnegie Mellon University

Gihyuk Ko
Carnegie Mellon University

Piotr Mardziel
Carnegie Mellon University

Shayak Sen
Carnegie Mellon University

ABSTRACT

This paper presents an approach to formalizing and enforcing a class of *use privacy* properties in data-driven systems. In contrast to prior work, we focus on use restrictions on proxies (i.e. strong predictors) of protected information types. Our definition relates proxy use to intermediate computations that occur in a program, and identify two essential properties that characterize this behavior: *1)* its result is strongly associated with the protected information type in question, and *2)* it is likely to causally affect the final output of the program. For a specific instantiation of this definition, we present a program analysis technique that detects instances of proxy use in a model, and provides a witness that identifies which parts of the corresponding program exhibit the behavior. Recognizing that not all instances of proxy use of a protected information type are inappropriate, we make use of a normative judgment oracle that makes this inappropriateness determination for a given witness. Our repair algorithm uses the witness of an inappropriate proxy use to transform the model into one that provably does not exhibit proxy use, while avoiding changes that unduly affect classification accuracy. Using a corpus of social datasets, our evaluation shows that these algorithms are able to detect proxy use instances that would be difficult to find using existing techniques, and subsequently remove them while maintaining acceptable classification performance.

CCS CONCEPTS

• **Security and privacy → Privacy protections**;

KEYWORDS

privacy; use privacy; proxy; causal analysis

1 INTRODUCTION

Restrictions on information use occupy a central place in privacy regulations and legal frameworks [28, 54, 61, 62]. We introduce the term *use privacy* to refer to privacy norms governing information use. A number of recent cases have evidenced that inappropriate information use can lead to violations of both privacy laws [68] and user expectations [16, 19], prompting calls for technology to assist with enforcement of use privacy requirements [53]. In order to meet these regulatory imperatives and user expectations, companies dedicate resources toward compliance with privacy policies governing information use [53, 57]. A large body of work has emerged around use privacy compliance governing the explicit use of protected information types (see Tschantz et al. [64] for a survey). Such methods are beginning to see deployment in major technology companies like Microsoft [58].

In this paper, we initiate work on formalizing and enforcing a richer class of use privacy restrictions—those governing the use of protected information indirectly through proxies in data-driven systems. Data-driven systems include machine learning and artificial intelligence systems that use large swaths of data about individuals in order to make decisions about them. The increasing adoption of these systems in a wide range of sectors, including advertising, education, healthcare, employment, and credit, underscores the critical need to address use privacy concerns [53, 57].

We start with a set of examples to motivate these privacy concerns and identify the key research challenges that this paper will tackle to address them. In 2012, the department store Target drew flak from privacy advocates and data subjects for using the shopping history of their customers to predict their pregnancy status and market baby items based on that information [19]. While Target intentionally inferred the pregnancy status and used it for marketing, the privacy concern persists even if the inference were not explicitly drawn. Indeed, the use of health condition-related search terms and browsing history—*proxies* (i.e., strong predictors) for health conditions—for targeted advertising have been the basis for legal action and public concern from a privacy standpoint [16, 44, 68]. Similar privacy concerns have been voiced about the use of personal information in the Internet of Things [40, 49, 52, 67].

Use privacy To address these threats, this paper articulates the problem of protecting *use privacy* in data-driven systems.

Use privacy constraints restrict the use of protected information types and some of their proxies in data-driven systems.

Setting A use privacy constraint may require that health information or its proxies not be used for advertising. Indeed there are calls for this form of privacy constraint [17, 46, 53, 68]. In this paper, we consider the setting where a data-driven system is audited to ensure that it complies with such use privacy constraints. The auditing could be done by a trusted data processor who is operating the system or by a regulatory oversight organization who has access to the data processors' machine learning models and knowledge of

CCS '17, October 30-November 3, 2017, Dallas, TX, USA

ACM ISBN 978-1-4503-4946-8/17/10…$15.00
https://doi.org/http://dx.doi.org/10.1145/3133956.3134097

the distribution of the dataset. In other words, we assume that the data processor does not act to evade the detection algorithm, and provides accurate information. This trusted data processor setting is similar to the one assumed in differential privacy [25].

In this setting, it is impossible to guarantee that data processors with strong background knowledge are not able to infer certain facts about individuals (e.g., their pregnancy status) [21]. Even in practice, data processors often have access to detailed profiles of individuals and can infer sensitive information about them [19, 66]. Use privacy instead places a more pragmatic requirement on data-driven systems: that they simulate ignorance of protected information types (e.g., pregnancy status) by not using them or their proxies in their decision-making. This requirement is met if the systems (e.g., machine learning models) do not infer protected information types or their proxies (even if they could) or if such inferences do not affect decisions.

Recognizing that not all instances of proxy use of a protected information type are inappropriate, our theory of use privacy makes use of a normative judgment oracle that makes this inappropriateness determination for a given instance. For example, while using health information or its proxies for credit decisions may be deemed inappropriate, an exception could be made for proxies that are directly relevant to the credit-worthiness of the individual (e.g., her income and expenses).

Proxy use A key technical contribution of this paper is a formalization of *proxy use* of protected information types in programs. Our formalization relates proxy use to intermediate computations obtained by decomposing a program. We begin with a qualitative definition that identifies two essential properties of the intermediate computation (the proxy): *1)* its result perfectly predicts the protected information type in question, and *2)* it has a causal affect on the final output of the program.

In practice, this qualitative definition of proxy use is too rigid for machine learning applications along two dimensions. First, instead of demanding that proxies are perfect predictors, we use a standard measure of association strength from the quantitative information flow security literature to define an ϵ-*proxy* of a protected information type; here $\epsilon \in [0, 1]$ with higher values indicating a stronger proxy. Second, qualitative causal effects are not sufficiently informative for our purpose. Instead we use a recently introduced causal influence measure [14] to quantitatively characterize influence. We call it the δ-*influence* of a proxy where $\delta \in [0, 1]$ with higher values indicating stronger influence. Combining these two notions, we define a notion of (ϵ, δ)-proxy use.

We arrive at this program-based definition after a careful examination of the space of possible definitions. In particular, we prove that it is impossible for a purely semantic notion of intermediate computations to support a meaningful notion of proxy use as characterized by a set of natural properties or axioms (Theorem 1). The program-based definition arises naturally from this exploration by replacing semantic decomposition with decompositions of the program. An important benefit of this choice of restricting the search for intermediate computations to those that appear in the text of the program is that it supports natural algorithms for detection and repair of proxy use. Our framework is parametric in the choice of a programming language in which the programs (e.g., machine learnt models) are expressed and the population to which it is applied. The choice of the language reflects the level of white-box access that the analyst has into the program.

Detection We instantiate our definition to a simple programming language that contains conditionals, arithmetic and logical operations, and decompositions that involve single variables and associative arithmetic. For example, decompositions of linear models include additive sets of linear terms, and decision forests include subtrees, and sets of decision trees. For this instantiation of the definition, we present a program analysis technique that detects proxy use in a model, and provides a witness that identifies which parts of the corresponding program exhibit the behavior (Algorithm 4). Our algorithm assumes access to the text of a program that computes the model, as well as a dataset that has been partitioned into analysis and validation subsets. The algorithm is program-directed and is directly inspired by the definition of proxy use. We prove that the algorithm is complete relative to our instantiation of the proxy use definition — it identifies every instance of proxy use in the program (Theorem 3) and outputs witnesses (i.e. intermediate computations that are the proxies). We provide three optimizations that leverage *sampling*, *pre-computation*, and *reachability* to speed up the detection algorithm.

Repair If a found instance of proxy use is deemed inappropriate, our repair algorithm (Algorithm 5) uses the witness to transform the model into one that provably does not exhibit that instance of proxy use (Theorem 4), while avoiding changes that unduly affect classification accuracy. We leverage the witnesses that localize where in the program a violation occurs in order to focus repair there. To repair a violation, we search through expressions local to the violation, replacing the one which has the least impact on the accuracy of the model and at the same time reduces the association or influence of the violation to below the (ϵ, δ) threshold.

Evaluation We empirically evaluate our proxy use definition, detection and repair algorithms on four real datasets used to train decision trees, linear models, and random forests. Our evaluation demonstrates the typical workflow for practitioners who use our tools for a simulated financial services application. It highlights how they help them uncover more proxy uses than a baseline procedure that simply eliminates features associated with the protected information type. For three other simulated settings on real data sets—contraception advertising, student assistance, and credit advertising—we find interesting proxy uses and discuss how the outputs of our detection tool could aid a normative judgment oracle determine the appropriateness of proxy uses. We evaluate the performance of the detection algorithm and show that, in particular cases, the runtime of our system scales linearly in the size of the model. We demonstrate the completeness of the detection algorithm by having it discover artificially injected violations into real data sets. Finally, we evaluate impact of repair on model accuracy, in particular, showing a graceful degradation in accuracy as the influence of the violating proxy increases.

Closely related work The emphasis on restricting use of information by a system rather than the knowledge possessed by agents distinguishes our work from a large body of work in privacy (see Smith [59] for a survey). The privacy literature on use

restrictions has typically focused on explicit use of protected information types, and not on proxy use (see Tschantz et al. [64] for a survey and Lipton and Regan [46]). Recent work on discovering personal data use by black-box web services focuses mostly on explicit use of protected information types by examining causal effects [2, 16, 27, 35–37, 43, 44, 47, 69, 71]); some of this work also examines associational effects [43, 44]. Associational effects capture some forms of proxy use but not others as we argue in Section 3.

In a setting similar to ours of a trusted data processor, differential privacy [25] protects against a different type of privacy harm. For a computation involving data contributed by a set of individuals, differential privacy minimizes any knowledge gains by an adversary that are caused by the contribution of a single individual. This requirement, however, says nothing about what information types about an individual are actually used by the data processor, the central concern of use privacy.

Lipton and Regan's notion of "effectively private" captures the idea that a protected feature is not explicitly used to make decisions, but does not account for proxy use [46]. Prior work on fairness has also recognized the importance of dealing with proxies in machine learning systems [22, 29, 63]. However treatments of proxy use considered there do not match the requirements of use privacy. We elaborate on this point in Section 3. In Section 7, we provide a more detailed comparison with related work highlighting that use privacy enhancing technology (PET) complements existing work on PETs. It is not meant to supplant other PETs geared toward restricting data collection and release. While the results of this paper represent significant progress toward enabling use privacy, as elaborated in Section 8, a host of challenging problems remain open.

Contributions In summary, we make the following contributions:

- An articulation of the problem of protecting *use privacy* in data-driven systems. Use privacy restricts the use of protected information types and some of their proxies (i.e., strong predictors) in automated decision-making systems (§1, 2).
- A formal definition of *proxy use*—a key building block for use privacy–and an axiomatic basis for this definition (§3).
- An algorithm for detection and tracing of an instantiation of proxy use in a machine learnt program, and proof that this algorithm is sound and complete (§4).
- A repair algorithm that provably removes violations of the proxy use instantiation in a machine learning model that are identified by our detection algorithm and deemed inappropriate by a normative judgment oracle (§5).
- An implementation and evaluation of our approach on popular machine learning algorithms applied to real datasets (§6).

An extended version[13] of this paper includes additional evaluation and further details on the datasets employed.

2 USE PRIVACY

We use the Target example described earlier in the paper to motivate our notion of use privacy. Historically, data collected in a context of interaction between a retailer and a consumer is not expected to result in flows of health information. However, such flow constraints considered in significant theories of privacy (e.g., see Nissenbaum [51]) cannot be enforced because of possible statistical inferences. In particular, prohibited information types (e.g., pregnancy status) could be inferred from legitimate flows (e.g., shopping history). Thus, the theory of use privacy instead ensures that the data processing systems "simulate ignorance" of protected information types (e.g., pregnancy status) and their proxies (e.g., purchase history) by not using them in their decision-making. Because not all instances of proxy use of a protected information type are inappropriate, our theory of use privacy makes use of a normative judgment oracle that makes this inappropriateness determination for a given instance.

We model the personal data processing system as a program p. The use privacy constraint governs a protected information type Z. Our definition of use privacy makes use of two building blocks: (1) a function that given p, Z, and a population distribution $\mathcal{P}$ returns a witness w of proxy use of Z in a program p (if it exists); and (2) a normative judgment oracle $\mathcal{O}(w)$ that given a specific witness returns a judgment on whether the specific proxy use is appropriate (TRUE) or not (FALSE).

DEFINITION 1 (USE PRIVACY). *Given a program p, protected information type Z, normative judgment oracle $\mathcal{O}$, and population distribution $\mathcal{P}$, use privacy in a program p is violated if there exists a witness w in p of proxy use of Z in $\mathcal{P}$ such that $\mathcal{O}(w)$ returns* FALSE.

In this paper, we formalize the computational component of the above definition of use privacy, by formalizing what it means for an algorithm to use a protected information type directly or through proxies (§3) and designing an algorithm to detect proxy uses in programs (§4). We assume that the normative judgment oracle is given to us and use it to identify inappropriate proxy uses and then repair them (§5). In our experiments, we illustrate how such an oracle would use the outputs of our proxy use analysis and recommend the repair of uses deemed inappropriate by it (§6).

This definition cleanly separates computational considerations that are automatically enforceable and ethical judgments that require input from human experts. This form of separation exists also in some prior work on privacy [33] and fairness [23].

3 PROXY USE: A FORMAL DEFINITION

We now present an axiomatically justified, formal definition of proxy use in data-driven programs. Our definition for proxy use of a protected information type involves *decomposing* a program to find an intermediate computation whose result exhibits two properties:

- *Proxy*: strong association with the protected type
- *Use*: causal influence on the output of the program

In § 3.1, we present a sequence of examples to illustrate the challenge in identifying proxy use in systems that operate on data associated with a protected information type. In doing so, we will also contrast our work with closely-related work in privacy and fairness. In §3.2, we formalize the notions of proxy and use, preliminaries to the definition. The definition itself is presented in §3.3 and §3.4. Finally, in §3.5, we provide an axiomatic characterization of the notion of proxy use that guides our definitional choices. We note

that readers keen to get to the detection and repair mechanisms may skip §3.5 without loss of continuity.

3.1 Examples of Proxy Use

Prior work on detecting use of protected information types [15, 30, 44, 63] and leveraging knowledge of detection to eliminate inappropriate uses [30] have treated the system as a black-box. Detection relied either on experimental access to the black-box [15, 44] or observational data about its behavior [30, 63]. Using a series of examples motivated by the Target case, we motivate the need to peek inside the black-box to detect proxy use.

Example 3.1. (Explicit use, Fig. 1a) A retailer explicitly uses pregnancy status from prescription data available at its pharmacy to market baby products.

This form of explicit use of a protected information type can be discovered by existing black-box experimentation methods that establish causal effects between inputs and outputs (e.g., see [15, 44]).

Example 3.2. (Inferred use, Fig. 1b) Consider a situation where purchase history can be used to accurately predict pregnancy status. A retailer markets specific products to individuals who have recently purchased products indicative of pregnancy (e.g., $a_1, a_2 \in$ purchases).

This example, while very similar in effect, does not use health information directly. Instead, it infers pregnancy status via associations and then uses it. Existing methods (see [30, 63]) can detect such associations between protected information types and outcomes in observational data.

Example 3.3. (No use, Fig. 1c) Retailer uses some uncorrelated selection of products ($a_1, n_1 \in$ purchases) to suggest ads.

In this example, even though the retailer could have inferred pregnancy status from the purchase history, no such inference was used in marketing products. As associations are commonplace, a definition of use disallowing such benign use of associated data would be too restrictive for practical enforcement.

Example 3.4. (Masked proxy use, Fig. 1d) Consider a more insidious version of Example 3.2. To mask the association between the outcome and pregnancy status, the company also markets baby products to people who are not pregnant, but have low retail engagement, so these advertisements would not be viewed in any case.

While there is no association between pregnancy and outcome in both Example 3.3 and Example 3.4, there is a key difference between them. In Example 3.4, there is an intermediate computation based on aspects of purchase history that is a predictor for pregnancy status, and this predictor is used to make the decision, and therefore is a case of proxy use. In contrast, in Example 3.3, the intermediate computation based on purchase history is uncorrelated with pregnancy status. Distinguishing between these examples by measuring associations using black box techniques is non-trivial. Instead, we leverage white-box access to the code of the classifier to identify the intermediate computation that serves as a proxy for pregnancy status. Precisely identifying the particular proxy used also aids the normative decision of whether the proxy use is appropriate in this setting.

f	A function
$\langle \mathbf{X}, \mathcal{A} \rangle_{\mathcal{P}}$	A model, which is a function $\mathcal{A}$ used for prediction, operating on random variables $\mathbf{X}$, in population $\mathcal{P}$
X	A random variable
p	A program
$\langle \mathbf{X}, p \rangle_{\mathcal{P}}$	A syntactic model, which is a program p, operating on random variables $\mathbf{X}$
$[p_1/X]p_2$	A substitution of p_1 in place of X in p_2
$\mathbf{X}$	A sequence of random variables

Table 1: Summary of notation used in the paper

3.2 Notation and Preliminaries

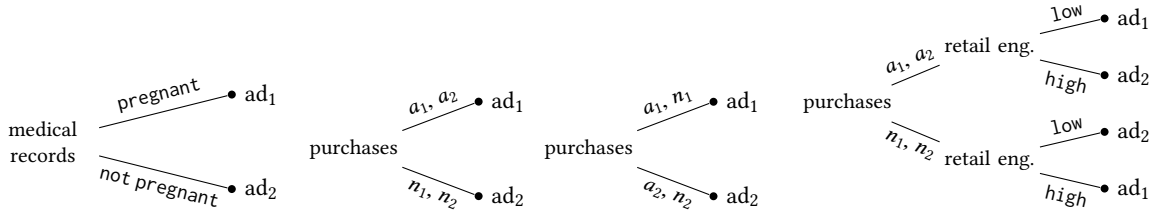
We assume individuals are drawn from a population distribution $\mathcal{P}$, in which our definitions are parametric. Random variables $W, X, Y, Z, \ldots$ are functions over $\mathcal{P}$, and the notation $W \in \mathcal{W}$ represents that the type of random variable is $W : \mathcal{P} \to \mathcal{W}$. An important random variable used throughout the paper is $\mathbf{X}$, which represents the vector of features of an individual that is provided to a predictive model. A predictive model is denoted by $\langle \mathbf{X}, \mathcal{A} \rangle_{\mathcal{P}}$, where $\mathcal{A}$ is a function that operates on $\mathbf{X}$. For simplicity, we assume that $\mathcal{P}$ is discrete, and that models are deterministic. Table 1 summarizes all the notation used in this paper, in addition to the notation for programs that is introduced later in the paper.

3.2.1 Proxies. A *perfect proxy* for a random variable Z is a random variable X that is perfectly correlated with Z. Informally, if X is a proxy of Z, then X or Z can be interchangeably used in any computation over the same distribution. One way to state this is to require that $\Pr(X = Z) = 1$, i.e. X and Z are equal on the distribution. However, we require our definition of proxy to be invariant under renaming. For example, if X is 0 whenever Z is 1 and vice versa, we should still identify X to be a proxy for Z. In order to achieve invariance under renaming, our definition only requires the existence of mappings between X and Z, instead of equality.

Definition 2 (Perfect Proxy). *A random variable $X \in \mathcal{X}$ is a* perfect proxy for $Z \in \mathcal{Z}$ *if there exist functions $f : \mathcal{X} \to \mathcal{Z}, g : \mathcal{Z} \to \mathcal{X}$, such that* $\Pr(Z = f(X)) = \Pr(g(Z) = X) = 1$.

While this notion of a proxy is too strong in practice, it is useful as a starting point to explain the key ideas in our definition of proxy use. This definition captures two key properties of proxies, *equivalence* and *invariance under renaming*.

Equivalence Definition 2 captures the property that proxies admit predictors in both directions: it is possible to construct a predictor of X from Z, and vice versa. This condition is required to ensure that our definition of proxy only identifies the part of the input that corresponds to the protected attribute and not the input attribute as a whole. For example, if only the final digit of a zip code is a proxy for race, the entirety of the zip code will not be identified as a proxy even though it admits a predictor in one direction. Only if the final digit is used, that use will be identified as proxy use.



(a) Explicit Use **(b) Use via proxy** **(c) No use** **(d) Masked use via proxy**

Figure 1: Examples of models (decision trees) used by a retailer for offering medicines and for selecting advertisements to show to customers. The retailer uses pregnancy status, past purchases, and customer's level of retail engagement. Products a_1 and a_2 are associated with pregnancy (e.g., prenatal vitamins, scent-free lotions) whereas products n_1 and n_2 are associated with a lack of pregnancy (e.g., alcohol, camping gear); all four products are equally likely. Retail engagement, (`high` or `low`), indicating whether the customer views ads or not, is independent of pregnancy.

The equivalence criterion distinguishes benign use of associated information from proxy use as illustrated in the next example. For machine learning in particular, this is an important pragmatic requirement; given enough input features one can expect any protected class to be predictable from the set of inputs. In such cases, the input features taken together are a strong associate in one direction, and prohibiting such one-sided associates from being used would rule out most machine learnt models.

Example 3.5. Recall that in Figure 1, a_1, a_2 is a proxy for pregnancy status. In contrast, consider Example 3.3, where purchase history is an influential input to the program that serves ads. Suppose that the criteria is to serve ads to those with a_1, n_1 in their purchase history. According to Definition 2, neither purchase history or a_1, n_1 are proxies, because pregnancy status does not predict purchase history or a_1, n_1. However, if Definition 2 were to allow one-sided associations, then purchase history would be a proxy because it can predict pregnancy status. This would have the unfortunate effect of implying that the benign application in Example 3.3 has proxy use of pregnancy status.

Invariance under renaming This definition of a proxy is invariant under renaming of the values of a proxy. Suppose that a random variable evaluates to 1 when the protected information type is 0 and vice versa, then this definition still identifies the random variable as a proxy.

3.2.2 Influence. Our definition of influence aims to capture the presence of a causal dependence between a variable and the output of a function. Intuitively, a variable x is influential on f if it is possible to change the value of f by changing x while keeping the other input variables fixed.

Definition 3. *For a function $f(x, y)$, x is influential if and only if there exists values x_1, x_2, y, such that $f(x_1, y) \neq f(x_2, y)$.*

In Figure 1a, pregnancy status is an influential input of the system, as just changing pregnancy status while keeping all other inputs fixed changes the prediction. Influence, as defined here, is identical to the notion of interference used in the information flow literature.

3.3 Definition

We use an abstract framework of program syntax to reason about programs without specifying a particular language to ensure that our definition remains general. Our definition relies on syntax to reason about decompositions of programs into intermediate computations, which can then be identified as instances of proxy use using the concepts described above.

Program decomposition We assume that models are represented by programs. For a set of random variables $\mathbf{X}$, $\langle \mathbf{X}, p \rangle_{\mathcal{P}}$ denotes the assumption that p will run on the variables in $\mathbf{X}$. Programs are given meaning by a denotation function $[\![\cdot]\!]_{\mathbf{X}}$ that maps programs to functions. If $\langle \mathbf{X}, p \rangle_{\mathcal{P}}$, then $[\![p]\!]$ is a function on variables in $\mathbf{X}$, and $[\![p]\!](\mathbf{X})$ represents the random variable of the outcome of p, when evaluated on the input random variables $\mathbf{X}$. Programs support substitution of free variables with other programs, denoted by $[p_1/X]p_2$, such that if p_1 and p_2 programs that run on the variables $\mathbf{X}$ and $\mathbf{X}, X$, respectively, then $[p_1/X]p_2$ is a program that operates on $\mathbf{X}$.

A decomposition of program p is a way of rewriting p as two programs p_1 and p_2 that can be combined via substitution to yield the original program.

Definition 4 (Decomposition). *Given a program p, a decomposition (p_1, X, p_2) consists of two programs p_1, p_2, and a fresh variable X, such that $p = [p_1/X]p_2$.*

For the purposes of our proxy use definition we view the first component p_1 as the intermediate computation suspected of proxy use, and p_2 as the rest of the computation that takes in p_1 as an input.

Definition 5 (Influential Decomposition). *Given a program p, a decomposition (p_1, X, p_2) is influential iff X is influential in p_2.*

Main definition

Definition 6 (Proxy Use). *A program $\langle \mathbf{X}, p \rangle_{\mathcal{P}}$ has proxy use of Z if there exists an influential decomposition (p_1, X, p_2) of $\langle \mathbf{X}, p \rangle_{\mathcal{P}}$, and $[\![p_1]\!](\mathbf{X})$ is a proxy for Z.*

Example 3.6. In Figure 1d, this definition would identify proxy use using the decomposition (p_1, U, p_2), where p_2 is the entire tree, but with the condition $(a_1, a_2 \in \text{purchases})$ replaced by the variable U. In this example, U is influential in p_2, since changing the value of U changes the outcome. Also, we assumed that the condition $(a_1, a_2 \in \text{purchases})$ is a perfect predictor for pregnancy, and is therefore a proxy for pregnancy. Therefore, according to our definition of proxy use, the model in 1d has proxy use of pregnancy status.

3.4 A Quantitative Relaxation

Definition 6 is too strong in one sense and too weak in another. It requires that intermediate computations be perfectly correlated with a protected attribute, and that there exists *some* input, however improbable, in which the result of the intermediate computation is relevant to the model. For practical purposes, we would like to capture imperfect proxies that are strongly associated with an attribute, but only those whose influence on the final model is appreciable. To relax the requirement of perfect proxies and non-zero influence, we quantify these two notions to provide a parameterized definition. Recognizing that neither perfect privacy nor perfect utility are practical, the quantitative definition provides a means for navigating privacy vs. utility tradeoffs.

ϵ-proxies We wish to measure how strongly a random variable X is a proxy for a random variable Z. Recall the two key requirements from the earlier definition of a proxy: (i) the association needs to be capture equivalence and measure association in both directions, and (ii) the association needs to be invariant under renaming of the random variables. The *variation of information metric* $d_{\text{var}}(X,Z) = H(X|Z) + H(Z|X)$ [12] is one measure that satisfies these two requirements. The first component in the metric, the conditional entropy of X given Z, $H(X|Z)$, measures how well X can be predicted from Z, and $H(Z|X)$ measures how well Z can be predicted from X, thus satisfying the requirement for the metric measuring association in both directions. Additionally, one can show that conditional entropies are invariant under renaming, thus satisfying our second criteria. To obtain a normalized measure in $[0,1]$, we choose $1 - \frac{d_{\text{var}}(X,Z)}{H(X,Z)}$ as our measure of association, where the measure being 1 implies perfect proxies, and 0 implies statistical independence. Interestingly, this measure is identical to normalized mutual information [12], a standard measure that has also been used in prior work in identifying associations in outcomes of machine learning models [63].

Definition 7 (Proxy Association). *Given two random variables X and Z, the strength of a proxy is given by normalized mutual information,*

$$d(X,Z) \stackrel{\text{def}}{=} 1 - \frac{H(X|Z) + H(Z|X)}{H(X,Z)}$$

where X is defined to be an ϵ-proxy for Z if $d(X,Z) \geq \epsilon$.

We do not present the complexity of association computation independently of detection as we rely on pre-computations to reduce the amortized runtime of the entire detection algorithm. The complexity as part of our detection algorithm is discussed in Appendix D.2.

δ-influential decomposition Recall that for a decomposition (p_1, X, p_2), in the qualitative sense, influence is interference which implies that there exists x, x_1, x_2, such that $[\![p_2]\!](x, x_1) \neq [\![p_2]\!](x, x_2)$. Here x_1, x_2 are values of p_1, that for a given x, change the outcome of p_2. However, this definition is too strong as it requires only a single pair of values x_1, x_2 to show that the outcome can be changed by p_1 alone. To measure influence, we quantify interference by using Quantitative Input Influence (QII), a causal measure of input influence introduced in [14]. In our context, for a decomposition (p_1, X, p_2), the influence of p_1 on p_2 is given by:

$$\iota(p_1, p_2) \stackrel{\text{def}}{=} \mathbb{E}_{\mathbf{X}, \mathbf{X}' \stackrel{\$}{\leftarrow} \mathcal{P}} \Pr\left([\![p_2]\!](\mathbf{X}, [\![p_1]\!](\mathbf{X})) \neq [\![p_2]\!](\mathbf{X}, [\![p_1]\!](\mathbf{X}'))\right).$$

Intuitively, this quantity measures the likelihood of finding randomly chosen values of the output of p_1 that would change the outcome of p_2. Note that this general definition allows for probabilistic models though in this work we only evaluate our methods on deterministic models.

The time complexity of influence computation as part of our detection algorithm can be found in Appendix D.2, along with discussion on estimating influence.

Definition 8 (Decomposition Influence). *Given a decomposition (p_1, X, p_2), the influence of the decomposition is given by the QII of X on p_2. A decomposition (p_1, X, p_2) is defined to be δ-influential if $\iota(p_1, p_2) > \delta$.*

(ϵ, δ)-proxy use Now that we have quantitative versions of the primitives used in Definition 6, we are in a position to define quantitative proxy use (Definition 9). The structure of this definition is the same as before, with quantitative measures substituted in for the qualitative assertions used in Definition 6.

Definition 9 ((ϵ, δ)-proxy use). *A program $\langle \mathbf{X}, p \rangle_{\mathcal{P}}$ has (ϵ, δ)-proxy use of random variable Z iff there exists a δ-influential decomposition (p_1, X, p_2), such that $[\![p]\!](\mathbf{X})$ is an ϵ-proxy for Z.*

This definition is a strict relaxation of Definition 6, which reduces to $(1, 0)$-proxy use.

3.5 Axiomatic Basis for Definition

We now motivate our definitional choices by reasoning about a natural set of properties that a notion of proxy use should satisfy. We first prove an important impossibility result that shows that no definition of proxy use can satisfy four natural semantic properties of proxy use. The central reason behind the impossibility result is that under a purely semantic notion of function composition, the causal effect of a proxy can be made to disappear. Therefore, we choose a syntactic notion of function composition for the definition of proxy use presented above. The syntactic definition of proxy use is characterized by syntactic properties which map very closely to the semantic properties.

Property 1. (Explicit Use) *If Z is an influential input of the model $\langle \{\mathbf{X}, Z\}, \mathcal{A} \rangle_{\mathcal{P}}$, then $\langle \{\mathbf{X}, Z\}, \mathcal{A} \rangle_{\mathcal{P}}$ has proxy use of Z.*

This property identifies the simplest case of proxy use: if an input to the model is influential, then the model exhibits proxy use of that input.

Property 2. (Preprocessing) *If a model $\langle \{\mathbf{X}, X\}, \mathcal{A} \rangle_{\mathcal{P}}$ has proxy use of random variable Z, then for any function f such that $\Pr(f(\mathbf{X}) = X) = 1$, let $\mathcal{A}'(x) \stackrel{\text{def}}{=} \mathcal{A}(x, f(x))$. Then, $\langle \mathbf{X}, \mathcal{A}' \rangle_{\mathcal{P}}$ has proxy use of Z.*

This property covers the essence of proxy use where instead of being provided a protected information type explicitly, the program uses a strong predictor for it instead. This property states that models that use inputs explicitly and via proxies should not be differentiated under a reasonable theory of proxy use.

Property 3. (Dummy) *Given $\langle \mathbf{X}, \mathcal{A} \rangle_{\mathcal{P}}$, define $\mathcal{A}'$ such that for all x, x', $\mathcal{A}'(x, x') \stackrel{\text{def}}{=} \mathcal{A}(x)$, then $\langle \mathbf{X}, \mathcal{A} \rangle_{\mathcal{P}}$ has proxy use for some Z iff $\langle \{\mathbf{X}, X\}, \mathcal{A}' \rangle_{\mathcal{P}}$ has proxy use of Z.*

This property states that the addition of an input to a model that is not influential, i.e., has no effect on the outcomes of the model, has no bearing on whether a program has proxy use or not. This property is an important sanity check that ensures that models aren't implicated by the inclusion of inputs that they do not use.

Property 4. (Independence) *If* $\mathbf{X}$ *is independent of* Z *in* $\mathcal{P}$*, then* $\langle \mathbf{X}, \mathcal{A} \rangle_{\mathcal{P}}$ *does not have proxy use of* Z*.*

Independence between the protected information type and the inputs ensures that the model cannot infer the protected information type for the population $\mathcal{P}$. This property captures the intuition that if the model cannot infer the protected information type then it cannot possibly use it.

While all of these properties seem intuitively desirable, it turns out that these properties can not be achieved simultaneously.

Theorem 1. *No definition of proxy use can satisfy Properties 1-4 simultaneously.*

See Appendix A for a proof of the impossibility result and a discussion. The key intuition behind this result is that Property 2 requires proxy use to be preserved when an input is replaced with a function that predicts that input via composition. However, with a purely semantic notion of function composition, after replacement, the proxy may get canceled out. To overcome this impossibility result, we choose a more syntactic notion of function composition, which is tied to how the function is represented as a program, and looks for evidence of proxy use within the representation.

We now proceed to the axiomatic justification of our definition of proxy use. As in our attempt to formalize a semantic definition, we base our definition on a set of natural properties given below. These are syntactic versions of their semantic counterparts defined earlier.

Property 5. (Syntactic Explicit Use) *If* X *is a proxy of* Z*, and* X *is an influential input of* $\langle \{\mathbf{X}, X\}, p \rangle_{\mathcal{P}}$*, then* $\langle \{\mathbf{X}, X\}, p \rangle_{\mathcal{P}}$ *has proxy use.*

Property 6. (Syntactic Preprocessing) *If* $\langle \{\mathbf{X}, X\}, p_1 \rangle_{\mathcal{P}}$ *has proxy use of* Z*, then for any* p_2 *such that* $\Pr\left([\![p_2]\!](\mathbf{X}) = X\right) = 1$, $\langle \mathbf{X}, [p_2/X]p_1 \rangle_{\mathcal{P}}$ *has proxy use of* Z*.*

Property 7. (Syntactic Dummy) *Given a program* $\langle \mathbf{X}, p \rangle_{\mathcal{P}}$, $\langle \mathbf{X}, p \rangle_{\mathcal{P}}$ *has proxy use for some* Z *iff* $\langle \{\mathbf{X}, X\}, p \rangle_{\mathcal{P}}$ *has proxy use of* Z*.*

Property 8. (Syntactic Independence) *If* $\mathbf{X}$ *is independent of* Z*, then* $\langle \mathbf{X}, p \rangle_{\mathcal{P}}$ *does not have proxy use of* Z*.*

Properties 5 and 6 together characterize a complete inductive definition, where the induction is over the structure of the program. Suppose we can decompose programs p into (p_1, X, p_2) such that $p = [p_1/X]p_2$. Now if X, which is the output of p_1, is a proxy for Z and is influential in p_2, then by Property 5, p_2 has proxy use. Further, since $p = [p_1/X]p_2$, by Property 6, p has proxy use. This inductive definition where we use Property 5 as the base case and Property 6 for the induction step, precisely characterizes Definition 6. Additionally, it can be shown that Definition 6 also satisfies Properties 7 and 8. Essentially, by relaxing our notion of function composition to a syntactic one, we obtain a practical definition of proxy use characterized by the natural axioms above.

Algorithm 1 Detection for expression programs.

Require: association (d), influence(ι) measures
  procedure ProxyDetect($p, \mathbf{X}, Z, \epsilon, \delta$)
    $P \leftarrow \varnothing$
    for each subprogram p_1 appearing in p **do**
      for each program p_2 such that $[p_2/u]p_1 = p$ **do**
        if $\iota(p_1, p_2) \geq \delta \wedge d([\![p_1]\!](\mathbf{X}), Z) \geq \epsilon$ **then**
          $P \leftarrow P \cup \{(p_1, p_2)\}$
    return P

4 DETECTING PROXY USE

In this section, we present an algorithm for identifying proxy use of specified variables in a given machine-learning model (Algorithm 1, Appendix B contains a more formal presentation of the algorithm for the interested reader). The algorithm is program-directed and is directly inspired by the definition of proxy use in the previous section. We prove that the algorithm is complete in a strong sense — it identifies every instance of proxy use in the program (Theorem 3). We also describe three optimizations that speed up the detection algorithm: sampling, reachability analysis, and contingency tables.

4.1 Environment Model

The environment in which our detection algorithm operates is comprised of a data processor, a dataset that has been partitioned into analysis and validation subsets, and a machine learning model trained over the analysis subset. We assume that the data processor does not act to evade the detection algorithm, and the datasets correspond to a representative sample from the population we wish to test proxy use with respect to. Additionally, we assume that information types we wish to detect proxies of are also part of the validation data. We discuss these points further in Section 8.

For the rest of this paper we focus on an instance of the proxy use definition, where we assume that programs are written in the simple expression language shown in Figure 2. However, our techniques are not tied to this particular language, and the key ideas behind them apply generally. This language is rich enough to support commonly-used models such as decision trees, linear and logistic regression, Naive Bayes, and Bayesian rule lists. Programs are functions that evaluate arithmetic terms, which are constructed from real numbers, variables, common arithmetic operations, and if-then-else (**ite**$(\cdot, \cdot, \cdot)$) terms. Boolean terms, which are used as conditions in **ite** terms, are constructed from the usual connectives and relational operations. Finally, we use λ-notation for functions, i.e., $\lambda x.e$ denotes a function over x which evaluates e after replacing all instances of x with its argument. Details on how machine learning models such as linear models, decision trees, and random forests are translated to this expression language are discussed in Appendix B.2 and consequences of the choice of language and decomposition in that language are further discussed in more detail in Section 8.

Distributed proxies Our use of program decomposition provides for partial handling of *distributed representations*, the idea that concepts can be distributed among multiple entities. In our case, influence and association of a protected information type can be distributed among multiple program points. First, substitution

$$\langle aexp \rangle ::= \mathbb{R} \mid \text{var} \mid \text{op}(\langle aexp \rangle, \ldots, \langle aexp \rangle) \mid \textbf{ite}(\langle bexp \rangle, \langle aexp \rangle, \langle aexp \rangle)$$
$$\langle bexp \rangle ::= \text{T} \mid \text{F} \mid \neg \langle bexp \rangle \mid \text{op}(\langle bexp \rangle, \ldots, \langle bexp \rangle) \mid \text{relop}(\langle aexp \rangle, \langle aexp \rangle)$$
$$\langle prog \rangle ::= \lambda \text{var}_1, \ldots, \text{var}_n \,.\, \langle aexp \rangle$$

Figure 2: Syntax for the language used in our analysis.

(denoted by $[p_1/X]p_2$) is defined to replace *all* instances of variable X in p_2 with the program p_1. If there are multiple instances of X in p_2, they are still describing a single decomposition and thus the multiple instances of p_2 in p_1 are viewed as a single proxy. Further, implementations of substitution can be (and is in our implementation) associativity-aware: programs like $x_1 + x_2 + x_3$ can be equivalent regardless of the order of the expressions in that they can be decomposed in exactly the same set of ways. If a proxy is distributed among x_1 and x_3, it will still be considered by our methods because $x_1 + (x_2 + x_3)$ is equivalent to $(x_1 + x_3) + x_2$, and the sub-expression $x_1 + x_3$ is part of a valid decomposition. Allowing such equivalences within the implementation of substitution partially addresses the problem that our theory does not respect semantic equivalence, which is a necessary consequence of Theorem 1.

4.2 Analyzing Proxy Use

Algorithm 1 describes a general technique for detecting (ϵ, δ)-proxy use in expression programs. In addition to the parameters and expression, it takes as input a description of the distribution governing the feature variables X and Z. In practice this will nearly always consist of an empirical sample, but for the sake of presentation we simplify here by assuming the distribution is explicitly given. In Section D.2, we describe how the algorithm can produce estimates from empirical samples.

The algorithm proceeds by enumerating sub-expressions of the given program. For each sub-expression e appearing in p, ProxyDetect computes the set of positions at which e appears. If e occurs multiple times, we consider all possible subsets of occurrences as potential decompositions. It then iterates over all combinations of these positions, and creates a decomposition for each one to test for (ϵ, δ)-proxy use. Whenever the provided thresholds are exceeded, the decomposition is added to the return set. This proceeds until there are no more subterms to consider. While not efficient in the worst-case, this approach is both sound and complete with respect to Definition 9.

Theorem 2 (Detection soundness). *Any decomposition (p_1, p_2) returned by ProxyDetect$(p, \mathbf{X}, \epsilon, \delta)$ is a decomposition of the input program p and had to pass the ϵ, δ thresholds, hence is a (ϵ, δ)-proxy use.*

Theorem 3 (Detection completeness). *Every decomposition which could be a (ϵ, δ)-proxy use is enumerated by the algorithm. Thus, if (p_1, p_2) is a decomposition of p with $\iota(p_1, p_2) \geq d$ and $d([\![p_1]\!](\mathbf{X}), Z) \geq \epsilon$, it will be returned by ProxyDetect$(p, \mathbf{X}, \epsilon, \delta)$.*

Our detection algorithm considers single terms in its decomposition. Sometimes a large number of syntactically different proxies with weak influence might collectively have high influence. A stronger notion of program decomposition that allows a collection of multiple terms to be considered a proxy would identify such a case of proxy use but will have to search over a larger space of expressions. Exploring this tradeoff between scalability and richer proxies is an important topic for future work.

The detection algorithm runs in time $\mathcal{O}\left(|p|\, c\, (|\mathcal{D}| + k\, |\mathcal{D}|)\right)$ where $|\mathcal{D}|$ is the size of a dataset employed in the analysis, c is the number of decompositions of a program, k is the maximum number of elements in the ranges of all sub-programs ($|\mathcal{D}|$ in the worst case), and $|p|$ is the number of sub-expressions of a program. The number of decompositions varies from $\mathcal{O}(|p|)$ to $\mathcal{O}\left(2^{|p|}\right)$ depending on the type of program analyzed. Details can be found in Appendix D along with more refined bounds for several special cases.

5 REMOVING PROXY USE VIOLATIONS

In this section we present a repair algorithm for removing violations of (ϵ, δ)-Proxy Use in a model. Our approach has two parts: first (Algorithm 2) is the iterative discovery of proxy uses via the ProxyDetect procedure described in the previous section and second (Algorithm 3) is the repair of the ones found by the oracle to be violations. We describe these algorithms informally here, and Appendix C contains formal descriptions of these algorithms. The iterative discovery procedure guarantees that the returned program is free of violations (Algorithm 5). Our repair procedures operate on the expression language, so they can be applied to any model that can be written in the language. Further, our violation repair algorithm does not require knowledge of the training algorithm that produced the model. The witnesses of proxy use localize where in the program violations occur. To repair a violation we search through expressions *local* to the violation, replacing the one which has the least impact on the accuracy of the model that at the same time reduces the association or influence of the violation to below the (ϵ, δ) threshold.

At the core of our violation repair algorithm is the simplification of sub-expressions in a model that are found to be violations. Simplification here means the replacement of an expression that is not a constant with one that is. Simplification has an impact on the model's performance hence we take into account the goal of preserving utility of the machine learning program we repair. We parameterize the procedure with a measure of utility v that informs the selection of expressions and constants for simplification. We briefly discuss options and implementations for this parameter later in this section.

The repair procedure (Algorithm 3) works as follows. Given a program p and a decomposition (p_1, p_2), it first finds the best simplification to apply to p that would make (p_1, p_2) no longer a violation. This is done by enumerating expressions that are *local* to p_1 in p_2 (Line 3). Local expressions are sub-expressions of p_1 as well as p_1 itself and if p_1 is a guard in an if-then-else expression, then local expressions of p_1 also include that if-then-else's true and false branches as well as their sub-expressions. Each of the local expressions corresponds to a decomposition of p into the local expression p_1' and the context around it p_2'. For each of these local decompositions we discover the best constant, in terms of utility, to replace p_1' with (Line 4). We then make the same simplification to the original decomposition (p_1, p_2), resulting in (p_1'', p_2'') (Line 5)

Algorithm 2 Witness-driven repair.

Require: association (d), influence (ι), utility (v) measures, oracle ($\mathcal{O}$)
procedure REPAIR($p, \mathbf{X}, Z, \epsilon, \delta$)
  $P \leftarrow \{d \in \text{PROXYDETECT}(p, \mathbf{X}, Z, \epsilon, \delta) : \text{ not } \mathcal{O}(d)\}$
  if $P \neq \emptyset$ **then**
    $(p_1, p_2) \leftarrow$ element of P
    $p' \leftarrow \text{PROXYREPAIR}(p, (p_1, p_2), \mathbf{X}, Z, \epsilon, \delta)$
    return $\text{REPAIR}(p', \mathbf{X}, Z, \epsilon, \delta)$
  else
    return p

Algorithm 3 Local Repair.

Require: association (d), influence (ι), utility (v) measures
1: **procedure** PROXYREPAIR($p, (p_1, p_2), \mathbf{X}, Z, \epsilon, \delta$)
2: $R \leftarrow \{\}$
3: **for** each subprogram p_1' of p_1 **do**
4: $r^* \leftarrow$ Optimal constant for replacing p_1'
5: $(p_1'', p_2'') \leftarrow (p_1, p_2)$ with r^* subst. for p_1'
6: **if** $\iota(p_1'', p_2'') \leq \delta \vee d([\![p_1'']\!](\mathbf{X}), Z) \leq \epsilon$ **then**
7: $R \leftarrow R \cup [u/r^*]p_2'$
8: **return** $\arg\max_{p^* \in R} v(p^*)$

Using this third decomposition we check whether making the simplification would repair the original violation (Line 6), collecting those simplified programs that do. Finally, we take the best simplification of those found to remove the violation (Line 8). Details on how the optimal constant is selected is described in Appendix C.1.

Two important things to note about the repair procedure. First, there is always at least one subprogram on Line 3 that will fix the violation, namely the decomposition (p_1, p_2) itself. Replacing p_1 with a constant in this case would disassociate it from the sensitive information type. Secondly, the procedure produces a model that is smaller than the one given to it as it replaces a non-constant expression with a constant. These two let us state the following:

THEOREM 4. *Algorithm 2 terminates and returns a program that does not have any (ϵ, δ)-Proxy Use violations (instances of (ϵ, δ)-Proxy Use for which oracle returns false).*

6 EVALUATION

In this section we empirically evaluate our definition and algorithms on several real datasets. In particular, we simulate a financial services application and demonstrate a typical workflow for a practitioner using our tools to detect and repair proxy use in decision trees and linear models (§6.1). We highlight that this workflow identifies more proxy uses over a baseline procedure that simply removes features associated with a protected information type. For three other simulated settings on real data sets—contraception advertising, student assistance, and credit advertising—we describe our findings of interesting proxy uses and demonstrate how the outputs of our detection tool would allow a normative judgment oracle to determine the appropriateness of proxy uses (§6.2). In §6.3, by injecting violations into real data sets so that we have ground truth, we evaluate the completeness of our algorithm, and show a graceful degradation in accuracy as the influence of the violating proxy increases.

Models and Implementation Our implementation currently supports linear models, decision trees, random forests, and rule lists. Note that these model types correspond to a range of commonly-used learning algorithms such as logistic regression, support vector machines [10], CART [6], and Bayesian rule lists [45]. Also, these models represent a significant fraction of models used in practice in predictive systems that operate on personal information, ranging from advertising [9], psychopathy [38], criminal justice [4, 5], and actuarial sciences [32, 34]. Our prototype implementation was written in Python, and we use scikit-learn package to train the models used in the evaluation. The benchmarks we describe later in this section were recorded on a Ubuntu Desktop with 4.2 GHz Intel Core i7 and 32GB RAM.

6.1 Example Workflow

A financial services company would like to expand its client base by identifying potential customers with high income. To do so, the company hires an analyst to build a predictive model that uses age, occupation, education level, and other socio-economic features to predict whether an individual currently has a "high" or "low" income. This practice is in line with the use of analytics in the financial industry that exploit the fact that high-income individuals are more likely to purchase financial products [70].

Because demographic data is known to correlate with marital status [50], the data processor would like to ensure that the trained model used to make income predictions does not effectively infer individuals' marital status from the other demographic variables that are explicitly used. In this context, basing the decision of which clients to pursue on marital status could be perceived as a privacy violation, as other socio-economic variables are more directly related to one's interest and eligibility in various financial services.

To evaluate this scenario, we trained an income prediction model from the UCI Adult dataset which consists of roughly 48,000 rows containing economic and demographic information for adults derived from publicly-available U.S. Census data. One of the features available in this data is marital status, so we omitted it during training, and later used it when evaluating our algorithms. In this scenario, we act as the oracle in order to illustrate the kind of normative judgments an analyst would need to make as an oracle.

After training a classifier on the preprocessed dataset, we found a strong proxy for marital status in terms of an expression involving relationship status. Figure 3 visualizes all of the expressions making up the model (marked as •), along with their association and influence measures. In decision trees, sub-expressions like these coincide with decompositions in our proxy use definition; each sub-expression can be associated with a decomposition that cuts out that sub-expression from the tree, and leaves a variable in its place. The connecting lines in the figure denote the sub-expression relationship. Together with the placement of points on the influence and association scales, this produces an overview of the decision tree and the relationship of its constituent parts to the sensitive attribute.

On further examination the relationship status was essentially a finer-grained version of marital status. While not interesting

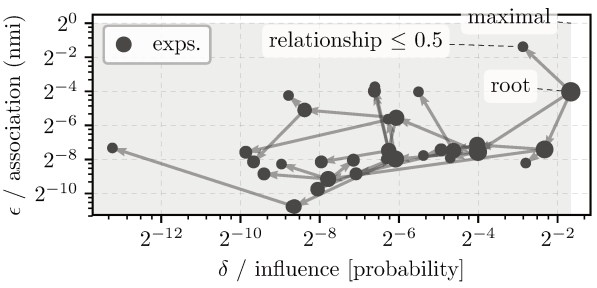


Figure 3: The association and influence of the expressions composing a decision tree trained on the UCI Adult dataset. Narrow lines designate the sub-expression relationship. Shaded area designates the feasible values for association and influence between none, and maximal. Marker size denotes the relative size of the sub-expressions pictured.

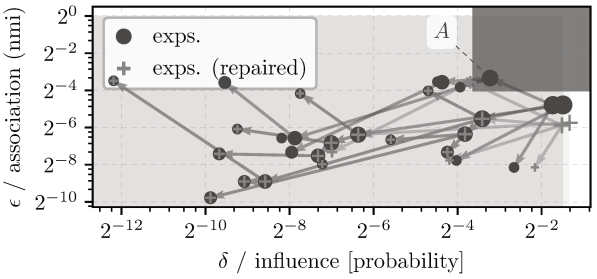


Figure 4: Decision tree trained on the UCI Adult dataset but with the `relationship` attribute removed (•), and the repaired version (+) of the same tree. Dark area in the upper-left designates the thresholds used in repair.

in itself, this occurrence demonstrates an issue with black-box use of machine learning without closely examining the structure of the data. In particular, one can choose to remove this feature, and the model obtained after retraining will make predictions that have low association with marital status. However, one sub-model demonstrated relatively strong proxy use ($\epsilon = 0.1, \delta = 0.1$): $\texttt{age} \leq 31$ and $\texttt{sex} = 0$ and $\texttt{capital\_loss} \leq 1882.50$ (labeled A in Figure 4). This demonstrates that simply removing a feature does not ensure that proxies are removed. When the model is retrained, the learning algorithm might select new computations over other features to embed in the model, as it did in this example. Also, note that the new proxy combines three additional features. Eliminating all of these features from the data could impact model performance. Instead we can use our repair algorithm to remove the proxy: we designate the unacceptable ϵ, δ thresholds (the darkest area in Figure 4) and repair any proxies in that range. The result is the decision tree marked with + in the figure. Note that this repaired version has no sub-expressions in the prohibited range and that most of the tree remains unchanged (the • and + markers largely coincide).

6.2 Other Case Studies

We now briefly discuss interesting examples for proxy use from other case studies, demonstrating how our framework aids normative use privacy judgments.

Targeted contraception advertising We consider a scenario in which a data processor wishes to show targeted advertisements for contraceptives to females. We evaluated this scenario using data collected for the 1987 National Indonesia Contraceptive Survey [1], which contains a number of socio-economic features, including feature indicating whether the individual's religious beliefs were Islam. A decision tree trained on this dataset illustrates an interesting case of potential use privacy via the following proxy for religion: $\mathit{ite}(\texttt{educ} < 4 \wedge \texttt{nchild} \leq 3 \wedge \texttt{age} < 31, \texttt{no}, \texttt{yes})$. This term predicts that women younger than 31, with below-average education background and fewer than four children will not use contraception. In fact, just the "guard" term $\texttt{educ} < 4$ alone is more closely associated with religion, and its influence on the model's output is nearly as high. This reveals a surprising association between education levels and religion leading to a potentially concerning case of proxy use.

Student assistance A current trend in education is the use of predictive analytics to identify students who are likely to benefit from certain types of interventions [31, 39]. We look at a scenario where a data processor builds a model to predict whether a secondary school student's grades are likely to suffer, based on a range of demographic features, social information, and academic information. To evaluate this scenario, we trained a model on the UCI Student Alcohol Consumption dataset [11], with alcohol use as the sensitive feature. Our algorithm found the following proxy for alcohol use: $\texttt{studytime} < 2$. This finding suggests that this instance of proxy use can be deemed an appropriate use, and not a privacy violation, as the amount of time a student spends studying is clearly relevant to their academic performance.

Credit advertisements We consider a situation where a credit card company wishes to send targeted advertisements for credit cards based on demographic information. In this context, the use of health status for targeted advertising is a legitimate privacy concern [18]. To evaluate this scenario, we trained a model to predict interest in credit cards using the PSID dataset. From this, we trained two models: one that identifies individuals with student loans and another that identifies individuals with existing credit cards as the two groups to be targeted. The first model had a number of instances of proxy use. One particular subcomputation that was concerning was a subtree of the original decision tree that branched on the number of children in the family. This instance provided negative outcomes to individuals with more children, and may be deemed inappropriate for use in this context. In the second model, one proxy was a condition involving income $\texttt{income} \leq 33315$. The use of income in this context is justifiable, and therefore this may be regarded as not being a use privacy violation.

6.3 Detection and Repair

For the remainder of the section we focus on evaluating the performance and efficacy of the detection and repair algorithms. We begin by exploring the impact of the dataset and model size on the detection algorithm's runtime.

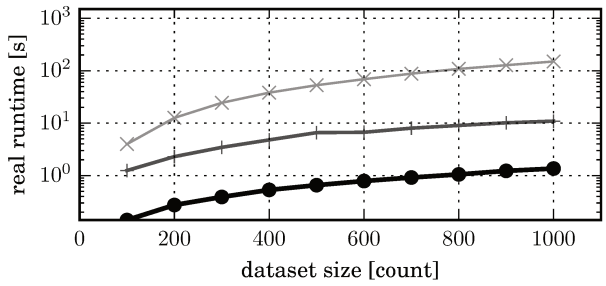


Figure 5: Worst-case detection algorithm run-time (average of 5 runs) as a function of input dataset size. Influence and association computed on each decomposition (hence worst-case). The models are decision tree(○), random forest(+), and logistic regression(×) trained on the UCI Adult dataset.

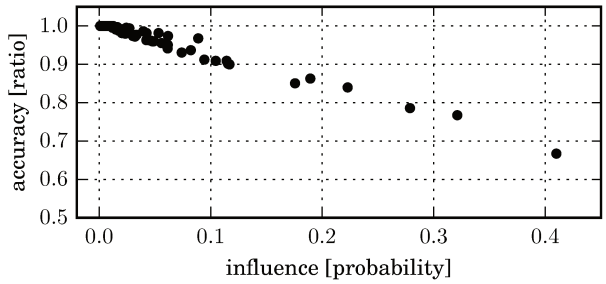


Figure 6: Repaired accuracy vs. influence of proxy during repair of a synthetic proxy inserted into random positions of a decision tree trained on the UCI Student Alcohol Consumption dataset. Accuracy is agreement to non-repaired model. The synthetic model is a (1.0)-proxy for alcohol use, inserted into a decision tree predicting student grade. Repair is configured for $(0.01, 0.01)$-proxy use removal. Note that other proxies (if they exist) are not repaired in this experiment.

Figure 5 demonstrates the runtime of our detection algorithm on three models trained on the UCI Adult dataset vs. the size of the dataset used for the association and influence computations. The algorithm here was forced to compute the association and influence metrics for each decomposition (normally influence can be skipped if association is below threshold) and thus represents a worst-case runtime. The runtime for the random forest and decision tree scales linearly in dataset size due to several optimizations. The logistic regression does not benefit from these and scales quadratically. Further, runtime for each model scales linearly in the number of decompositions , but logistic regression models contain an exponential number of decompositions as a function of their size.

To determine the completeness of our detection algorithm we inserted a proxy in a trained model to determine whether we can detect it. To do this, we used the UCI Student Alcohol Consumption dataset to train two decision trees: one to predict students' grades, and one to predict alcohol consumption. We then inserted the second tree into random positions of the first tree thereby introducing a proxy for alcohol consumption. We observed that in each case, we were able to detect the introduced proxy. While not interesting in itself due to our completeness theorem, we used this experiment to explore how much utility is actually lost due to repair. We evaluate our repair algorithm on a set of similar models with inserted violations of various influence magnitude. The results can be seen in Figure 6. We can see that the accuracy (i.e., ratio of instances that have agreement between repaired and unrepaired models) falls linearly with the influence of the inserted proxy. This implies that repair of less influential proxies will incur a smaller accuracy penalty than repair of more influential proxies. In other words, our repair methods do not unduly sacrifice accuracy when repairing only minor violations.

A point not well visible in this figure is that occasionally repair incurs no loss of utility. This is due to our use of the scikit-learn library for training decision trees as it does not currently support pruning unnecessary nodes. Occasionally such nodes introduce associations without improving the model's accuracy. These nodes can be replaced by constants without loss. We have also observed this in some of our case studies.

7 RELATED WORK

7.1 Definition

Minimizing disclosures In the computer science literature, privacy has been thought of as the ability to protect against undesired flows of information to an adversary. Much of the machinery developed in cryptography, such as encryption, anonymous communication, private computation, and database privacy have been motivated by such a goal. Differential privacy [25] is one of the main pillars of privacy research in the case of computations over data aggregated from a number of individuals, where any information gained by an adversary observing the computation is not caused by an individual's participation. However, none of these technologies cover the important setting of individual-level data analytics, where one may want to share some information while hiding others from adversaries with arbitrary background knowledge. This absence is with good reason, as in the general case it is impossible to prevent flows of knowledge from individual-level data, while preserving the utility of such data, in the presence of arbitrary inferences that may leverage the background knowledge of an adversary [21]. In this work, we do not attempt to solve this problem either.

Nevertheless, the setting of individual level data analytics is pervasive, especially in the case of predictive systems that use machine learning. Since these systems are largely opaque, even developers do not have a handle on information they may be inadvertently using via inferences. Therefore, in this work, we make the case for proxy use restrictions in data driven systems and develop techniques to detect and repair violations of proxy use. Restrictions on information use, however do not supplant the need for other privacy enhancing technologies geared for restricting information collection and disclosure, which may be useful in conjunction with the enforcement of use restrictions. For example, when machine learning models are trained using personal data, it is desirable to minimize disclosures pertaining to individuals in the training set, and to reduce the use of protected information types for the individuals the models are applied to.

Identifying explicit use The privacy literature on use restrictions has typically focused on explicit use of protected information types, not on proxy use (see Tschantz et al. [64] for a survey and Lipton and Regan [46]). Recent work on discovering personal data use by black-box web services focuses mostly on explicit use of protected information types by examining causal effects [16, 44]; some of this work also examines associational effects [43, 44]. Associational effects capture some forms of proxy use but not others as we argued in Section 3.

7.2 Detection and Repair Models

Our detection algorithm operates with white-box access to the prediction model. Prior work requires weaker access assumptions.

Access to observational data Detection techniques working under an associative use definition [30, 63] usually only require access to observational data about the behavior of the system.

Access to black-box experimental data Detection techniques working under an explicit use definition of information use [16, 44] typically require experimental access to the system. This access allows the analyst to control some inputs to the system and observe relevant outcomes.

The stronger white-box access level allows us to decompose the model and trace an intermediate computation that is a proxy. Such traceability is not afforded by the weaker access assumptions in prior work. Thus, we explore a different point in the space by giving up on the weaker access requirement to gain the ability to trace and repair proxy use.

Tramèr et al. [63] solve an important orthogonal problem of efficiently identifying populations where associations may appear. Since our definition is parametric in the choice of the population, their technique could allow identifying relevant populations for further analysis using our methods.

Repair Removal of violations of privacy can occur at different points of the typical machine learning pipeline. Adjusting the training dataset is the most popular approach, including variations that relabel only the class attribute [48], modify entire instances while maintaining the original schema [30], and transform the dataset into another space of features [24, 72]. Modifications to the training algorithm are specific to the trainer employed (or to a class of trainers). Adjustments to Naive Bayes [7] and trainers amiable to regularization [42] are examples. Several techniques for producing differentially-private machine learning models modify trained models by perturbing coefficients [3, 8]. Other differentially-private data analysis techniques [26] instead perturb the output by adding symmetric noise to the true results of statistical queries. All these repair techniques aim to minimize associations or inference from the outcomes rather than constrain use.

8 DISCUSSION

Beyond strict decomposition Theorem 1 shows that a definition satisfying natural semantic properties is impossible. This result motivates our syntactic definition, parameterized by a programming language and a choice of program decomposition. In our implementation, the choice of program decomposition is strict. It only considers single terms in its decomposition. However, proxies may be distributed across different terms in the program. As discussed in Section 4.1, single term decompositions can also deal with a restricted class of such distributed proxies. Our implementation does not identify situations where each of a large number of syntactically different proxies have weak influence but together combine to result in high influence. A stronger notion of program decomposition that allows a collection of multiple terms to be considered a proxy would identify such a case of proxy use.

The choice of program decomposition also has consequences for the tractability of the detection and repair algorithms. The detection and repair algorithms presented in this paper currently enumerate through all possible subprograms in the worst case. Depending on the flexibility of the language chosen and the model[1] being expressed there could be an exponentially large number of subprograms, and our enumeration would be intractable.

Important directions of future work are therefore organized along two thrusts. The first thrust is to develop more flexible notions of program decompositions that identify a wide class of proxy uses for other kinds of machine learning models, including deep learning models that will likely require new kinds of abstraction techniques due to their large size. The second thrust is to identify scalable algorithms for detecting and repairing proxy use for these flexible notions of program decompositions.

Data and access requirements Our definitions and algorithms require (i) a specification of which attributes are protected, (ii) entail reasoning using data about these protected information types for individuals, and (iii) white box access to models and a representative dataset of inputs. Obtaining a complete specification of protected information types can be challenging when legal requirements and privacy expectations are vague regarding protected information types. However, in many cases, protected types are specified in laws and regulations governing the system under study (e.g., HIPAA, GDPR), and also stated in the data processor's privacy policies.

Further, data about protected information types is often not explicitly collected. Pregnancy status, for example, would rarely find itself as an explicit feature in a purchases database (though it was the case in the Target case). Therefore, to discover unwanted proxy uses of protected information types, an auditor might need to first infer the protected attribute from the collected data to the best extent available to them. Though it may seem ethically ambiguous to perform a protected inference in order to (discover and) prevent protected inferences, it is consistent with the view that privacy is a function of both information and the purpose for which that information is being used [65][2]. In our case, the inference and use of protected information by an auditor has a different (and ethically justified) purpose than potential inferences in model being audited. Further, protected information has already been used by public and private entities in pursuit of social good: affirmative action requires the inference or explicit recording of minority membership, search engines need to infer suicide tendency in order to show suicide prevention information in their search results[60], health conditions can potentially be detected early from search logs of affected individuals [56]. Supported by law and perception of public

[1] Though deep learning models can be expressed in the example language presented in this paper, doing so would result in prohibitively large programs.

[2] This principle is exemplified by law in various jurisdictions including the PIPEDA Act in Canada [54], and the HIPAA Privacy Rule in the USA [55].

good, we think it justified to expect system owners be cooperative in providing the necessary information or aiding in the necessary inference for auditing.

Finally, in order to mitigate concerns over intellectual property due to access requirements for data and models, the analyst will need to be an internal auditor or trusted third party; existing privacy-compliance audits (Sen et al. [58]) that operate under similar requirements could be augmented with our methods.

Normative judgments Appropriateness decisions by the analyst will be made in accordance with legal requirements and ethical norms. Operationally, this task might fall on privacy compliance teams. In large companies, such teams include law, ethics, and technology experts. Our work exposes the specific points where these complex decisions need to be made. In our evaluation, we observed largely human-interpretable witnesses for proxies. For more complex models, additional methods from interpretable machine learning might be necessary to make witnesses understandable.

Another normative judgment is the choice of acceptable ϵ, δ parameters. Similar to differential privacy, the choice of parameters requires identifying an appropriate balance between utility and privacy. Our quantitative theory could provide guidance to the oracle on how to prioritize efforts, e.g., by focusing on potentially blatant violations (high ϵ, δ values).

9 CONCLUSION

We develop a theory of use privacy in data-driven systems. Distinctively, our approach constrains not only the direct use of protected information types but also their proxies (i.e. strong predictors), unless allowed by exceptions justified by ethical considerations.

We formalize proxy use and present a program analysis technique for detecting it in a model. In contrast to prior work, our analysis is white-box. The additional level of access enables our detection algorithm to provide a witness that localizes the use to a part of the algorithm. Recognizing that not all instances of proxy use of a protected information type are inappropriate, our theory of use privacy makes use of a normative judgment oracle that makes this appropriateness determination for a given witness. If the proxy use is deemed inappropriate, our repair algorithm uses the witness to transform the model into one that does not exhibit proxy use. Using a corpus of social datasets, our evaluation shows that these algorithms are able to detect proxy use instances that would be difficult to find using existing techniques, and subsequently remove them while maintaining acceptable classification performance.

Acknowledgments We would like to thank Amit Datta, Sophia Kovaleva, and Michael C. Tschantz for their thoughtful discussions throughout the development of this work. We thank our shepherd Aylin Caliskan and anonymous reviewers for their numerous suggestions that improved this paper.

This work was developed with the support of NSF grants CNS-1704845, CNS-1064688 as well as by DARPA and the Air Force Research Laboratory under agreement number FA8750-15-2-0277. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes not withstanding any copyright notation thereon. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of DARPA, the Air Force Research Laboratory, the National Science Foundation, or the U.S. Government.

REFERENCES

[1] 2013. Indonesia - National Contraceptive Prevalence Survey 1987. (2013). http://microdata.worldbank.org/index.php/catalog/1398/study-description (Accessed Nov 11, 2016).
[2] Paul Barford, Igor Canadi, Darja Krushevskaja, Qiang Ma, and S. Muthukrishnan. 2014. Adscape: Harvesting and Analyzing Online Display Ads. In *Proceedings of the 23rd International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 597–608.
[3] Raef Bassily, Adam Smith, and Abhradeep Thakurta. 2014. Private Empirical Risk Minimization: Efficient Algorithms and Tight Error Bounds. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*. 464–473.
[4] Richard Berk and Justin Bleich. 2014. Forecasts of Violence to Inform Sentencing Decisions. *Journal of Quantitative Criminology* 30, 1 (2014), 79–96.
[5] Richard A. Berk, Susan B. Sorenson, and Geoffrey Barnes. 2016. Forecasting Domestic Violence: A Machine Learning Approach to Help Inform Arraignment Decisions. *Journal of Empirical Legal Studies* 13, 1 (2016), 94–115.
[6] Leo Breiman. 2001. Random Forests. *Mach. Learn.* 45, 1 (Oct. 2001), 5–32.
[7] Toon Calders and Sicco Verwer. 2010. Three naive Bayes approaches for discrimination-free classification. *Data Mining and Knowledge Discovery* 21, 2 (2010), 277–292.
[8] Kamalika Chaudhuri, Claire Monteleoni, and Anand D. Sarwate. 2011. Differentially Private Empirical Risk Minimization. *Journal of Machine Learning Research* 12 (2011), 1069–1109.
[9] David Maxwell Chickering and David Heckerman. 2000. A Decision Theoretic Approach to Targeted Advertising. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI'00)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 82–88.
[10] Corinna Cortes and Vladimir Vapnik. 1995. Support-Vector Networks. *Mach. Learn.* 20, 3 (Sept. 1995), 273–297.
[11] Paulo Cortez and Alice Maria Goncalves Silva. 2008. Using data mining to predict secondary school student performance. Technical Report, Department of Computer Science, University of Camerino. (2008).
[12] Thomas M Cover and Joy A Thomas. 2012. *Elements of information theory*. John Wiley & Sons.
[13] Anupam Datta, Matthew Fredrikson, Gihyuk Ko, Piotr Mardziel, and Shayak Sen. 2017. Use Privacy in Data-Driven Systems: Theory and Experiments with Machine Learnt Programs. *arXiv preprint arXiv:1705.07807* (2017).
[14] Anupam Datta, Shayak Sen, and Yair Zick. 2016. Algorithmic Transparency via Quantitative Input Influence: Theory and Experiments with Learning Systems. In *Proceedings of IEEE Symposium on Security & Privacy 2016*.
[15] A. Datta, M.C. Tschantz, and A. Datta. 2015. Automated Experiments on Ad Privacy Settings: A Tale of Opacity, Choice, and Discrimination. In *Proceedings on Privacy Enhancing Technologies (PoPETs 2015)*. 92–112.
[16] Amit Datta, Michael Carl Tschantz, and Anupam Datta. 2015. Automated Experiments on Ad Privacy Settings: A Tale of Opacity, Choice, and Discrimination. In *Proceedings on Privacy Enhancing Technologies (PoPETs)*. De Gruyter Open.
[17] Wendy Davis. 2016. FTC's Julie Brill Tells Ad Tech Companies To Improve Privacy Protections. (2016). http://www.mediapost.com/publications/article/259210/ftcs-julie-brill-tells-ad-tech-companies-to-impro.html Accessed Nov 11, 2016.
[18] Pam Dixon and Robert Gellman. 2014. The Scoring of America: How Secret Consumer Scores Threaten Your Privacy and Your Future. (2014). http://www.worldprivacyforum.org/wp-content/uploads/2014/04/WPF-Scoring-of-America-April2014-fs.pdf Accessed: 2016-11-05.
[19] Charles Duhigg. 2012. How Companies Learn Your Secrets. (2012). http://www.nytimes.com/2012/02/19/magazine/shopping-habits.html (Accessed Aug 13, 2016).
[20] Olive Jean Dunn. 1959. Estimation of the Medians for Dependent Variables. *The Annals of Mathematical Statistics* 30, 1 (03 1959), 192–197.
[21] Cynthia Dwork. 2006. Differential Privacy. In *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, P (Lecture Notes in Computer Science)*, Vol. 4052. Springer, 1–12.
[22] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard Zemel. 2012. Fairness Through Awareness. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference (ITCS '12)*. ACM, New York, NY, USA, 214–226.
[23] C. Dwork, M. Hardt, T. Pitassi, O. Reingold, and R. Zemel. 2012. Fairness Through Awareness. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference (ITCS 2012)*. 214–226.
[24] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard S. Zemel. 2011. Fairness Through Awareness. *Computing Research Repository (CoRR)* (2011).
[25] Cynthia Dwork, Frank Mcsherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating Noise to Sensitivity in Private Data Analysis. In *Theory of Cryptography Conference*. Springer, 265–284.

[26] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating Noise to Sensitivity in Private Data Analysis. In *TCC*. 265–284.

[27] Steven Englehardt, Christian Eubank, Peter Zimmerman, Dillon Reisman, and Arvind Narayanan. 2014. Web Privacy Measurement: Scientific principles, engineering platform, and new results. Manuscript posted at http://randomwalker.info/publications/WebPrivacyMeasurement.pdf. (June 2014). Accessed Nov. 22, 2014.

[28] European Commission. 2016. General Data Protection Regulation (GDPR). Regulation (EU) 2016/679, L119. (May 2016).

[29] Michael Feldman, Sorelle A. Friedler, John Moeller, Carlos Scheidegger, and Suresh Venkatasubramanian. 2015. Certifying and Removing Disparate Impact. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '15)*. ACM, New York, NY, USA, 259–268.

[30] Michael Feldman, Sorelle A. Friedler, John Moeller, Carlos Scheidegger, and Suresh Venkatasubramanian. 2015. Certifying and Removing Disparate Impact. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*.

[31] Nicole Freeling. 2016. How Big Data is helping students graduate on time. (2016). https://www.universityofcalifornia.edu/news/how-big-data-helping-students-graduate-time (Accessed Nov 11, 2016).

[32] Edward W. Frees, Richard A. Derrig, and Glenn Meyers. 2014. *Predictive Modeling Applications in Actuarial Science*. Cambridge University Press.

[33] Deepak Garg, Limin Jia, and Anupam Datta. 2011. Policy auditing over incomplete logs: theory, implementation and applications. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*.

[34] Adrian Gepp, J. Holton Wilson, Kuldeep Kumar, and Sukanto Bhattacharya. 2012. A Comparative Analysis of Decision Trees Vis-a-vis Other Computational Data Mining Techniques in Automotive Insurance Fraud Detection. *Journal of Data Science* 10, 3 (2012), 537–561.

[35] Saikat Guha, Bin Cheng, and Paul Francis. 2010. Challenges in Measuring Online Advertising Systems. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement (IMC '10)*. ACM, New York, NY, USA, 81–87.

[36] Aniko Hannak, Piotr Sapiezynski, Arash Molavi Kakhki, Balachander Krishnamurthy, David Lazer, Alan Mislove, and Christo Wilson. 2013. Measuring Personalization of Web Search. In *Proceedings of the 22nd International Conference on World Wide Web (WWW '13)*. ACM, New York, NY, USA, 527–538.

[37] Aniko Hannak, Gary Soeller, David Lazer, Alan Mislove, and Christo Wilson. 2014. Measuring Price Discrimination and Steering on E-commerce Web Sites. In *Proceedings of the 2014 Conference on Internet Measurement Conference (IMC '14)*. ACM, New York, NY, USA, 305–318.

[38] Robert Hare. 2003. *Manual For the Revised Psychopathy Checklist*. Multi-Health Systems.

[39] Benjamin Harold. 2016. The Future of Big Data and Analytics in K-12 Education. (1 2016).

[40] Andrew Hilts, Christopher Parsons, and Jeffrey Knockel. 2016. Every Step You Fake: A Comparative Analysis of Fitness Tracker Privacy and Security. (2016). https://openeffect.ca/fitness-tracker-privacy-and-security/ Accessed Nov 11, 2016.

[41] Wassily Hoeffding. 1963. Probability Inequalities for Sums of Bounded Random Variables. *J. Amer. Statist. Assoc.* 58, 301 (March 1963), 13–30.

[42] Toshihiro Kamishima, Shotaro Akaho, and Jun Sakuma. 2011. Fairness-aware learning through regularization approach. In *Proceedings of the Workshop on Privacy Aspects of Data Mining*.

[43] Mathias Lécuyer, Guillaume Ducoffe, Francis Lan, Andrei Papancea, Theofilos Petsios, Riley Spahn, Augustin Chaintreau, and Roxana Geambasu. 2014. XRay: Enhancing the Web's Transparency with Differential Correlation. In *Proceedings of the 23rd USENIX Conference on Security Symposium (SEC'14)*. USENIX Association, Berkeley, CA, USA, 49–64.

[44] Mathias Lecuyer, Riley Spahn, Yannis Spiliopolous, Augustin Chaintreau, Roxana Geambasu, and Daniel Hsu. 2015. Sunlight: Fine-grained Targeting Detection at Scale with Statistical Confidence. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15)*. ACM, New York, NY, USA, 554–566.

[45] Benjamin Letham, Cynthia Rudin, Tyler H. McCormick, and David Madigan. 2015. Interpretable classifiers using rules and Bayesian analysis: Building a better stroke prediction model. *Ann. Appl. Stat.* 9, 3 (09 2015), 1350–1371.

[46] Richard J. Lipton and Kenneth W. Regan. 2016. Making Public Information Secret. (2016). https://rjlipton.wordpress.com/2016/05/20/making-public-information-secret/ Accessed Aug 13, 2016.

[47] Changchang Liu, Supriyo Chakraborty, and Prateek Mittal. 2016. Dependence Makes You Vulnerable: Differential Privacy Under Dependent Tuples. In *Network and Distributed System Security Symposium (NDSS)*. The Internet Society.

[48] Binh Thanh Luong, Salvatore Ruggieri, and Franco Turini. 2011. k-NN As an Implementation of Situation Testing for Discrimination Discovery and Prevention. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*.

[49] Teena Maddox. 2016. The Dark Side of Wearables. (2016). http://www.techrepublic.com/article/the-dark-side-of-wearables-how-theyre-secretly-jeopardizing-your-security-and-privacy/ Accessed Nov 11, 2016.

[50] Sumaria Mohan-Neill, Indira Neill Hoch, and Meng li. 2014. An Analysis of Us Household Socioeconomic Profiles Based on Marital Status and Gender. *Journal of Economics and Economic Education Research* 3 (9 2014).

[51] Helen Nissenbaum. 2009. *Privacy in Context: Technology, Policy, and the Integrity of Social Life*. Stanford University Press.

[52] Helen Nissenbaum and Heather Patterson. 2016. A Value for n-Person Games. In *Quantified: Biosensing Technologies in Everyday Life*. MIT Press, 80–100.

[53] The President's Council of Advisors on Science and Technology. 2014. *Big Data and Privacy: A Technological Perspective*. Technical Report. Executive Office of the President.

[54] Office of the Privacy Commissioner of Canada. 2015. PIPEDA legislation and regulations. (2015). https://www.priv.gc.ca/en/privacy-topics/privacy-laws-in-canada/the-personal-information-protection-and-electronic-documents-act-pipeda/ Accessed May 15, 2017.

[55] Office for Civil Rights. 2003. Summary of the HIPAA Privacy Rule. OCR Privacy Brief, U.S. Department of Health and Human Services. (2003).

[56] John Paparrizos, Ryen W. White, and Eric Horvitz. 2016. Screening for Pancreatic Adenocarcinoma Using Signals From Web Search Logs: Feasibility Study and Results. *Journal of Oncology Practice* 12, 8 (2016), 737–744. https://doi.org/10.1200/JOP.2015.010504 PMID: 27271506.

[57] Frank Pasquale. 2015. *The Black Box Society: The Secret Algorithms That Control Money and Information*. Harvard University Press, Cambridge, MA, USA. http://www.hup.harvard.edu/catalog.php?isbn=9780674368279

[58] Shayak Sen, Saikat Guha, Anupam Datta, Sriram K. Rajamani, Janice Tsai, and Jeannette M. Wing. 2014. Bootstrapping Privacy Compliance in Big Data Systems. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy (SP '14)*. IEEE Computer Society, Washington, DC, USA, 327–342.

[59] Geoffrey Smith. 2015. Recent Developments in Quantitative Information Flow (Invited Tutorial). In *Proceedings of the 2015 30th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS) (LICS '15)*. IEEE Computer Society, Washington, DC, USA, 23–31.

[60] S.E. Smith. 2015. How do search engines respond when you Google 'suicide'? (2015). https://www.dailydot.com/via/germanwings-suicide-hotline/ Accessed May 15, 2017.

[61] Daniel J. Solove. 2006. A Taxonomy of Privacy. *University of Pennsylvania Law Review* 154, 3 (Jan. 2006), 477–560.

[62] Hugo Teufel III. 2008. Privacy Policy Guidance Memorandum: The Fair Information Practice Principles: Framework for Privacy Policy at the Department of Homeland Security. Memorandum Number: 2008-01. (Dec. 2008). https://www.dhs.gov/xlibrary/assets/privacy/privacy$_{p}$olicyguide$_{2}$008-01.pdf

[63] Florian Tramèr, Vaggelis Atlidakis, Roxana Geambasu, Daniel J. Hsu, Jean-Pierre Hubaux, Mathias Humbert, Ari Juels, and Huang Lin. 2015. Discovering Unwarranted Associations in Data-Driven Applications with the FairTest Testing Toolkit. *CoRR* abs/1510.02377 (2015).

[64] Michael Carl Tschantz, Anupam Datta, and Jeannette M. Wing. 2012. Formalizing and Enforcing Purpose Restrictions in Privacy Policies. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*. Washington, DC, USA, 176–190.

[65] Michael Carl Tschantz, Anupam Datta, and Jeannette M. Wing. 2012. Formalizing and Enforcing Purpose Restrictions in Privacy Policies. In *IEEE Symposium on Security and Privacy, SP 2012, 21-23 May 2012, San Francisco, California, USA*. 176–190.

[66] Joseph Turow. 2011. *The Daily You: How the New Advertising Industry Is Defining Your Identity and Your Worth*. Yale University Press.

[67] J. Turow. 2017. *The Aisles Have Eyes: How Retailers Track Your Shopping, Strip Your Privacy, and Define Your Power*. Yale University Press.

[68] Findings under the Personal Information Protection and Electronic Documents Act (PIPEDA). 2014. Use of sensitive health information for targeting of Google ads raises privacy concerns. (2014). https://www.priv.gc.ca/en/opc-actions-and-decisions/investigations/investigations-into-businesses/2014/pipeda-2014-001/ (Accessed May 15, 2017).

[69] Thomas Vissers, Nick Nikiforakis, Nataliia Bielova, and Wouter Joosen. 2014. Crying wolf? On the price discrimination of online airline tickets. In *7th Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETs 2014)*. https://lirias.kuleuven.be/handle/123456789/454872"

[70] Siva Viswanathan. 2010. Business Intelligence and Predictive Analytics for Financial Services: The Untapped Potential of Soft Information. In *Digits: Center for Digital Innovation, Technology, and Strategy "Research in Practice" Paper Series*. Robert H. Smith School of Business, University of Maryland.

[71] Craig E. Wills and Can Tatar. 2012. Understanding what they do with what they know. In *Proceedings of the 2012 ACM Workshop on Privacy in the Electronic Society*. New York, NY, USA, 13–18. http://doi.acm.org/10.1145/2381966.2381969

[72] Rich Zemel, Yu Wu, Kevin Swersky, Toni Pitassi, and Cynthia Dwork. 2013. Learning Fair Representations. In *Proceedings of the Internetional Conference on Machine Learning*.

A PROOF OF THEOREM 1

Theorem 1. *No definition of proxy use can satisfy Properties 1-4 simultaneously.*

Proof. Proof by contradiction. Assume that a definition of proxy use satisfies all four properties. Let X, Y, and Z be uniform binary random variables, such that $\Pr(Y = X \oplus Z) = 1$, but X, Y and Z are pairwise independent. By (explicit use of proxy), the model $\mathcal{A}(Y, Z) = Y \oplus Z$ has proxy use of Z. By (dummy), the model $\mathcal{A}'(Y, Z, X) = Y \oplus Z$ has proxy use of Z. Choose $f(x, z) = x \oplus z$. By our assumption earlier, $\Pr(Y = f(X, Z)) = 1$. Therefore, by (preprocessing), the model $\mathcal{A}''(Z, X) = \mathcal{A}'(f(X, Z), Z, X)$ has proxy use of Z. Note that $\mathcal{A}''(Z, X) = X \oplus Z \oplus Z = X$. Therefore, by (dummy), $\mathcal{A}'''(X) = X$ has proxy use of Z. But, by (independence), $\mathcal{A}'''$ does not have proxy use of Z. Therefore, we have a contradiction. □

The key intuition behind this result is that Property 2 requires proxy use to be preserved when an input is replaced with a function that predicts that input via composition. However, with a purely semantic view of function composition, the causal effect of the proxy can disappear. The particular example of this observation we use in the proof is $Y \oplus Z$, where Z is the protected information type. This function has proxy use of Z. However, if $X \oplus Z$ is a perfect predictor for Y, then the example can be reduced to $X \oplus Z \oplus Z = X$, which has no proxy use of Z. To overcome this impossibility result, we choose a more syntactic notion of function composition, which is tied to how the function is represented as a program, and looks for evidence of proxy use within the representation.

B ALGORITHM FOR DETECTION

In this section we provide technical details about the detection algorithm skipped from the main body of the paper. In particular, we formally define the decomposition used in the implementation, how machine learning models are translated to the term language, and how associational tests mitigate spurious results due to sampling.

B.1 Decomposition

Before we present the formal algorithm for detection, we need to develop notation for precisely denoting decompositions. Decomposition follows naturally from the subterm relation on expressions. However, as identical subterms can occur multiple times in an expression, care must be taken during substitution to distinguish between occurrences. For this reason we define substitution positionally, where the subterm of expression $e = \mathrm{op}(e_1, \ldots, e_n)$ at position q, written $e|_q$, is defined inductively:

$$\mathrm{op}(e_1, \ldots, e_n)|_q = \begin{cases} \mathrm{op}(e_1, \ldots, e_n) & \text{if } q = \epsilon \\ e_i|_{q'} & \text{if } q = iq' \wedge 1 \le i \le n \\ \mathrm{op}(e_{i_1}, \ldots, e_{i_k}) & \text{if } q = \{i_1, \ldots, i_k\} \\ \bot & \text{otherwise} \end{cases}$$

We denote q as 'positional indicator'. Specifically, q has the syntax of the following.

$\langle q \rangle ::= \epsilon \mid i\langle q \rangle \mid \{i_1, \ldots, i_k\}$

We then define the term obtained by substituting s in e at position q, written $e[s]_q$, to be the term where $e[s]_q|_q = s$, and $e[s]_q|_{q'} = e_{q'}$ for all q' that are not prefixed by q. For a sequence of positions $q_1, \ldots, q_n$ and terms $s_1, \ldots, s_n$, we write $e[s_1, \ldots, s_n]_{q_1, \ldots, q_n}$ to denote the sequential replacement obtained in order from 1 to n. Given a program $p = \lambda\vec{x}.e$, we will often write $p|_q$ or $p[s]_q$ for brevity to refer to $e|_q$ and $e[s]_q$, respectively. The set of decompositions of a program p is then defined by the set of positions q such that $p|_q \neq \bot$. Given position q, the corresponding decomposition is simply $(\lambda\vec{x}.p|_q, u, \lambda\vec{x}, u.p[u]_q)$.

Example B.1. Consider a simple model,

$$\begin{aligned} p &= \lambda x, y.\mathbf{ite}(x + y \le 0, 1, 0) \\ &= \lambda x, y.\mathbf{ite}(\le (+(x, y), 0), 1, 0) \end{aligned}$$

There are eight positions in the body expression, namely $\{\epsilon, 1, 2, 3, 11, 12, 111, 112\}$. The subexpression at position 112 is y, and $p[u]_{11} = \mathbf{ite}(u \le 0, 1, 0)$. This corresponds to the decomposition:

$$(\lambda x, y.x + y, u, \lambda x, y, u.\mathbf{ite}(u \le 0, 1, 0))$$

With this notation in place, we can formally describe the detection algorithm in Algorithm 4.

Algorithm 4 Detection for expression programs.

Require: association (d), influence(ι) measures
  procedure ProxyDetect($p, \mathbf{X}, Z, \epsilon, \delta$)
    $P \leftarrow \varnothing$
    for each term e appearing in p **do**
      $p_1 \leftarrow \lambda x_1, \ldots, x_n.e$
      $Q \leftarrow \{q \mid p|_q = e\}$
      for each $k \in [1, \ldots, |Q|], (q_1, \ldots, q_k) \in Q$ **do**
        $p_2 \leftarrow \lambda x_1, \ldots, x_n, u.p[u]_{q_1, \ldots, q_k}$
        if $\iota(p_1, p_2) \ge \delta \wedge d([\![p_1]\!](\mathbf{X}), Z) \ge \epsilon$ **then**
          $P \leftarrow P \cup \{(p_1, p_2)\}$
        end if
      end for
    end for
    return P

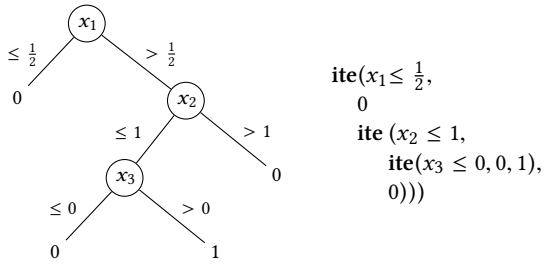


Figure 7: Decision tree and corresponding expression program.

B.2 Translation

This section describes the translation of machine learning models used in our implementation to the term language.

B.2.1 Decision trees and Rule lists. Decision trees can be written in this language as nested **ite** terms, as shown in Figure 7. The Boolean expression in each term corresponds to a guard, and the arithmetic expressions to either a proper subtree or a leaf. Bayesian rule lists are a special kinds of decision trees, where the left subtree is always a leaf.

B.2.2 Linear models. Linear regression models are expressed by direct translation into an arithmetic term, and linear classification models (e.g., logistic regression, linear support vector machines, Naive Bayes) are expressed as a single **ite** term, i.e.,

$$\text{sgn}(\vec{\mathbf{w}} \cdot \vec{\mathbf{x}} + b) \text{ becomes } \lambda\vec{\mathbf{x}}.\mathbf{ite}(\vec{\mathbf{w}} \cdot \vec{\mathbf{x}} + b \geq 0, 1, 0)$$

Importantly, the language supports n-ary operations when they are associative, and allows for rearranging operands according to associative and distributive equivalences. In other words, the language computes on terms modulo an equational theory. Without allowing such rearrangement, when a linear model is expressed using binary operators, such as $((((w_1 \times x_1) + (w_2 \times x_2)) + (w_3 \times x_3))$, then the algorithm cannot select the decomposition:

$$p_1 = \lambda\vec{\mathbf{x}}.(w_1 \times x_1) + (w_3 \times x_3)$$
$$p_2 = \lambda\vec{\mathbf{x}}, u.u + (w_2 \times x_2)$$

B.2.3 Decision Forests. Decision forests are linear models where each linear term is a decision tree. We combine the two translations described above to obtain the term language representation for decision forests.

B.3 Validity Testing

We use mutual information to determine the strength of the statistical association between $[\![p_1]\!](\mathbf{X})$ and Z. Each test of this metric against the threshold ϵ amounts to a hypothesis test against a null hypothesis which assumes that $d([\![p_1]\!](\mathbf{X}), Z) < \epsilon$. Because we potentially take this measure for each valid decomposition of p, it amounts to many simultaneous hypothesis tests from the same data source. To manage the likelihood of encountering false positives, we employ commonly-used statistical techniques. The first approach that we use is cross-validation. We partition the primary dataset n times into training and validation sets, run Algorithm 4 on each training set, and confirm the reported proxy uses on the corresponding validation set. We only accept reported uses that appear at least t times as valid.

The second approach uses bootstrap testing to compute a p-value for each estimate $\hat{d}(p_1(\mathbf{X}), Z)$, and applying Bonferroni correction [20] to account for the number of simultaneous hypothesis tests. Specifically, the bootstrap test that we apply takes n samples of $(\mathbf{X}, Z)$, $[(\hat{X}_i, \hat{Z}_i)]_{1 \leq i \leq n}$, and permutes each $\hat{X}_i, \hat{Z}_i$ to account for the null hypothesis that $\mathbf{X}$ and Z are independent. We then estimate the p-value by computing:

$$p = \frac{1}{n} \sum_{1 \leq i \leq n} \mathbb{1}(d(\hat{X}_i, \hat{Z}_i) < d([\![p_1]\!](\mathbf{X}), Z))$$

After correction, we can bound the false positive discovery rate by only accepting instances that yield $p \leq \alpha$, for sufficiently small α. We note, however, that this approach is only correct when the association strength $\epsilon = 1$, as the null hypothesis in this test assumes that $[\![p_1]\!]$ is independent of Z. To use this approach in general, we would need to sample $[(\hat{X}_i, \hat{Z}_i)]_{1 \leq i \leq n}$ under the assumption that $d(\hat{X}_i, \hat{Z}_i) \geq \epsilon$. We leave this detail to future work.

C ALGORITHMS FOR REPAIR

We now provide a formal description of the repair algorithms informally described in the paper. Algorithm 5, and 6 correspond to 2, and 3 respectively.

Algorithm 5 Witness-driven repair.

Require: association (d), influence (ι), utility (υ) measures, oracle ($\mathcal{O}$)

```
procedure REPAIR(p, X, Z, ε, δ)
    P ← {d ∈ PROXYDETECT(p, X, Z, ε, δ) : not O(d)}
    if P ≠ ∅ then
        (p1, p2) ← element of P
        p' ← PROXYREPAIR(p, (p1, p2), X, Z, ε, δ)
        return REPAIR(p', X, Z, ε, δ)
    else
        return p
    end if
```

Algorithm 6 Local Repair.

Require: association (d), influence (ι), utility (υ) measures

```
1:  procedure PROXYREPAIR(p, (p1, p2), X, Z, ε, δ)
2:      R ← {}
3:      for each decomp. (p1', p2') w/ p1' local to p1 in p2 do
4:          r* ← arg max_r υ([u/r]p2')
5:          (p1'', p2'') ← (p1, p2) with r* substituted for p1'
6:          if ι(p1'', p2'') ≤ δ ∨ d([[p1'']](X), Z) ≤ ε then
7:              p* ← [u/r*]p2'
8:              R ← R ∪ {p*}
9:          end if
10:     end for
11:     return arg max_{p*∈R} υ(p*)
```

C.1 Optimal constant selection

As constant terms cannot be examples of (ϵ, δ)-Proxy Use, there is freedom in their selections as replacements for implicated subprograms. In Algorithm 6 we pick the replacement that optimizes some measure of utility of the patched program. If the given program was constructed as a classifier, we define utility as the patched program's prediction accuracy on the data set using 0-1 loss. Similarly, if the program were a regression model, υ would correspond to mean-squared error.

If the program computes a continuous convex function, as in the case of most commonly-used regression models, then off-the-shelf convex optimization procedures can be used in this step. However, because we do not place restrictions on the functions computed by programs submitted for repair, the objective function might not satisfy the conditions necessary for efficient optimization. In these cases, it might be necessary to develop a specialized procedure for the model class. Below we describe such a procedure for the case of decision trees.

Decision trees Decision trees are typically used for classification of instances into a small number of classes C. For these models, the only replacement constants that will provide reasonable accuracy are those that belong to C, so in the worst case, the selection procedure must only consider a small finite set of candidates. However, it is possible to calculate the optimal constant with a single pass through the dataset.

Given a decomposition (p_1, p_2) of p, let ϕ be the weakest formula over p's variables such that $\forall\vec{x}.p_1(\vec{x}) = p(\vec{x})$. ϕ corresponds to the

conjoined conditions on the path in p prefixing p_1. We can then define the objective function:

$$v(r) = \sum_{\vec{x} \in \vec{X}} \mathbb{1}(\phi(\vec{x}) \rightarrow \vec{x}_c = r)$$

This objective is minimized when r matches the greatest number of class labels for samples that pass through p_1. This minimizes classification error over $\vec{X}$, and is easily computed by taking the class-label mode of training samples that satisfy ϕ.

Example C.1. Consider the tree in Figure 7, and assume that x_1 and x_2 are distributed according to $\mathcal{N}(\frac{1}{2}, 1)$, and $x_3 = x_1 + x_2$. For simplicity, assume that the class label for each instance is given exactly by the tree. Then given the decomposition:

$$p_1 = \lambda \vec{x}.\mathbf{ite}(x_3 \leq 0, 0, 1)$$
$$p_2 = \lambda \vec{x}, u.\mathbf{ite}(x_1 \leq 1/2, 0, \mathbf{ite}(x_2 \leq 1, u, 0))$$

we need to find an optimal constant to replace the subtree rooted at x_3. In this case, $\phi \stackrel{\text{def}}{=} x_1 > \frac{1}{2} \wedge x_2 \leq 1$, so we select $\vec{X}_\phi = \{\vec{x} \in \vec{X} | x_1 > \frac{1}{2} \wedge x_2 \leq 1\}$ and take the mode of the empirical sample $[p(\vec{x})]_{\vec{x} \in \vec{X}_\phi}$.

D COMPLEXITY

The complexity of the presented algorithms depend on several factors, including the type of model being analyzed, the number of elements in the ranges of sub-programs, and reachability of sub-programs by dataset instances . In this section we describe the the complexity characteristics of the detection and repair algorithms under various assumptions. Complexity is largely a property of the association and influence computations and the number of decompositions of the analyzed program. We begin by noting our handling of probability distributions as specified by datasets, several quantities of interest, discuss the complexity of components of our algorithms, and conclude with overall complexity bounds.

D.1 Distributions, datasets, and probability

It is rarely the case that one has access to the precise distribution from which data is drawn. Instead, a finite sample must be used as a surrogate when reasoning about random variables. In our formalism we wrote $X \stackrel{\$}{\leftarrow} \mathcal{P}$ to designate sampling of a value from a population. Given a dataset surrogate $\mathcal{D}$, this operation is implemented as an enumeration $\mathbf{x} \in \mathcal{D}$, with each element having probability $1/|\mathcal{D}|$. We will overload the notation and use $\mathcal{D}$ also as the random variable distributed in the manner just described. We assume here that the sensitive attribute Z is a part of the random variable X.

The following sections use the following quantities to express complexity bounds, mostly overloading prior notations:

- $\mathcal{D}$ - The number of instances in the population dataset.
- p - The number of expressions in a program p being analyzed.
- Z - The number of elements in the support of Z.
- k - The maximum number of unique elements in support of every sub-expression, that is $\max_{p' \in p} |\text{support}\,([\![p]\!]\mathcal{D})|$.
- c - The number of decompositions in a given program. We will elaborate on this quantity under several circumstances later in this section.
- b - The minimum branching factor of sub-expressions in a given program.

We will assume that the number of syntactic copies of any sub-expression in a program is no more than some constant. This means we will ignore the asymptotic effect of decompositions with multiple copies of the same sub-program p_1.

The elementary operation in our algorithms is a lookup of a probability of a value according to some random variable. We pre-compute several probabilities related to reachability and contingency tables to aid in this operation. When we write "p_1 is reached", we mean that the evaluation of p, containing p_1, on a given instance $\mathbf{X}$, will reach the sub-expression p_1 (or that p_1 needs to be evaluated to evaluate p on $\mathbf{X}$).

Probability pre-computation
For every decomposition $[\![p_2]\!]\,(X, [\![p_1]\!]X) = [\![p]\!]\,(X)$, we compute:

(1) the r.v. $([\![p_1]\!]\mathbf{X}, \mathbf{X}_Z)$ for $\mathbf{X} \stackrel{\$}{\leftarrow} \mathcal{D}$,
(2) the r.v. $\mathbf{X}|$ (p_1 is reached by $\mathbf{X}$) for $\mathbf{X} \stackrel{\$}{\leftarrow} \mathcal{D}$, and
(3) the value $Pr_{\mathbf{X} \stackrel{\$}{\leftarrow} \mathcal{D}}$ (p_1 is reached by $\mathbf{X}$).

In point (1) above we write $\mathbf{X}_Z$ to designate the sensitive attribute component of $\mathbf{X}'$, hence this point computes the r.v. representing the output of p_1 along with the sensitive attribute Z. This will be used for the association computation.

The complexity of these probability computations varies depending on circumstances. In the worst case, the complexity is $\mathcal{O}(c\mathcal{D}p)$. However, under some assumptions related to programs p and datasets $\mathcal{D}$, these bounds can be improved. We define two types of special cases which we call splitting and balanced:

Definition D.1. p is *splitting* for $\mathcal{D}$ iff it has at most a constant number of reachable op operands (arguments of op expressions).

The Decision trees are local for any dataset as they do not contain any op operands (they do contain relop operands). Further, if number of trees in random forests or number of coefficients in linear regression are held constant, then these models too are splitting for any dataset. The reasoning behind this definition is to prohibit arbitrarily large programs that do not split inputs using if-then-else expressions. It is possible to create such programs using arithmetic and boolean operations, but not using purely relational operations.

Definition D.2. p is *b-balanced* for $\mathcal{D}$ iff all but a constant number of sub-expressions e' have parent e with $b > 1$ sub-expressions which split the instances that reach them approximately equally among their children.

Balanced implies splitting as op operands do not satisfy the balanced split property hence there has to be only a constant number of them. Also, the definition is more general than necessary for the language presented in this paper where the branching factor is always 2 because the if-then-else expressions are the only ones that can satisfy the balanced split condition. Decision trees trained using sensible algorithms are usually balanced due to the branch split criteria employed preferring approximately equal splits of training instances. For the same reason, if the number of trees are held constant, then random forests are also likely to be balanced.

When p is splitting for $\mathcal{D}$, the probability computation step reduces to $\mathcal{O}\left(\mathcal{D}p^2\right)$. This stems from the fact that the number

of decompositions is asymptotically equal to the number of subexpressions (limits to operands prevent more decompositions). Further, if p is b-balanced for $\mathcal{D}$, the probability pre-computation reduces to $\mathcal{O}\left(\mathcal{D}\log_b \mathcal{D}\right)$. In the language presented $b = 2$. These bounds derive similarly to the typical divide and conquer program analysis; there are $\log_b \mathcal{D}$ layers of computation, each processing $\mathcal{D}$ instances.

D.2 Influence and Association

Our proxy definition further relies on two primary quantities used in Algorithm 1, influence and association. We describe the methods we use to compute them here.

Quantitative decomposition influence Given a decomposition (p_1, u, p_2) of p, the influence of p_1 on p_2's output is defined as:

$$\iota(p_1,p_2) \stackrel{\text{def}}{=} \mathop{\mathbb{E}_{\$}}_{X,X'\leftarrow\mathcal{D}} \left[\Pr\left(\llbracket p_2\rrbracket\left(\mathrm{X}, \llbracket p_1\rrbracket \mathrm{X}\right) \neq \llbracket p_2\rrbracket\left(\mathrm{X}, \llbracket p_1\rrbracket \mathrm{X}'\right)\right)\right]$$

This quantity requires $\mathcal{D}^2$ samples to compute in general. Each sample takes at most $\mathcal{O}(p)$ time, for a total of $\mathcal{O}\left(p\mathcal{D}^2\right)$. However, we can take advantage of the pre-computations described in the prior section along with balanced reachability criteria and limited ranges of values in expression outputs to do better. We break down the definition of influence into two components based on reachability of p_1:

$$\begin{aligned}
\iota(p_1,p_2) &\stackrel{\text{def}}{=} \mathop{\mathbb{E}_{\$}}_{\mathrm{X},\mathrm{X}'\leftarrow\mathcal{D}} \left[\Pr\left(\llbracket p_2\rrbracket\left(\mathrm{X}, \llbracket p_1\rrbracket \mathrm{X}\right) \neq \llbracket p_2\rrbracket\left(\mathrm{X}, \llbracket p_1\rrbracket \mathrm{X}'\right)\right)\right] \\
&= \mathop{\mathbb{E}_{\$}}_{\mathrm{X}\leftarrow\mathcal{D}} \left[\mathop{\mathbb{E}_{\$}}_{\mathrm{X}'\leftarrow\mathcal{D}} \left[\Pr\left(\llbracket p_2\rrbracket\left(\mathrm{X}, \llbracket p_1\rrbracket \mathrm{X}\right) \neq \llbracket p_2\rrbracket\left(\mathrm{X}, \llbracket p_1\rrbracket \mathrm{X}'\right)\right)\right]\right] \\
&= \Pr\left(p_1 \text{ not reached}\right) \cdot \mathop{\mathbb{E}}_{\mathrm{X}\stackrel{\$}{\leftarrow}\mathcal{D}|p_1 \text{ not reached}} [\cdots] \\
&\quad + \Pr\left(p_1 \text{ reached}\right) \cdot \mathop{\mathbb{E}}_{\mathrm{X}\stackrel{\$}{\leftarrow}\mathcal{D}|p_1 \text{ reached}} [\cdots] \\
&= 0 + \Pr\left(p_1 \text{ reached}\right) \cdot \\
&\quad \mathop{\mathbb{E}}_{\mathrm{X}\stackrel{\$}{\leftarrow}\mathcal{D}|p_1 \text{ reached}} \left[\mathop{\mathbb{E}_{\$}}_{\mathrm{X}\leftarrow\mathcal{D}} \left[\Pr\left(\llbracket p\rrbracket(\mathrm{X}) \neq \llbracket p_2\rrbracket\left(\mathrm{X}, \llbracket p_1\rrbracket \mathrm{X}'\right)\right)\right]\right] \\
&= \Pr\left(p_1 \text{ reached}\right) \cdot \\
&\quad \mathop{\mathbb{E}}_{\mathrm{X}\stackrel{\$}{\leftarrow}\mathcal{D}|p_1 \text{ reached}} \left[\mathop{\mathbb{E}}_{\mathrm{Y}\stackrel{\$}{\leftarrow}\llbracket p_1\rrbracket\mathcal{D}} \left[\Pr\left(\llbracket p\rrbracket(\mathrm{X}) \neq \llbracket p_2\rrbracket\left(\mathrm{X}, \mathrm{Y}\right)\right)\right]\right]
\end{aligned}$$

Note that all both random variables and one probability value in the final form of influence above have been pre-computed. Further, if the number of elements in the support of $\llbracket p_1\rrbracket X$ is bounded by k, we compute influence using $k\mathcal{D}$ samples (at most $\mathcal{D}$ for X and at most k for Y), for total time of $\mathcal{O}(kp\mathcal{D})$.

Influence can also be estimated, $\hat{\iota}$ by taking a *sample* from $\mathcal{D}\times\mathcal{D}$. By Hoeffding's inequality [41], we select the subsample size n to be at least $\log(2/\beta)/2\alpha^2$ to ensure that the probability of the error $\hat{\iota}(p_1,p_2) - \iota(p_1,p_2)$ being greater than β is bounded by α.

Association As discussed in Section 3, we use mutual information to measure the association between the output of a subprogram and Z. In our pre-computation steps we have already constructed the r.v. $(\llbracket p_1\rrbracket \mathrm{X}, \mathrm{X}_Z)$ for $\mathrm{X} \stackrel{\$}{\leftarrow} \mathcal{D}$. This joint r.v. contains both the subprogram outputs and the sensitive attribute hence it is sufficient to compute association metrics. In case of normalized mutual information, this can be done in time $\mathcal{O}(kZ)$, linear in the size of the support of this random variable.

D.3 Decompositions

The number of decompositions of a model determines the number of proxies that need to be checked in detection and repair algorithms. We consider two cases, splitting and non-splitting programs. For splitting models, the number of decompositions is bounded by the size of the program analyzed, whereas in case of non-splitting models, the number of decompositions can be exponential in the size of the model. These quantities are summarized in Table 2.

	splitting	non-splitting
worst-case general	$\mathcal{O}(p)$	$\mathcal{O}(2^p)$
linear model with (constant or f) number of coefficients	$\mathcal{O}(1)$	$\mathcal{O}\left(2^f\right)$
decision tree of height h	$\mathcal{O}\left(2^h\right)$	$\mathcal{O}\left(2^h\right)$
random forest of (constant or t) number of trees of height h	$\mathcal{O}\left(2^h\right)$	$\mathcal{O}\left(2^t 2^h\right)$

Table 2: The number of decompositions in various types of models. When we write "(constant or f)" we denote two cases: one in which a particular quantity is considered constant in a model making it satisfy the splitting condition, and one in which that same quantity is not held constant, falsifying the splitting condition.

D.4 Detection

The detection algorithm can be written $\mathcal{O}(A + B \cdot C)$, a combination of three components. A is probability pre-computation as described earlier in this section, B is the complexity of association and influence computations, and C is the number of decompositions.

The complexity in terms of the number of decompositions under various conditions is summarized in Table 3. Instantiating the parameters, the overall complexity ranges from $\mathcal{O}\left(\mathcal{D}\log_b \mathcal{D} + p^2\mathcal{D}\right)$ in case of models like balanced decision trees with a constant number of classes, to $\mathcal{O}\left(p2^p\mathcal{D}^2\right)$ in models with many values and associative expressions like linear regression. If the model size is held constant, these run-times become $\mathcal{O}\left(\mathcal{D}\log_b \mathcal{D}\right)$ and $\mathcal{O}\left(\mathcal{D}^2\right)$, respectively.

non-splitting	$\mathcal{O}(pc(\mathcal{D} + k\mathcal{D}))$
splitting	$\mathcal{O}(p(\mathcal{D}p + ck\mathcal{D}))$
b–balanced	$\mathcal{O}\left(\mathcal{D}\log_b \mathcal{D} + ckp\mathcal{D}\right)$

Table 3: The complexity of the detection algorithm under various conditions, as a function of the number of decompositions.