CaFA: Cost-aware, Feasible Attacks With Database Constraints Against Neural Tabular Classifiers

Matan Ben-Tov*, Daniel Deutch*, Nave Frost†, and Mahmood Sharif*
*Tel Aviv University †eBay
Email: matanbentov@mail.tau.ac.il, {danielde, mahmoods}@tauex.tau.ac.il, nafrost@ebay.com

***Abstract*—This work presents CaFA, a system for Cost-aware Feasible Attacks for assessing the robustness of neural tabular classifiers against adversarial examples realizable in the problem space, while minimizing adversaries' effort. To this end, CaFA leverages TabPGD—an algorithm we set forth to generate adversarial perturbations suitable for tabular data—and incorporates integrity constraints automatically mined by state-of-the-art database methods. After producing adversarial examples in the feature space via TabPGD, CaFA projects them on the mined constraints, leading, in turn, to better attack realizability. We tested CaFA with three datasets and two architectures and found, among others, that the constraints we use are of higher quality (measured via soundness and completeness) than ones employed in prior work. Moreover, CaFA achieves higher feasible success rates—i.e., it generates adversarial examples that are often misclassified while satisfying constraints—than prior attacks while simultaneously perturbing few features with lower magnitudes, thus saving effort and improving inconspicuousness. We open-source CaFA,[1] hoping it will serve as a generic system enabling machine-learning engineers to assess their models' robustness against realizable attacks, thus advancing deployed models' trustworthiness.**

1. Introduction

Evasion attacks producing adversarial examples—slightly but strategically manipulated variants of benign samples inducing misclassifications—have emerged as a technically deep challenge posing risk to safety- and security-critical deployments of machine learning (ML) [6]. For example, adversaries may inconspicuously manipulate their appearance to circumvent face-recognition systems [15], [41], [42]. As another example, attackers may introduce seemingly innocuous stickers to traffic signs, leading traffic-sign recognition models to err [20]. Such adversarial examples have also become the *de facto* means for assessing ML models' robustness (i.e., ability to withstand inference-time attacks) in adversarial settings [6], [38]. Nowadays, numerous critical applications employ ML models on tabular data, including for medical diagnosis, malware detection, fraud detection, and credit scoring [7]. Still, adversarial examples against such models remain underexplored.

1. https://github.com/matanbt/attack-tabular

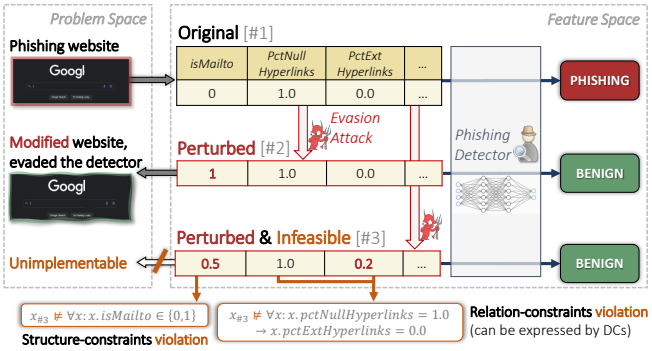


Figure 1: Adversarial examples in the problem space (e.g., a phishing website imitating Google) and the feature space (i.e., feature vectors serving as inputs to ML models). The original feature vector (***#1***) represents a website correctly detected by ML-based phishing detection. The adversary finds a minimal perturbation (***#2***) realizable as a problem-space instance while misleading the detector. Attacks, however, may also fail to satisfy data-integrity constraints (***#3***), rendering them unrealizable in the problem space. Different types of data-integrity constraints exist, including structure (defined by features' domains) and relation (defined by relations between samples and features) constraints.

In the tabular domain, ML models classify problem-space artifacts (e.g., phishing pages or malicious programs) based on their feature representations (e.g., [13], [26], [33], [38]), posing challenges to evaluating robustness against practical attacks realizable in the problem space (i.e., via an artifact causing misclassification). Some approaches for evaluating robustness against practical attacks operate directly in the problem space by manipulating (realizable) artifacts directly to evade ML models (e.g., [20], [30], [41], [42], [47]). For instance, some attacks suggest means to strategically transform malicious programs such that their features would mislead malware detectors [30], [47]. These approaches, however, are domain-specific, rendering them unsuitable for assessing robustness across multiple domains. Namely, to evaluate robustness against practical attacks in domains not previously studied, one still needs to invest manual effort and rely on domain expertise to develop an attack for the specific domain [21], [28].

In contrast, some techniques generate adversarial examples in the feature space to assess model robustness [38]. These, however, often do not accurately reflect models' robustness in practice, as they produce feature changes not realizable in the problem space due to violating data-integrity constraints [38] (Fig. 1). While approaches for incorporating constraints—e.g., in Valiant's boolean conjunctive normal form (CNF) [43], [49]—exist, as we show (§6.1 and §6.6), such constraints may lack soundness (i.e., adversarial examples satisfying the constraints often violate realizability), and the attacks have limited success, leading to misapproximation of true robustness.

Effort minimization poses another challenge for attacks on tabular data. Specifically, adversaries typically seek to minimize adversarial perturbations of original samples to maintain attack inconspicuousness, thus generating harder-to-detect adversarial examples (e.g., [22], [25], [42], [48]). Furthermore, as adversaries are typically rational and economically motivated [23], minimizing adversarial perturbations can also help minimize attacks' (monetary) costs [25]. Certain approaches incorporate domain knowledge, assigning monetary value to perturbations of different feature types [25]. These, however, cannot be applied generically across domains. More generic approaches limit the extent to which features can be perturbed (i.e., perturbations' ℓ_∞- or ℓ_2-norms) but may allow many features to be manipulated and ignore the heterogeneity of features often encountered in tabular data (e.g., binary vs. continuous features covering large ranges) [46]. Other approaches limit the amount of features that can be manipulated (i.e., perturbations' ℓ_0-norms) but may allow features to be manipulated to an arbitrary extent [8], [32], [43], [44]. Thus, there remains a need for approaches minimizing adversarial-example-generation effort in a generic manner well-suited for tabular data.

To address these challenges, we propose **CaFA**, a three-stage system for **C**ost-**a**ware **F**easible **A**ttacks to enable generic evaluation of ML models ingesting tabular data against evasion (§4). CaFA automatically mines so-called denial constraints (DCs) via databases-based techniques [14] identifying relations between features and samples. Then, CaFA perturbs samples via TabPGD—an algorithm we offer to generate evasive samples in feature space while satisfying structure constraints defined by feature developers and minimizing a newly introduced cost metric accounting for the extent to which features are manipulated and the number of features perturbed. Lastly, CaFA projects the evasive samples on the mined DCs as a mean to comply with genuine data-integrity constraints and ensure realizability. We conducted experiments with three datasets and two neural network (NN) architectures, exploring the quality of the mined constraints and the feasible success rates (i.e., portion of adversarial samples satisfying the mined constraints) attained by CaFA. Our findings show that:

- DCs better balance soundness (i.e., out-of-domain samples violate constraints) and completeness (i.e., in-domain samples satisfy constraints) than Valiant's constraints with tractable parameterization, rendering them more adequate for robustness evaluation against practical attacks (§6.1).
- CaFA, with DCs employed, attains higher feasible success rates (of over than ×25%; §6.2), while perturbing relatively limited number of features to fewer extents (§6.3) than prior attacks. CaFA also has markedly faster run times than attacks with competitive feasible success rates (§6.4).
- When DCs are *not* incorporated, TabPGD with Valiant's constraints employed attained significantly higher feasible success rates than previously found (87.6% vs. ≤60.0%), countering prior belief that integrating Valiant's integrity constraints harms attack success [43] (§6.5).
- In a realistic phishing-page detection setting, adversarial examples satisfying the mined DCs found by CaFA could be realized in the problem space more successfully than ones found via prior techniques (§6.6). Intuitively, as the databases community pushes the boundaries of constraint mining, yielding constraints with enhanced completeness and soundness, CaFA's ability to assess robustness against practical attacks would also improve.

Next, we discuss related work and background (§2) and our threat model (§3) before presenting our technical approach (§4), and results (§5–6). We wrap up with a discussion (§7) and a conclusion (§8).

2. Background and Related Work

2.1. Evasion Attacks

Our work studies evasion attacks, popularized by the work of Biggio et al. [5] and Szegedy et al. [48]. In these attacks, adversaries modify inputs at inference time, creating so-called adversarial examples, to mislead ML models. Various methods with varied assumptions (e.g., full, white-box vs. limited, query-only, black-box access) for generating adversarial examples have been proposed [6], [11], [16], [31], [35], [36]. A common method that serves as the bedstone for state-of-the-art attacks is Projected Gradient Descent (PGD) [31]. To produce adversarial examples, PGD iteratively modifies samples in the direction of the gradient while limiting the magnitude of the perturbation. Concretely, given a model with a loss function L and a sample x of class y, PGD iteratively calculates:

$$x'^{(i+1)} := x'^{(i)} + \Pi(\alpha \cdot \text{sign}(\nabla_x L(x'^{(i)}, y)))$$

where $x'^{(i)}$ is the perturbed sample in the i-th iteration ($x'^{(0)} = x$), Π is a function projecting samples per a budget (e.g., clipping to an ℓ_∞-norm ϵ-ball centered at x), and α is a step size. We propose a modification of PGD better suited for misleading models classifying tabular data (§4.2).

2.2. Challenges of Feature-Space Attacks

To evade ML models in practice, adversaries need to modify problem-space artifacts to induce misclassifications (e.g., altering phishing websites to evade detection) [38]. To the best of our knowledge, there exists no technique capable of generating problem-space attacks in a generic way, across domains. Consequently, domain-specific attacks

for generating adversarial examples in the problem space are constantly being proposed, even when attacks in closely related domains already exist. Indeed, such attacks have been proposed for domains ranging from malicious PDF detection [52] to malware detection [47], and from phishing detection [34] to traffic-sign recognition [20].

As a generic alternative to problem-space attacks, one may generate adversarial examples in the feature space that can be later mapped to problem-space instances. Such an approach faces two primary challenges. First, feature-space attacks are often *unrealizable* in the problem space [38]. This is often due to violating data-integrity constraints not accounted for during adversarial example generation (e.g., see example in §2.3). While specific attempts to produce realizable adversarial examples in the feature space exist (e.g., [21], [22]), they usually rely on domain expertise to define permissible perturbation ensuring the evasive features are realizable, rendering them non-generic. Moreover, as we find (§6.2), the heterogeneous features with varying scales and types (e.g., categorical and continuous) hinder the success of standard attacks used in prior work. Second, feature-space attacks typically require high *adversarial effort*, either due to manipulating many features (mainly, when minimizing ℓ_2- or ℓ_∞-norms) [46], or due to manipulating a few features by significant amounts (when minimizing ℓ_0-norm) [8], [32], [43], [44]. This, in turn, renders adversarial examples easy to detect [20], [22], [42], and may increase the monetary cost of crafting them in the problem space [25]. As mentioned above (§1), this work seeks to overcome these challenges.

2.3. Data-Integrity Constraints

Identifying and accounting for feature-space constraints in attacks can help improve realizability. In our work, we capture feature-space constraints with the known structural limitations of the feature space, and with the integrity constraints learned from the dataset itself, both are then integrated into our attack. Further, we evaluate and compare integrity-constraint sets by measuring the extent of violations detected (i.e. soundness) and the extent of in-domain samples admitted (i.e. completeness) (see §5.1). We focus on two constraint types—structure and relation constraints.

2.3.1. Structure Constraints. Non-realizability sometimes stems from violations of basic feature-space structural properties. For instance, in our example (Fig. 1), $x_3.isMailto = 0.5$, while $isMailto$ is a binary feature, stating whether the HTML uses the *mailto* function or not. Thus, it is unclear how x_3 can be transformed, if at all, to an evasive HTML in the problem space. To address these kind of violations, we specify the type of each feature, which can be *ordinal*, *continuous*, or *categorical*. For each feature, we then define a permissible domain (i.e., range or set of values). Additional structure constraints may stem from the feature processing (e.g., from one-hot encoding of categorical features). Such structure constraints can be typically derived from the mapping defined during feature engineering.

2.3.2. Relation Constraints. Non-realizability may also arise from violations of semantic relations between features. For instance, in our example (x_3 in Fig. 1), while the proportion of empty links ($pctNullHyperlinks$) is 100%, the proportion of external links ($pctExtHyperlinks$) is 20%, which is impossible for real-world HTMLs. Preventing this type of complex cross-feature violations requires expressive modeling. We employ the highly expressive Denial Constraints (DCs) [14] to capture relation constraints. We also refer to and compare with Valiant's constraints [49].

Valiant's Constraints. Valiant's PAC learning algorithm [49] derives a set of boolean constraints, as CNFs, from a given dataset. The algorithm and its constraints were previously used to capture domain constraints by Sheatsley et al. [43]. In this work, we employ Valiant's constraints as a baseline for feature-space constraints (§5.4).

Denial Constraints. DCs are widely used and provably expressive type of data-integrity constraints (e.g., DCs can express the most commonly used constraint types [14], [29]). DCs apply to pairs of samples in the dataset and are composed of negated conjunctions of predicates. Namely, a DC expresses a set of predicates that *cannot* hold together for a pair of samples. The set of possible DCs can be formalized as:

$$\left\{ \forall x, x' \in X . \neg \left(\bigwedge_{p \in P} p \right) \mid P \subset \Psi \right\}$$
$$\text{s.t.} \quad \Psi := \left\{ \left(x_i \diamond x'_j \right) \mid \diamond \in \{=, \neq, <, >, \leq, \geq\}, i, j \in [d] \right\}$$

where Ψ is the set of possible predicates, which can vary in different flavors of DCs. Each predicate includes a *main* tuple, x, the *other* tuple x', and an operator between these tuples, $\diamond$. The set of possible operators may naturally differ between features (e.g., categorical features can only use the operators $=, \neq$). Using this definition, DCs can also express the relation constraint violated in Fig. 1 by instantiating the *other* tuple values as constant operands. Due to their expressivity, we employ DCs as to capture feature-space relation constraints.

Approaches to *mining* constraints from a given dataset can be roughly divided into two. The first mines *hard constraints*, which requires the mined constraints to hold for *all* samples in the learned set. The other, more expressive type of constraints, namely, *soft constraints*, allows small violation rates in the learned set. In contrast to other constraints mining algorithms—such as Valiant's algorithm [49]—that mine hard constraints, we leverage algorithms for mining soft DCs due to their lower sensitivity to malformed or anomalous samples [29], [51].

2.4. Existing Attacks on Tabular Datasets

Adversarial examples in the tabular domain have gained an increasing attention in recent years. Still, these remain relatively under-explored compared to attacks in the vision and text domains. Tab. 1 summarizes and compares prior work and ours, focusing on the aforementioned challenges.

Work \ Criteria	Setting	Constraints: Structural?	Constraints: Relational?	Constraints: Automated Mining?	Cost: Type	Cost: Automated?	Domain-General?	Open Source?
Ballet et al. [3]	White-box	○	○	-	Feature importance	●	●	●
Bostani et al. [8]	White-box	●	● (via statistical correlations)	● (OPF algorithm)	ℓ_0	●	○ (only binary features)	●
Cartella et al. [12]	Black-box	◒	○	-	Feature importance	●	●	○
Keeriv et al. [25]	Black-box	●	○	-	Monetary cost	○	●	○
Mathov et al. [32]	Black-box	●	◒ (via trained encoder model)	●	ℓ_0	●	●	○
Sheatsley et al. [43]	White-box	●	● (Valiant's)	● (Valiant's)	ℓ_0	●	●	○
Sheatsley et al. [44]	White-box	●	●	◒ (manual & heuristics)	ℓ_0	●	●	○
Simonetto et al. [46]	White-box (C-PGD)	●	●	○	ℓ_∞/ℓ_2	●	●	●
Simonetto et al. [46]	Grey-box (MoEvA2)	●	●	○	ℓ_∞/ℓ_2	●	●	●
CaFA (ours)	White-box	●	● (DCs)	● (FastADC [51])	*standardized*-ℓ_∞ + ℓ_0	●	●	●

TABLE 1: Comparing prior attacks on tabular data in terms of incorporating constraints for realizability, attack-cost minimization, generality across feature domains, and open-source availability. Each cell is marked to denote whether attacks support (●), partially support (◒), or do not support (○) certain properties.

Realizability Through Constraints. Previous works on tabular attacks acknowledge the importance of adhering to constraints imposed by the problem-space when crafting adversarial samples [8], [32], [43], [44]. Relatedly, an analogous claim was made for finding better counter-factual examples by Deutch and Frost [19]. Most prior attacks have focused on structure constraints, whose significance was also discussed by Mathov et al. [32].

While many efforts acknowledge relation constraints, only a few suggested automatic and general schemes utilizing them (Tab. 1). Notably, Sheatsley et al.'s closely related work [43] uses constraints mined by Valiant's algorithm [49]. They incorporate the constraints into their attacks, for which they report low rates of misclassified adversarial samples, thus arguing that data-integrity constraints increase robustness. We later show findings contradicting Sheatsley et al.'s (§6.5). Moreover, we argue that DCs are better aligned with the genuine problem-space constraints relative to Valiant's constraints (§6.6).

There are several ways to integrate feature-space constraints into attacks. Simonetto et al. [46] minimize a penalty associated with the constraints to improve adversarial examples' compliance. Another option is applying SAT solvers to project adversarial examples onto the constrained feature space [43], [46]; however, this approach demonstrated poor feasible success rates [43]. In this work, we also leverage SAT solvers, but precede projection with an attack well-suited for tabular data (§4.2) and employ new heuristics during projection to preserve misclassification while simultaneously satisfying constraints (§4.3). Our experiments demonstrate that, prior to projection on constraints using solvers, our approach already produces fewer violations of relation constraints than past attacks, lending itself to higher feasible success rates after projection (§6.2).

Adversarial Effort (Cost). The literature varies in its definition of *cost* for tabular adversarial example attacks. Some efforts adopt metrics commonly used in computer vision (ℓ_2- or ℓ_∞-norms) [46]; some argued for the suitability of the ℓ_0 cost for tabular data [32], [43], [44]; while others formed novel cost measures (e.g., feature-importance-based cost [3], [12]). Keeriv et al. [25] offered a nuanced discussion on the importance of cost in tabular attacks, distinguishing it from the imperceptibility goal in the vision domain and describing it as the financial cost of the attack. Accordingly, to bound costs, they require manually defined financial costs for each feature. Their interpretation and incorporation of cost in the attack is conceptually close to ours, albeit our attack's cost is automatically derived.

Overall, as can be seen from Tab. 1, our proposed system, CaFA, is the first that assesses robustness to realizable attacks by incorporating constraints while simultaneously minimizing adversarial effort automatically and generically across domains, regardless of feature types. As our experiments show (§5–6), CaFA also attains higher attack success-rates at lower costs than leading attacks, providing more reliable robustness estimation.

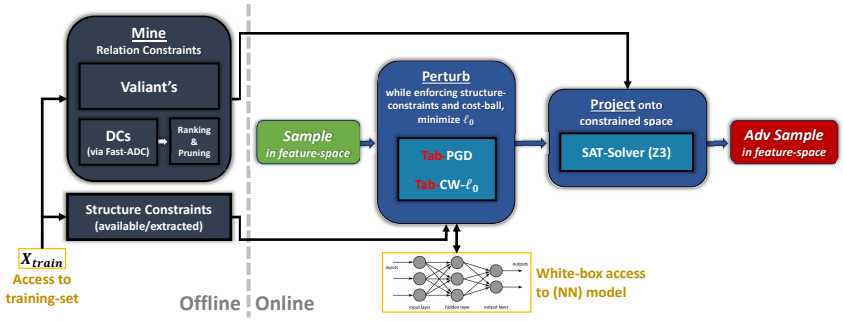
3. Threat Model

In this paper, we study white-box feature-space attacks incorporating learned constraints to produce feasible (in terms of the chosen constraint set) and cost-aware adversarial examples. We focus on targeting Neural Network (NN) models, due to their popularity [43], performance [2], and other desirable properties (e.g., lending themselves to distributed training and self-supervised learning) [7].

We study adversaries targeting NN-based tabular data classifiers, denoted by $M : X \to Y$, where $X \subseteq \mathbb{R}^d$ is the feature space and Y is the label space. X is composed of heterogeneous features, each from a different (possibly dependent) unknown distribution. We assume adversaries possess a sample $x \in X$ they seek to misclassify and a corresponding label $y \in Y$.

We consider a white-box setting where the attacker knows the classifier's architecture. The attacker also has access to the inherent structure constraints of X defined during feature engineering, as well as to a sample dataset for mining relation constraints. These assumptions, considering

Figure 2: CaFA's full flow, including the *offline* phase—where CaFA mines feature-space constraints (§4.1)–and the *online* phase–where CaFA incorporates constraints when generating adversarial examples. In the online stage, we initially find adversarial perturbations that both comply with the structure constraints and mislead the model (§4.2). Then, we project the adversarial sample to comply with the mined relations constraints (e.g., DCs) (§4.3).



the worst-case adversary, are standard and common in the literature (e.g., [4], [10], [45]).

The adversary's primary goal, given x and y, is to craft a realistic feature-space instance misleading the ML model. Such instance can be found through a feature-space attack, with the following objective:

$$\begin{aligned} &\text{Find} \quad \delta \in \mathbb{R}^d \\ &\text{s.t.} \quad M(x+\delta) \neq y,\ (x+\delta) \vDash T, \\ &\quad Cost(\delta) \leq B \end{aligned}$$

Namely, the adversary aspires to find a feature-space perturbation (δ) that modifies the targeted sample (x) to be *misclassified* by the model; the modified sample must be *feasible* under the feature-space constraints (T), lending the sample to better problem-space realizability; and the modification of the original problem-space sample should involve *minimal costs* (i.e., adversarial effort), which is embodied by a computable $Cost$ function and a budget bound (B) in the feature space.

4. Technical Approach

To address the challenges set forth in the previous sections, we propose CaFA, a *feature-space attack* targeting NN models, which crafts *cost*-aware adversarial examples that are feasible relative to *feature-space constraints*. As depicted in Fig. 2, we divide our method to three stages, similarly to Sheatsley et al. [43]: (1) mining the feature constraints and defining their utilization (§4.1); (2) perturbing the given sample to fool the classifier model while maintaining *structure constraints* and low cost with TabPGD-CWL0 (§4.2); and (3) projecting the perturbed sample onto the learned constrained space using the Z3 solver [18] (§4.3).

4.1. Mining Constraints

As previously mentioned, we seek constraints of two types—*structure constraints* and *relation constraints* in the form of DCs. While the former are usually readily available (as defined during feature engineering), those of the latter type are typically not given explicitly and require extraction from the data using specialized mining frameworks.

To mine DCs, we use FastADC [51], a state-of-the-art technique for discovering *soft* DCs. We opt for soft constraints as hard constraints can limit and potentially exclude valuable and insightful constraints due to anomalous samples (e.g., a key constraint or a trend valid for ≥99% of samples might be omitted because of a malformed sample or rare feature value). We mine the constraints from the training set, tolerating up to a predefined violation rate among the dataset's sample pairs.

When using DCs in the attack, we seek to perturb samples while complying with DCs. Recall that a DC accounts for a *pair* of samples—*other* and *main* (§2.3). Accordingly, to test the compliance of a sample with a DC, we are required to assign the perturbed sample as the *main* sample and assign each of the training samples $\in X_{train}$ as the *other* sample, resulting in $|X_{train}|$ *practical* constraints per DC. Thus, for a large set of DCs, D, the number of *practical* constraints becomes excessively large ($|D| \cdot |X_{train}|$). Such a large number of constraints may harm run-time performance, thus necessitating filtering the DC sets. This filtration also allows us to control the constraints' soundness-completeness trade-off (see §5.1 and §6.1). Namely, removing low-quality DCs covering a few samples can increase completeness while incurring limited decrease in soundness.

To select well-performing DCs during filtration, we form a ranking-scheme based on well-established metrics [14], [51] quantifying succinctness (favoring concise constraints), coverage (favoring constraints more supported by data), and certain forms of constraint violations (see App. A.2). Subsequently, we pick the top-ranked constraints and closely related tuples. More specifically, we pick the highest-ranked DCs according to the metrics (amount denoted by n_{dcs}), and, for each DC, we pick the *other* tuples that provided the best compliance with the DC (an amount denoted by n_{tuples}, uniform over all DCs). This process allows evaluating multiple DC sets and choosing a well-sized, high-quality set of constraints, which we denote as T.

4.2. Perturbing Samples

Now we present TabPGD and TabPGD+CWL0, novel evasion attacks for crafting adversarial samples complying with *structure constraints*, and minimizing adversaries' effort. We mathematically define the cost function (§4.2.1), refer to the employed structure constraints (§4.2.2), and, finally, fully describe the TabPGD attack (§4.2.3) and its

CWL0 extension (§4.2.4). The samples crafted with this attack are later projected onto relation constraints (§4.3).

4.2.1. Heterogeneous Cost. As a proxy to the adversarial effort, our attack aspires to bound and minimize the adversarial perturbation with a ℓ_∞-norm variant (*standardized-*ℓ_∞) and the ℓ_0-norm, respectively. Specifically, in our framework, we bound the first (TabPGD) then minimize the second (CWL0). In doing so, we seek to make attacks more inconspicuous and, by proxy, limit their attacks' financial cost. Incidentally, as a byproduct, we empirically find that this approach also increases compliance with relation constraints, possibly due to limiting deviation from original samples that already satisfy these constraints (§6.2), and helps preserve functionality in the problem space (§6.6).

standardized-*ℓ_∞*-norm ensures small-magnitude changes in ordered features (i.e., continuous and ordinal). To account for the heterogeneity of tabular data, we perform a min-max scaling before calculating the ℓ_∞-norm. Said differently, given a range size of each feature, R_i, derived from the i^{th} feature's support, we standardize the features before applying the ℓ_∞-norm. Formally, *standardized-*$\ell_\infty(\delta) = \max_i \left|\frac{\delta_i}{R_i}\right|$, where i iterates over the ordered features. We define the *standardized-*ℓ_∞ ϵ-ball accordingly: $\forall i : [x_i - \epsilon \cdot R_i, x_i + \epsilon \cdot R_i]$, and call the R_is as the standardization factors.

ℓ_0-norm minimization limits the overall number of features that are altered.

Since each tabular feature mostly refers to a different subdomain, one can interpret the first cost as minimizing the *extent* of required effort within each subdomain, and the second cost as minimizing the *variety* of efforts required. Both goals are essential to ensure minimal effort when transforming feature-space samples to the problem space.

4.2.2. Structure Constraints. Our attack incorporates the targeted dataset's structure constraints (§2.3), utilizing each feature's *type* and *set of permissible values*. Furthermore, we account for the *encoding methods* of categorical features. Chiefly, for MLPs (§5.3), our attack seeks to preserve one-hot encodings' syntax, and, for TabNets, it handles multi-dimensional continuous embeddings of discrete features (§6.2.2). For both encoding methods, our attack follows the same principled approach (§4.2.3).

NNs often utilize one-hot encodings to process categorical features [39], thus capturing features' categories without imposing arbitrary order on feature values. Specifically, given a categorical feature f with S categories, its one-hot encoding is a binary vector $OneHot(f) \in \{0,1\}^{|S|}$, where the j^{th} entry is 1 if $j = f$ and 0 otherwise. Consequently, preserving valid encoding introduces an additional *structure constraint* on the feature vector: (1) each encoded coordinate is a single binary feature; and (2) exactly one coordinate in $OneHot(f)$ must be 1.

4.2.3. TabPGD's Algorithm. TabPGD tailors the PGD framework to tabular data. PGD, originally developed for the vision domain, ignores structure constraints and uses a fixed step size for all features. To overcome limitations, TabPGD involves *heterogeneous* update steps and a cost bound (*standardized-*ℓ_∞), and incorporates structure constraints throughout the perturbation. A concise pseudocode is provided in Alg. 1 and explained in what follows.

Algorithm 1 TabPGD

Require: x (sample), y (label),
ϵ (*standardized-*ℓ_∞ bound),
α (step-size factor), R (standardization factor),
n (iterations), *structureConstraints*
Ensure: x' is adversarial example, or $\perp$ if the attack fails.
1: $x' \leftarrow$ randomInitUnderStructureConstraint(x)
2: $g^{(accum)} \leftarrow 0$
3: **for** $\_ \in \{0, 1, 2, \dots, n\}$ **do**
4: $g \leftarrow \nabla_{x'}$CrossEntropy($M(x'), y$)
5: $x'_{temp} \leftarrow x' + \alpha \cdot (R \odot sign(g))$
6: $g^{(accum)} \leftarrow g^{(accum)} + g_{[\text{categorical coordinates}]}$
7: $x' \leftarrow$ perturbContinuousFeatures(x'_{temp}, x')
8: $x' \leftarrow$ perturbIntegerFeatures($\lceil x'_{temp} \rceil, x'$)
9: $x' \leftarrow$ perturbCategoricalFeatures($g^{(accum)}, x'$)
10: $x' \leftarrow$ clipToRange(x')
11: $x' \leftarrow$ clipTo ϵ Cost(x')
12: **if** $M(x') \neq y$ **then**
13: **return** x'
14: **end if**
15: **end for**
16: **return** $\perp$

TabPGD starts (Line 1) by randomly initializing the adversarial sample (x') within the *standardized-*ℓ_∞ ϵ-ball around the original sample (x). It then performs n iterations, each updating x' using gradients. Similar to PGD, we calculate loss gradients w.r.t to the perturbed sample (g, Line 4), then, we set the temporary perturbation (x'_{temp}, Line 5) using the gradient's sign, a predefined step size (α), and the standardization factor (R) (to account for heterogeneity, unlike PGD). Since the temporal perturbation (x'_{temp}) may violate structure constraints (specified by *StructureConstraints*), we update the perturbations to comply with these constraints: *continuous features* are left unchanged (Line 7); *ordinal, integer features* are rounded to the closest integer (Line 8); and *categorical features* (Line 9) are modified based on the accumulated gradient vector ($g^{(accum)}$) of their encoding (e.g., one-hot encoding), selecting the category encoded by the largest accumulated gradient (across iterations) while preserving valid encoding. To further enforce *structure constraints* and the *cost bound* we keep the perturbed features within their allowed ranges (Line 10) and within the *standardized-*ℓ_∞ ϵ-ball (Line 11). The algorithm terminates once the perturbed sample fools the model (Lines 12-14) or after n iterations without success (Line 16).

4.2.4. CWL0 Variant. While TabPGD aims to bound the *standardized-*ℓ_∞ cost of ordered features, it may still alter many features, thus increasing adversaries' effort. To address this, CWL0 augments attacks to account for the overall ℓ_0 cost. This extension adapts Carlini and Wagner (CW)'s ℓ_0-norm attack [11]—shown effective in reducing the ℓ_0-

norm of perturbations in gradient-based attacks—to work in tandem with TabPGD.

CWL0 operates by running *TabPGD* iteratively as a black-box. Each TabPGD run generates a perturbation, δ, and its corresponding loss gradient w.r.t to the perturbed sample, g. We employ a modified version of CW's heuristic, assigning each feature i a score

$$p_i(\delta, g, R) = \frac{g_i \cdot \delta_i}{R_i}$$

to rank its importance for the attack; intuitively, the higher p_i the more essential the feature for the attack. Note that, if the i^{th} feature is a coordinate in a one-hot encoding, we sum the importance over all of the encoding's coordinates, to form a unified score for the categorical feature. After each iteration, the feature with the lowest importance, $\text{argmin}_i\{p_i\}$, is frozen, meaning it remains unperturbed in subsequent iterations. The algorithm iterates, freezing one feature at a time, until either it fails to make further improvements or reaches a predefined maximum number of iterations.

4.3. Projecting on Relation Constraints

Equipped with mined constraints (T, §4.1), CaFA eventually projects the preliminary adversarial sample, which, at this point, complies with simple *structure constraints*, onto the constrained space imposed by the more complex mined *relation constraints* (e.g., DCs). We employ SAT Solvers to guarantee the obtained samples satisfy the constraints after projection, in a manner similar to prior work [43], in contrast to approaches treating the constraints as approximated objectives [46]. We implement the projection as follows: CaFA generates a first-order logic formula, Φ_T, that captures T—the set of mined constraints—and uses a SAT solver to ensure compliance. Furthermore, we introduce ϕ_T with additional assertions to verify the perturbations comply with structure constraints and adhere to attack-defined cost metrics (by bounding the allowed projection).

When a perturbed sample, x', violates Φ_T, we attempt to satisfy it with partial assignment by relaxing a portion of the literals (previously termed the *projection budget* [43]). Utilizing Sheatsley et al.'s heuristic [43], we prioritize relaxing the features that are least constrained. Namely, we relax the features satisfying the fewest assertions independently (intuitively, avoiding the most constrained features would increase the odds of a successful projection). We further optimize this by employing binary search on the required budget, aiming to minimize the number of relaxed features while ensuring a successful projection. We empirically found both methods (ranking and binary search) helpful for keeping the number of projected features minimal, thus helping to preserve attack success after projection.

Having decided which literals to free, our problem is reduced to solving partially-assigned Φ_T, with the partial assignments given by the non-relaxed features of x'. To do so, we employ the *Z3* Solver [18], obtaining the projected, perturbed sample satisfying both structural and relation constraints.

5. Experimental Setup

Next, we elaborate on the evaluation setup. We define the metrics we used (§5.1), describe the datasets (§5.2) and NN models (§5.3), specify the constraints employed (§5.4), and list the evaluated attacks (§5.5).

5.1. Metrics

We used various metrics to evaluate constraints' quality and attack efficacy.

5.1.1. Constraint Metrics. To evaluate constraints, we used and extended well-established metrics previously used in the space [43].

Compliance checks if a sample x satisfies all constraints in T:

$$\forall t \in T : x \vDash t \quad \text{, or simply:} \quad x \vDash T.$$

If true, we say x complies with T.

Completeness checks if *all* feature-space samples $\in X$ comply with T:

$$\forall x \in \mathbb{R}^d : \quad x \in X \implies x \vDash T.$$

If this holds, we say T is complete relative to the feature space X.

Soundness checks if all instances (real vectors) complying with T pertain to the genuine feature space (i.e., $\in X$):

$$\forall x \in \mathbb{R}^d : \quad x \vDash T \implies x \in X.$$

If they do, we say T is sound relative to X.

Since the mathematical definitions of soundness and completeness are not feasible to compute (they require enumerating all samples in an unknown feature space), we define empirical counterparts, lending themselves to being computed on given a test set $X_{test} \subset X$ and a set of constraints to evaluate T.

Empirical Completeness computes the proportion of X_{test} samples complying with T.

$$\widehat{Completeness}_T(X_{test}) = \frac{\left|\{x \in X_{test} \mid x \vDash T\}\right|}{\left|X_{test}\right|}$$

Empirical Soundness, a metric we introduce, leverages manually constructed, ground-truth constraints (called golden constraints; denoted as $\mathring{T}$) [29] to test whether the mined constraint set can detect violations of data-integrity constraints known to hold. For each sample $x \in X_{test}$ that complies with our learned constraints T, we generate a *modified* sample $\mathring{x}$ by intentionally violating a golden constraint $\mathring{t} \in \mathring{T}$. The violating *modification* is done by taking the negation of the constraint and uniformly sampling a random feature value that necessarily satisfies this negation. This creates out-of-domain samples that *should* also violate T if it is sound.

Hence, empirical soundness quantifies the proportion of modified samples ($\mathring{x}$) that, while complying with T *prior* to the modification, violate T *after* the modification. We apply

this prior restriction to samples that originally comply with T, since we aspire to attribute the violations exclusively to violating golden constraints. We consider each golden constraint separately in the metric, as their mining complexity may vary:

$$\widehat{Soundness}_T(X_{test}, \mathring{T}) = \frac{\left\{x \in X_{test}, \mathring{t} \in \mathring{T} \mid x \vDash T \wedge \mathring{x} \nvDash T\right\}}{\left|\mathring{T}\right| \cdot \left|\left\{x \in X_{test} \mid x \vDash T\right\}\right|}$$

F1 derives a composite score from (empirical) soundness and completeness. Attaining perfect completeness (by accepting all samples) or soundness (by rejecting all constraints) alone is trivial, hence there is a need to balance both. One can view completeness and soundness as analogous to recall and precision [17], respectively. Thus, following this analogy, we adopt the F1 score—the harmonic mean of completeness and soundness—as a quality measure of the constraints set, seeking to maximize it. Formally:

$$F1 := \frac{2 \cdot completeness \cdot soundness}{completeness + soundness}.$$

5.1.2. Attack Metrics. We use a battery of metrics to estimate attacks' efficacy.

Attack Success-Rate measures the proportion of misclassified adversarial samples:

$$\frac{|\{M(x+\delta_x) \neq y \mid x \in X_{test}\}|}{|X_{test}|}$$

where δ_x is the adversarial perturbation the attack yields for x. This is a widely accepted metric for evaluating attacks (e.g., [9], [41]).

Feasible Attack Success-Rate computes the proportion of misclassified samples that also comply with T.

$$\frac{|\{M(x+\delta_x) \neq y \ \wedge \ (x+\delta_x) \vDash T \mid x \in X_{test}\}|}{|X_{test}|}$$

Related work also used this metric (e.g., [43], [46]).

Cost is measured in ℓ_0 and *standardized-*ℓ_∞ (§4.2).

5.2. Datasets

We used three commonly used [7], [19], [43] tabular datasets. All datasets were randomly split into training and testing sets (87% and 13% ratio). We summarize the features of each dataset in Tab. 2.

Phishing. The *phishing* dataset [13], for detecting phishing websites, comprises 10K samples: 5K phishing and 5K legitimate websites. It includes features derived from webpage URLs and HTML source code. We focus on 10 out of 48 features that achieve maximal model accuracy, as per work proposing the set [13] and Sheatsley et al. [43].

Bank Marketing. This *bank* dataset [33], for predicting whether clients would purchase bank services, has 45K samples, with 5.2K positive labels indicating service purchase. We selected 11 out of 16 features leading to the highest validation accuracy, as per Borisov et al.'s survey [7].

Dataset	# Samples	# Features by type			
		Categorical	Continuous	Ordinal	All
Phishing	10K	5	2	3	10
Bank	45K	4	2	5	11
Adult	32.5K	7	0	6	13

TABLE 2: Dataset sizes and feature counts by feature types.

Model \ Dataset	Adult	Bank	Phishing
MLP	86.1%	89.0%	94.7%
TabNet	87.0%	89.7%	95.9%

TABLE 3: The benign accuracy achieved over the test sets.

Adult. The *adult* dataset [26], for predicting whether people's income levels surpasses a certain threshold, contains 32.5K samples, with 8K classified as high-income. We used all 13 available features.

5.3. ML Models

We tested two NN architectures, training models for each dataset. First, we used MLPs with ReLU activations, as is standard [7], [43]. For these models, we preprocessed our data with standard techniques [7], applying one-hot encodings for categorical features and coordinate-wise normalization based on training-data statistics. The MLPs' hyperparameters were determined through a grid search, optimizing for benign accuracy (i.e., accuracy on clean, unperturbed data). The chosen models have five hidden layers, each with width of 128, and were trained using the Adam optimizer [24] with a learning rate of $5{\times}10^{-4}$. Besides targeting MLPs, we experimented with a transformer-based state-of-the-art architecture geared for tabular data, TabNet [2], choosing its hyperparameters via a grid search, and using the official implementation. Tab. 3 reports the models' benign accuracy on the three datasets. For all datasets, the models achieved accuracy comparable to or exceeding past benchmarks [7], [13], [33]. The TabNet models achieved slightly better, yet similar, benign accuracy to their MLP counterparts.

5.4. Constraints

We evaluated two types of relation constraints and used them to measure feasibility. Particularly, we used DCs to assess feasibility, except for §6.5, where we used Valiant's.

Valiant's Constraints. We used a variant of Valiant's algorithm, proposed by Sheatsley et al. [43], to mine Valiant's Constraints in CNF form. Motivated to model the most general constraint theory, Sheatsley et al. proved that their parameterization mines the least constrained constraint theory. We note that any other parameterization of Valiant's is practically intractable to run, due to the exponential complexity of its mining algorithm (App. A.1).

Denial Constraints (DCs). We mine DCs with the state-of-the-art FastADC algorithm [51], allowing violation rates

of up to 1% for each soft constraint (i.e., no more than 1% of training samples can violate each constraint), similarly to Livshits at al. [29]. (We clarify that this is different from the completeness metric from §5.1, estimating the rate at which *test samples* violate *all* constraints.) For performance reasons imposed by FastADC, we restrict the set of possible DC predicates to only compare between the same features. Note that cross-feature comparisons are mostly irrelevant due to semantic differences between features in tabular data. Yet, relations between features can still be learned, as the constraint itself considers multiple features (each in its own predicate). Additionally, we identified the predicate space as a significant bottleneck in FastADC's runtime, stemming from an exponential dependency of the algorithm in the predicate space's size. Thus, formally, we used the predicate space:

$$\Psi := \left\{ \left(x_i \diamond x_i' \right) \mid \diamond \in \{=, \neq, <, >, \leq, \geq\}, i \in [d] \right\}.$$

This formalism can still capture expressive constraints of the form presented in Fig. 1 (and does so in practice). The specific set of DCs utilized in experiments was chosen after evaluating multiple sets (see §6.1).

5.5. Evaluated Attacks

Besides the attack variants we propose, we evaluated and compared multiple attacks whose hyperparameters were selected as to optimize the feasible attack success-rates. Prior attacks were chosen for close characteristic with CaFA [43], [46]; other attacks were either difficult to properly reproduce in the absence of an implementation, did not involve relation constraints, or were domain specific (see Tab. 1).

CaFA (§4) is our proposed framework composed of several components. We set **TabPGD**'s step size α to $\frac{\epsilon}{100}$, its *standardized-*ℓ_∞'s ϵ to $\frac{1}{30}$ (a choice which we further explore in §6.3) and the number of iteration to 100. TabPGD can be augmented with CWL0 and followed by projection with a SAT solver. We ran CWL0 for up to 30 iterations and used the Z3 solver [18]. Projections were done by freeing a subset of the features, ranging from 0% to 50% of features and determined by binary search.

PGD [31] is an iterative gradient-based white-box attack, limiting perturbations' ℓ_∞-norms (§2.1). We set the norm bound (ϵ) to 100 and the step-size (α) to 1, ran attacks 100 iterations with early stopping, and used the cross-entropy loss.

C-PGD [46] is a PGD variant transforming constraints into a penalty function that is added to the attack's loss function (§2.4). The attack optimizes three objectives—penalty function (constraints), misclassification, and distance. We adopted the same parameters as in PGD, assigning 0.1 as the weight for the additional penalty, and used the official implementation.

MoEvA2 [46] is a query-based attack using models' output probabilities and a multi-objective genetic algorithm (R-NSGA-III [50]) to optimize an analogous loss to C-PGD's. We used the official implementation and adopted the variant bounding the ℓ_2-norm due to the higher feasible attack success rates attained compared to the ℓ_∞-norm variant (see App. B). The remaining parameters were adopted from the original work and implementation.

Sheatsley et al.'s Attacks [43] include (1) a PGD variant with structural constraints, and (2) a Constrained-Saliency Projection (CSP) attack, an ℓ_0-norm-based attack [36], both integrating Valiant's constraints. We compared these attacks to CaFA in a setting identical to the one considered in the original work (i.e., under Valiant's constraint space).

6. Experimental Results

Now we turn to our results. We start with evaluating different constraint sets' quality (§6.1) before assessing attacks' effectiveness in terms of feasible success rates under DCs (§6.2). Then, we test attacks on two other dimensions—attack cost (§6.3) and run time (§6.4). Subsequently, we measure attacks' feasible success rates with Valiant's constraints (§6.5). We close the section with a case study of real-world phishing websites exploring the extent to which CaFA and other attacks can produce adversarial samples implementable in the problem space (§6.6).

6.1. Constraints' Quality

Experiment Description. We empirically evaluated the completeness and soundness (§5.1) of relation-constraint sets of different types to identify high-quality ones. Specifically, we evaluated constraints mined with Valiant's algorithm [43], [49] and DCs mined with FastADC [51]. Based on this, we found constraints that best balance completeness and soundness, maximizing the $F1$ score (§5.1). Here, we report on the evaluation using the test set of the *bank* dataset. We further validate the findings on the *phishing* dataset in App. C.2.

For this experiment, we used a set of 1.5K samples data points sampled from the *bank* dataset's test set, where we mined the constraints from the disjoint training set. For measuring soundness, we used the four golden constraints identified by Deutch and Frost for the the *bank* dataset [19]; notably, such constraints are not readily available for other datasets, where a manual effort was required to extract them (see App. C.2). We list these golden constraints in Tab. 4 (e.g., the first constraint requires that for a client, x, who was not contacted in the previous marketing campaign (i.e., $previous$=0), the outcome of the previous campaign is of category *unknown* (i.e., $poutcome$=-1)). We used these constraints to generate violating samples (§5.1) that should be rejected by the mined constraints.

We evaluate multiple sets of DCs, each of different size, to attain different soundness-completeness tradeoffs: increasing the number of constraints should increase soundness, whereas decreasing it should boost completeness. We formed the subsets by picking the highest ranked constraints, per the ranking scheme previously presented (§4.1), limiting both the required amount of constraints (n_{dcs}) and the *other* tuples (n_{tuples}).

Constraint	Comp. Rate (%)
$x.previous = 0 \implies x.poutcome = -1$	100.0%
$x.previous = 0 \impliedby x.poutcome = -1$	100.0%
$x.job =$ 'student' $\implies x.marital =$ 'single' $\wedge\ x.age \leq 35$	99.8%
$x.job =$ 'admin' $\implies x.education =$ 'secondary'	100.0%

TABLE 4: Golden constraints derived by Deutch and Frost [19] from the *bank* dataset. We describe each constraint relative to a feature vector $x \in X$ and report the precentage of samples complying with it.

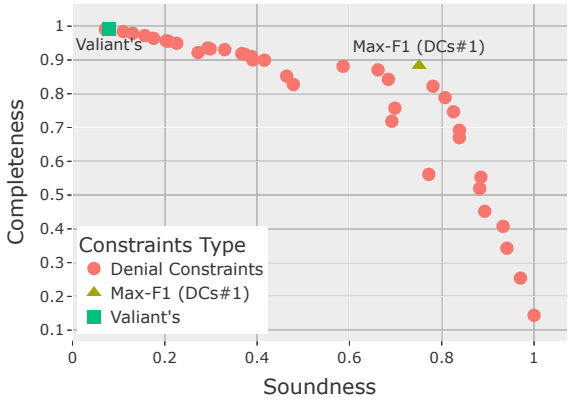


Figure 3: Evaluation of constraints' soundness and completeness over the test set. We distinguish Valiant's constraints set from the rest of the DCs, mined by FastADC [51]. We also mark DCs set with the highest F1 score that we use in most experiments.

Type	n_{dcs}	n_{tuples}	# Constr.	Compl.	Sound.	F1	Notes
Valiant's	-	-	51K	**99.1%**	7.9%	14.7%	
DCs	5000	1	5K	88.3%	75.1%	**81.2%**	Max. F1, DCs #1
DCs	7000	500	3.5M	14.4%	**100.0%**	25.2%	DCs #2
DCs	100	1	0.1K	**99.1%**	7.1%	13.3%	DCs #3

TABLE 5: Constraint sets of interest and their empirical soundness, completeness, and F1 scores. DCs differ in their configuration (i.e., n_{dcs} and n_{tuples}).

Experiment Results. The empirical completeness and soundness for different constraint sets are depicted in Fig. 3. Tab. 5 presents constraint sets of interest. From our evaluation, we conclude that Valiant's constraints serve as a strong baseline in terms of completeness—consistent with the inherent properties of their mining process (§5.4) [43]. To achieve this performance, Valiant's algorithm crafts numerous constraints (51K; see Tab. 5). In contrast, DCs manage to achieve comparable soundness and completeness measures with significantly fewer constraints (0.1K; DCs #3 in Tab. 5). Furthermore, Valiant's algorithm yields low *soundness*. This can be attributed to the algorithm configuration, which favors completeness over soundness. We reiterate that other parameterizations are infeasible to run (§5.4), preventing us from attaining other soundness-completeness trade-offs with Valiant's constraints. Note that, since soundness is monotone in the constraint sets (i.e., adding more constraints to a set implies, by definition, larger soundness), inspecting subsets of Valiant's full constraint set (as done for DCs) could only harm soundness.

The expected completeness-soundness trade-off is visible in the analysis of DCs—larger sets lead to higher soundness and lower completeness (see *DCs #2* and *DCs #3* in Tab. 5). To balance the two metrics, picking performant DCs for the remaining experiments, we chose the constraint set achieving highest F1 measure (Fig. 3), resulting in a set containing 5K constraints (*DCs #1*, Tab. 5). Applying the same evaluation on the *phishing* dataset, we reached similar conclusion (App. C.2), leading us to use the 5K constraints as a default configuration for DCs. The chosen DCs sets of *phishing*, *adult*, *bank* datasets achieved 95.4%, 84.7%, and 88.3% empirical completeness, respectively, on the test data.

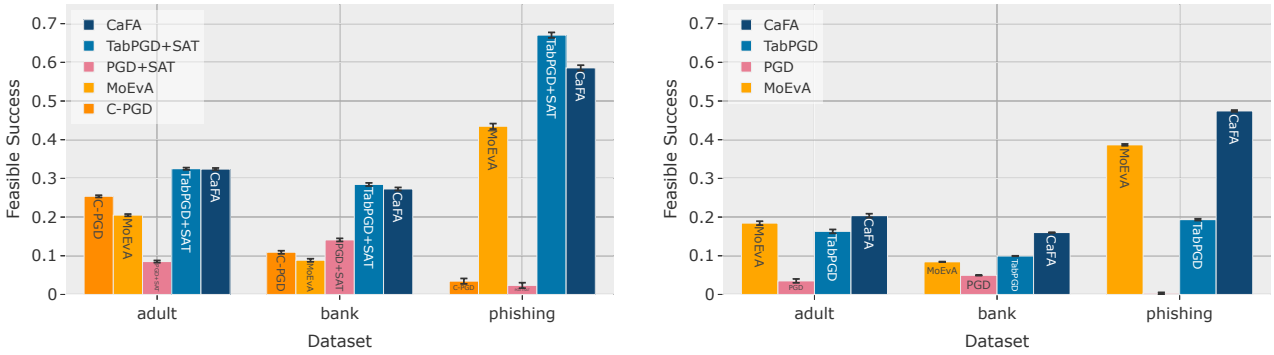
6.2. Feasible Attack Success

We evaluate attacks' success, in terms of misleading the targeted models (MLPs and TabNets) and feasibility of the adversarial samples relative to the chosen set of DCs. In addition to evaluating *full* attacks, when DCs were integrated, we also examined variations where we prevented access to DCs by the attacks (e.g., by disabling projection after TabPGD in CaFA, or not including constraints in C-PGD's and MoEvA2's losses). Doing so helped us gain further insights about how attacks work and allowed us to estimate success when adversaries have no access to auxiliary data for mining constraints.

6.2.1. Attacking MLPs. We performed three experimental runs, each with a different random seed and different 1K test samples. We report the average results in Fig. 4 (more details are included in App. C.4).

As evaluations of the *full* attacks show (Fig. 4a), both variants of our CaFA (TabPGD and TabPGD+CWL0) attained higher feasible success rates than other attacks on all datasets. We note a wider performance gap (Fig. 4a, between CaFA and other attacks) compared to the gap shown when disabling DC access (Fig. 4b), indicating CaFA benefits more from DC integrating. We also noticed varying performance across datasets, which could be attributed to their distinct characteristics (e.g., amount of categorical features). Nonetheless, CaFA consistently found feasible adversarial samples in challenging datasets like the *bank* dataset, where other attacks showed limited success.

As anticipated, attacks not integrating DCs were more likely to produce out-of-domain adversarial samples, as indicated by the low feasible success rates (Fig. 4b). Still, across all datasets, CaFA consistently achieved the highest feasible success, with MoEvA2 providing slightly a lower rate. This validates the effectiveness of approaches incorporating structure constraints, and more critically, providing perturbations with low ℓ_0-norms. In particular, these findings show that, in the absence of complex constraints, minimizing perturbations' ℓ_0-norms can improve the likelihood of producing



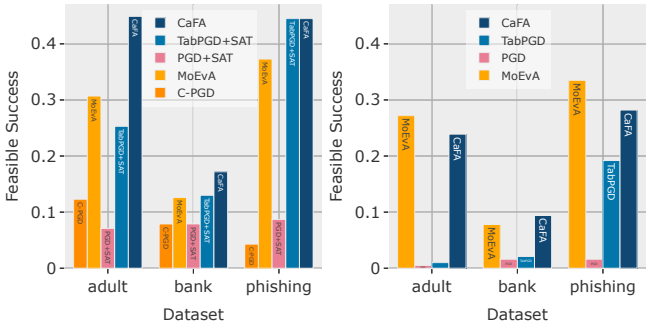
(a) **Full attacks** (b) **Ablation:** Disabling DCs access

Figure 4: Comparison of the proportion of feasible and misclassified adversarial samples across attacks on MLPs.

feasible adversarial samples. This is presumably since altering fewer features reduces the interference with feature dependencies.

6.2.2. Attacking TabNets. To further validate our findings, we tested feasible success rates on state-of-the-art transformer-based TabNet models. A slight adjustment to TabPGD was required to attack the transformer-based models. Since TabNet uses a continuous embedding layer to encode discrete categorical features, a special treatment was required for their perturbation (§4.2). Particularly, inspired by the attack on binary malware by Kreuk et al. [27], we performed the following in each gradient-update step of the attack: we calculated the continuous perturbation on the embedding layer (as if these are continuous features we attack) using step size maximizing attack success per line search, and picked the discrete categories whose embeddings were closest (in Euclidean distance) to the perturbed embedding.

From the results (Fig. 5), we observe that CaFA's feasible success surpasses prior attacks, with similar trends to those presented when attacking MLPs. When disabling DCs access, we notice a decrease in the feasible success, with MoEvA2 outperforming CaFA on two datasets. These results emphasize that CaFA benefits from DC access more than other attacks do.



(a) Full attacks (b) Disabling DCs access

Figure 5: Comparison of the proportion of feasible and misclassified adversarial samples across attacks on TabNets.

6.3. Attack Cost

Next, we measured attacks' cost—a crucial aspect of attacks, besides their feasible success rates (§3). We first inspect the costs of the different attacks (*standardized-*ℓ_∞,ℓ_0) while considering success, then we specifically examine CaFA's efficiency in reducing perturbations' ℓ_0-norms.

Comparing Attacks. In this experiment, we ran attacks and measured their costs and feasible success rates, while averaging these over the datasets. We report the results in Fig. 6, and provide a more fine-grained analysis of the *standardized-*ℓ_∞- and ℓ_0 costs in App. C.1.

We find that our attacks provide the best trade-off between ℓ_0- and *standardized-*ℓ_∞-norms, while maintaining the highest feasible success. This is opposed to other attacks, which either incur high costs in at least one norm (MoEvA2, C-PGD), or fail to produce successful attacks (PGD). We contend that using a standardized distance metric is particularly useful for tabular data, where features' value ranges may vary, thereby requiring different levels of adversarial effort for an equivalent feature modifications.

CaFA's ℓ_0-norm. We measured the perturbation's ℓ_0-norm of the feasible and successful adversarial samples crafted by CaFA's variants. The results (Fig. 7) evidence CWL0's importance—it can reduce up to half of the number of perturbed features compared to TabPGD alone. Overall, CaFA requires modifying 15–30% of features, on average, for feasible and successful attacks.

6.4. Attack Run-Time

Run time is another factor that may impact CaFA's and other attacks' adoption. To this end, we evaluated and compared attacks' average run time for producing a single adversarial example. We executed attacks against the MLPs over 500 test samples, one sample at a time, and calculated the average run time. We then averaged the results across the three datasets.

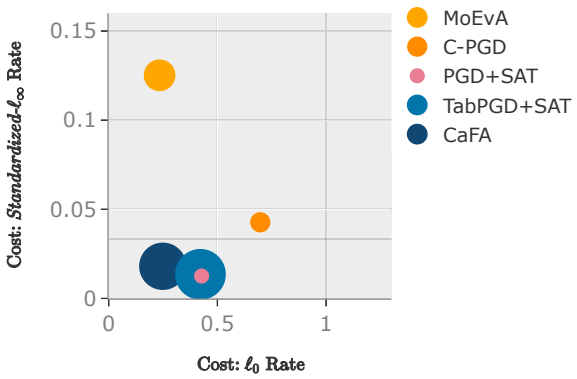


Figure 6: Comparing attacks' costs (ℓ_0 and *standardized-*ℓ_∞) while accounting for their feasible success (as indicated by the bubble size). Dashed line marks the ϵ parameter used by CaFA and TabPGD. Measures were averaged over all datasets (see App. C.1 for per-dataset results). Larger bubbles closer to the bottom-left corner are better.

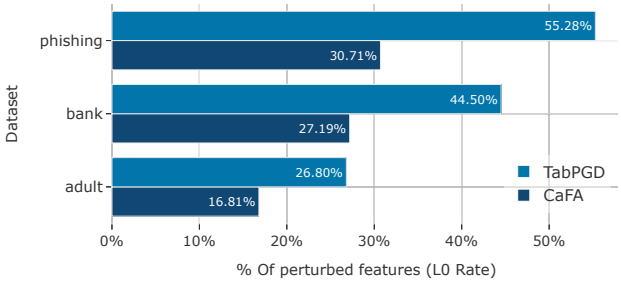


Figure 7: ℓ_0 measures of the perturbations generated by feasible and successful adversarial samples of CaFA's variations (lower is better). To enable comparison across datasets, the measures were normalized by the number of features.

As shown in Fig. 8, both CaFA variants are roughly as fast as PGD, making them the fastest-measured methods incorporating constraints. A notable bottleneck in CaFA is the projection using SAT solvers. However, we observe this bottleneck is alleviated when reducing the ℓ_0 cost, as seen with CWL0.

6.5. Feasible Success With Valiant's Constraints

Different constraint types can be integrated into CaFA. To test its flexibility, we explored incorporating Valiant's constraints, mining them as described by Sheatsley et al. [43] (see §A.1). We compared the findings to those attained by Sheatsley et al.'s [43] attacks (PGD and CSP) on the *phishing* dataset. Due to the lack of publicly available implementation, we also tested their PGD variants' performance with our attempted implementation.

Tab. 6 presents the results on the *phishing* dataset (see App. C.5 for results on other datasets). CaFA yields

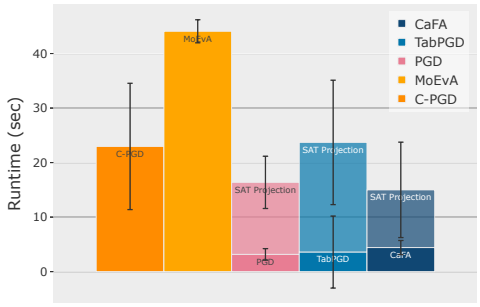


Figure 8: Average attack run-times per sample. For CaFA (TabPGD variants), we distinguish the perturbation stage from the projection.

Attack	Full Attacks		Disabling Access	
	Comp.	Comp. & Mis.	Comp.	Comp. & Mis.
CaFA (Valiant's)	100.0%	**87.6%**	77.7%	77.7%
PGD [31] (with tuning)	100.0%	67.1%	3.3%	3.3%
Sheatsley et al.'s PGD (∗) [43]	60.0%	60.0%	25.0%	-
Sheatsley et al.'s PGD [43]	89.4%	32.4%	2.8%	2.8%
Sheatsley et al.'s CSP (∗) [43]	100%	60.0%	80.0%	-

TABLE 6: Attacking phishing detection with Valiant's constraints. We report the percentage of adversarial examples complying with the constraints and ones both complying and misclassified (i.e., feasible success rate) on the full attacks and when disabling Valiant's access. (∗) denotes results reported in prior work [43] (otherwise, the results were produced by our experiments).

∼78% successful and feasible adversarial samples before projection, increasing to ∼88% post-projection—about ×1.5 more than the baseline attacks. Interestingly, Sheatsley et al. previously concluded that Valiant's constraints markedly harms attack success-rates [43]. Our findings draw a different picture, with attacks succeeding relatively often even when Valiant's constraints are accounted for. Results on the other datasets further corroborate this finding (App. C.5). We attribute CaFA's improved success rates mainly to how the attack handles heterogeneous features (e.g., step size) and to the minimization of the projected-features count, contributing to maintaining the samples misclassified.

6.6. Case Study: Evading Phishing Detection

We ran a case study to challenge CaFA with a real-world problem space. Namely, we employed CaFA to guide manual modification of *actual* phishing websites to evade detection by the phishing-detection MLP (§5.3). Through this study, we critically analyzed different attack variants from a problem-space perspective, highlighting the importance of CaFA's attack objectives (§3) in a real-world scenario. We chose to focus on a set of *phishing* samples, as this domain has clear and precise definition for *implementability* (i.e., producing a valid HTML), in addition to up-to-date and realistic samples available online.

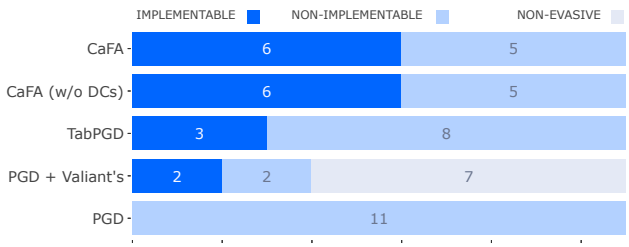


Figure 9: Results of manually examining 11 samples of phishing websites originally detected by our phishing-detection MLP. We attempted to implement (the HTML and URL) the adversarial sample created by each attack (or at different stages of CaFA) and annotated whether it was implementable or not. Some adversarial samples were also found to be non-evasive (i.e., they were correctly classified).

For the study, we collected 11 samples of phishing websites in the wild (i.e., HTML files), recently identified in the PhishTank archive [1] (sample IDs found in App. C.3). To perform attacks, we first extracted the website's features. Then, we employed CaFA framework to generate an adversarial feature-space instance. This instance subsequently served as a *recipe* for modifying the problem-space instances (original URL and HTML code) to fool the targeted ML detector. We repeated this process with different attacks, attempting to manually implement the problem-space instances for each, and annotating whether it was possible.

Using our attack, we successfully manipulated up to ~55% of the phishing websites to evade the model, while performing inconspicuous changes. Fig. 9 presents the result, and App. C.3 documents a problem-space sample we produced.

Attacks' Realizability. Tailoring the attack to the tabular domain markedly increased attacks' success (PGD vs. CaFA in Fig. 9) and led to better alignment with problem-space constraints compared to prior work (PGD+Valiant's vs. CaFA in Fig. 9). Enforcing structure constraints (TabPGD) has immediately made some attacks practical by eliminating critical feature inconsistencies, such as ensuring an *integer* URL in *reasonable* length. Additionally, minimizing the number of perturbed features (i.e., ℓ_0-norm minimized by TabPGD+CWL0 even without incorporating DCs in CaFA), originally motivated by cost, also resulted in implementable adversarial samples in the feature space. This result is consistent with those show in §6.2, where minimizing ℓ_0-norm already improved feasible success rates, even without integrating DCs.

Direction for Improving Realizability. The results show that, in the particular use-case examined, integrating the DCs mined with FastADC [51] did not provide an improvement in realizability compared to the attack variant merely minimizing the perturbations' ℓ_0-norms (CaFA vs. CaFA (w/o DCs) in Fig. 9). A closer inspection showed that the mined set of DCs lacked highly nuanced constraints—for instance, subtle dependencies between different proportions of hyperlinks, akin to the last golden constraint presented in Tab. 7 (App. C.2), were not mined—as opposed to the coarser dependencies the DCs aptly identified (e.g., numerical character count in a URL not exceeding its length). This is not surprising, due to imperfect completeness and soundness measures of DCs (App. C.2). Accordingly, projecting on the mined DCs still led to violations of constraints imposed by the problem space, thus harming realizability. While DCs posses of high expressivity, capable of capturing nuanced dependencies, a profound challenge lays in mining them to capture genuine constraints with high completeness and soundness. Thus, enhancements in mining DCs, leading to higher quality constraints in the future, would also boost CaFA's efficacy.

Cost Efficiency. Throughout the manual translation of perturbations into problem-space modifications, the significance of cost was apparent. The bounded *standardized*-ℓ_∞-norm in CaFA has contributed to limiting the effort required for certain modifications. For example, it meant that only minor character alterations were required for the URL. A drastic URL change would necessitate significant effort, such as potentially purchasing a new domain. Additionally, minimizing ℓ_0-norm helped in narrowing down the *variety* of efforts required from the attacker. For instance, modifying only *href* HTML tags was simpler than tampering with multiple, potentially interdependent, HTML logics. In this study we also observed that minimizing the adversarial effort has served the implementable adversarial samples in maintaining their functionality (i.e., phishing pages kept identical appearance when implemented after evasion).

7. Discussion

We now discuss potential defenses against CaFA and other future directions.

Defenses. Adversarial training, demonstrated effective under various threat models (e.g., [8], [31], [46]). It is perhaps the most natural defense against adversarial examples generated by CaFA. To conduct adversarial training, one would need to execute CaFA for each training batch to craft adversarial examples for training. Doing so would incur a significant training-time overhead (Fig. 8) potentially rendering adversarial training infeasible. Thus, speeding up attacks (e.g., by accelerating, or relaxing, the projection) could be critical for enabling adversarial training.

Yet another interesting direction to explore would be detecting adversarial examples crafted by CaFA. In particular, due to the nature of projections with SAT solvers, we conjecture that, compared to benign samples, CaFA's adversarial examples would violate constraints by changing more features by smaller amounts. Accordingly, if such differences exist, statistical techniques would allow telling adversarial and benign samples apart. These techniques could supplement pre-existing detectors (e.g., [40]), enhancing their detection accuracy.

Extending a concept proposed by Simonetto et al. [46], another approach is enriching the model's input with more

features computed as complicated functions of other features, with the goal of inducing a challenge for constraint mining. For example, one may add a feature via a complicated arithmetic function of other features that cannot be represented by DCs, despite their expressivity.

Finally, as ML models interoperate with other modules (e.g., expert-defined rule-based tools for phishing detection) when deployed in real-world systems, the complexities of such systems may pose natural challenges to adversaries. Therefore, an intriguing direction for future research would be to evaluate the robustness of complex real-world systems against CaFA and similar attacks.

Future Work. Besides enhancing models' robustness against CaFA and related attacks, it would be interesting to explore avenues to improve CaFA's performance and further generalize it. For instance, improved projection techniques can help improve CaFA's performance, possibly by relaxing the requirement to satisfy *all* the constraints to *most* of them, or explicitly accounting for cost and misclassification objectives. Additionally, to make it more widely applicable, it would also be helpful to extend CaFA to non-NN-based ML models.

8. Conclusion

We presented CaFA, a system producing adversarial examples to mislead ML models classifying tabular data. CaFA tackles the two primary challenges of realizability—i.e., adversarial perturbations in the feature space may not be implementable in the problem space—and cost minimization—i.e., ensuring adversaries' effort is minimized when crafting adversarial artifacts. To tackle the former, CaFA leverages structure constraints and automatically mined denial constraints, ensuring that adversarial perturbations comply with domain-imposed restrictions. For the latter, CaFA seeks to minimize the number of features perturbed as well as the extent to which they are altered while accounting for the heterogeneity of features. The empirical evaluation with three commonly used datasets and two standard models demonstrate, among others, the *(1)* advantages of the constraints used by CaFA (specifically, better balancing soundness and completeness than previously used constraints); *(2)* CaFA superiority at generating feasible adversarial examples that are misclassified while satisfying integrity constraints compared to prior attacks; and *(3)* CaFA ability to minimize the number of feature's perturbed and the perturbations' magnitude. We open-source CaFA's implementation, hopefully it would be used as a generic mean for evaluating tabular classifiers' robustness against practical attacks prior to deployment.

Acknowledgements

This work has been partially funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 804302) and has been supported in part by a grant from the Blavatnik Interdisciplinary Cyber Research Center (ICRC); by Intel® via a Rising Star Faculty Award; by a gift from KDDI Research; by Len Blavatnik and the Blavatnik Family foundation; by a Maof prize for outstanding young scientists; by the Ministry of Innovation, Science & Technology, Israel (grant number 0603870071); by a gift from the Neubauer Family foundation; by NVIDIA via a hardware grant; and by a grant from the Tel Aviv University Center for AI and Data Science (TAD).

References

[1] Phishtank. https://www.phishtank.com/. Accessed on 2024-04-18.

[2] Sercan Ö. Arik and Tomas Pfister. TabNet: Attentive interpretable tabular learning. In *AAAI*, 2021.

[3] Vincent Ballet, Xavier Renard, Jonathan Aigrain, Thibault Laugel, Pascal Frossard, and Marcin Detyniecki. Imperceptible Adversarial Attacks on Tabular Data. In *NeurIPSW*, 2019.

[4] Shumeet Baluja and Ian Fischer. Learning to attack: Adversarial transformation networks. In *AAAI*, 2018.

[5] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Srndic, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *ECML/PKDD*, 2013.

[6] Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. In *CCS*, 2018.

[7] Vadim Borisov, Tobias Leemann, Kathrin Sessler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. Deep neural networks and tabular data: A survey. *IEEE Trans Neural Netw Learn Syst*, 2022.

[8] Hamid Bostani, Zhengyu Zhao, Zhuoran Liu, and Veelasha Moonsamy. Domain constraints in feature space: Strengthening robustness of android malware detection against realizable adversarial examples. *arXiv*, 2022.

[9] Nicholas Carlini, Anish Athalye, Nicolas Papernot, Wieland Brendel, Jonas Rauber, Dimitris Tsipras, Ian J. Goodfellow, Aleksander Madry, and Alexey Kurakin. On evaluating adversarial robustness. *arXiv*, 2019.

[10] Nicholas Carlini, Steve Chien, Milad Nasr, Shuang Song, Andreas Terzis, and Florian Tramer. Membership inference attacks from first principles. In *S&P*, 2022.

[11] Nicholas Carlini and David A. Wagner. Towards evaluating the robustness of neural networks. In *S&P*, 2016.

[12] Francesco Cartella, Orlando Anunciação, Yuki Funabiki, Daisuke Yamaguchi, Toru Akishita, and Olivier Elshocht. Adversarial attacks for tabular data: Application to fraud detection and imbalanced data. In *SafeAI*, 2021.

[13] Kang Leng Chiew, Choon Lin Tan, KokSheik Wong, Kelvin S.C. Yong, and Wei King Tiong. A new hybrid ensemble feature selection framework for machine learning-based phishing detection system. *Inf. Sci.*, 2019.

[14] Xu Chu, Ihab F. Ilyas, and Paolo Papotti. Discovering denial constraints. In *VLDB*, 2013.

[15] Amit Cohen and Mahmood Sharif. Accessorize in the dark: A security analysis of near-infrared face recognition. In *ESORICS*, 2023.

[16] Francesco Croce and Matthias Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In *ICML*, 2020.

[17] Jesse Davis and Mark Goadrich. The relationship between precision-recall and ROC curves. In *ICML*, 2006.

[18] Leonardo Mendonça de Moura and Nikolaj S. Bjørner. Z3: an efficient SMT solver. In *TACAS*, 2008.

[19] Daniel Deutch and Nave Frost. Constraints-based explanations of classifications. In *ICDE*, 2019.

[20] Ivan Evtimov, Kevin Eykholt, Earlence Fernandes, Tadayoshi Kohno, Bo Li, Atul Prakash, Amir Rahmati, and Dawn Song. Robust physical-world attacks on machine learning models. In *CVPR*, 2018.

[21] Kevin Eykholt, Taesung Lee, Douglas Schales, Jiyong Jang, and Ian Molloy. URET: Universal robustness evaluation toolkit (for evasion). In *USENIX Security*, 2023.

[22] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. Adversarial examples for malware detection. In *ESORICS*, 2017.

[23] Cormac Herley. Why do Nigerian scammers say they are from Nigeria? In *WEIS*, 2012.

[24] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[25] Klim Kireev, Bogdan Kulynych, and Carmela Troncoso. Adversarial robustness for tabular data through cost and utility awareness. In *NDSS*, 2023.

[26] Ron Kohavi. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In *KDD*, 1996.

[27] Felix Kreuk, Assi Barak, Shir Aviv-Reuven, Moran Baruch, Benny Pinkas, and Joseph Keshet. Deceiving end-to-end deep learning malware detectors using adversarial examples. In *NeurIPSW*, 2018.

[28] Bogdan Kulynych, Jamie Hayes, Nikita Samarin, and Carmela Troncoso. Evading classifiers in discrete domains with provable optimality guarantees. *arXiv*, 2018.

[29] Ester Livshits, Alireza Heidari, Ihab Ilyas, and Benny Kimelfeld. Approximate denial constraints. In *VLDB*, 2020.

[30] Keane Lucas, Sharif, Mahmood, Bauer, Lujo, Michael K. Reiter, and Saurabh Shintre. Malware makeover: Breaking ML-based static analysis by modifying executable bytes. In *AsiaCCS*, 2021.

[31] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *ICLR*, 2018.

[32] Yael Mathov, Eden Levy, Ziv Katzir, Asaf Shabtai, and Yuval Elovici. Not all datasets are born equal: On heterogeneous tabular data and adversarial examples. *Knowl Based Syst.*, 2022.

[33] Sérgio Moro, Paulo Cortez, and Paulo Rita. A data-driven approach to predict the success of bank telemarketing. *Decis. Support Syst.*, 2014.

[34] Thomas Kobber Panum, Kaspar Hageman, René Rydhof Hansen, and Jens Myrup Pedersen. Towards adversarial phishing detection. In *CSET*, 2020.

[35] Nicolas Papernot, Patrick D. McDaniel, Ian J. Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *AsiaCCS*, 2017.

[36] Nicolas Papernot, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *EuroS&P*, 2015.

[37] Eduardo H. M. Pena, Eduardo C. de Almeida, and Felix Naumann. Discovery of approximate (and exact) denial constraints. In *VLDB*, 2019.

[38] Fabio Pierazzi, Feargus Pendlebury, Jacopo Cortellazzi, and Lorenzo Cavallaro. Intriguing properties of adversarial ML attacks in the problem space. In *S&P*, 2020.

[39] Kedar Potdar, Taher S Pardawala, and Chinmay D Pai. A comparative study of categorical variable encoding techniques for neural network classifiers. In *IJCA*, 2017.

[40] Kevin Roth, Yannic Kilcher, and Thomas Hofmann. The odds are odd: A statistical test for detecting adversarial examples. In *International Conference on Machine Learning*, 2019.

[41] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K. Reiter. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *CCS*, 2016.

[42] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K. Reiter. A general framework for adversarial examples with objectives. *ACM TOPS*, 22(3):16:1–16:30, 2019.

[43] Ryan Sheatsley, Blaine Hoak, Eric Pauley, Yohan Beugin, Michael J. Weisman, and Patrick McDaniel. On the robustness of domain constraints. In *CCS*, 2021.

[44] Ryan Sheatsley, Nicolas Papernot, Michael J. Weisman, Gunjan Verma, and Patrick D. McDaniel. Adversarial examples in constrained domains. *arXiv*, 2020.

[45] Reza Shokri, Marco Stronati, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *S&P*, 2017.

[46] Thibault Simonetto, Salijona Dyrmishi, Salah Ghamizi, Maxime Cordy, and Yves Le Traon. A unified framework for adversarial attack and defense in constrained feature space. In *IJCAI*, 2022.

[47] Wei Song, Xuezixiang Li, Sadia Afroz, Deepali Garg, Dmitry Kuznetsov, and Heng Yin. MAB-Malware: A reinforcement learning framework for blackbox generation of adversarial malware. In *AsiaCCS*, 2022.

[48] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *ICLR*, 2014.

[49] Leslie G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, 1984.

[50] Yash Vesikar, Kalyanmoy Deb, and Julian Blank. Reference Point Based NSGA-III for Preferred Solutions. In *SSCI*, 2018.

[51] Renjie Xiao, Zijing Tan, Haojin Wang, and Shuai Ma. Fast approximate denial constraint discovery. In *VLDB*, 2022.

[52] Weilin Xu, Yanjun Qi, and David Evans. Automatically evading classifiers. In *NDSS*, 2016.

Appendix A. Additional Technical Details

A.1. Valiant's Constraints

Inspired by Sheatsley et al. [43], we consider the boolean constraints mined by Valiant's PAC learning algorithm [49] to capture relation constraints. Specifically, as described next, we focus on a particular variant of Valiant's constraints.

Definition. Valiant's algorithm learns boolean constraint theories from the data [49]. It captures constraints as Conjunctive Normal Form (CNF) logical formulas, where each predicate enforces the equality of some feature to a constant. Following Sheatsley et al., we use Valiant's algorithm to mine permissive constraint theories,[2] enabling us to evaluate robustness against the least constrained adversary. Doing so also renders the mining process tractable, as mining less permissive constraint theories with Valiant's algorithm is often intractable in practice.

Binning. The complexity of the Valiant's algorithm is exponential in the size of the features' support (i.e., values they can admit). In practice, many features have large support (e.g. continuous features may admit as many distinct values as the number of samples). Hence, to make Valiant's

2. Specifically, we set k=1

algorithm feasible to run, we discretize continuous features, similarly to prior work [43]. Particularly, we discretize continuous features by binning their values into one of k_{bin} bins formed by applying the k-means algorithm[3] to each feature's support. We note that k-means led to the best observed performance, higher than the OPTICS algorithm considered in prior work [43]. We denote the discretized support of feature i as $\widetilde{S}_i$ (a discrete set of size k_{bin}).[4] For features that were not discretized, $\widetilde{S}_i$ simply stands for the feature's actual support. We denote the discretized feature space $\widetilde{X}$.

Parameterization. Our implementation runs over the following form of possible constraints:

$$\Gamma := \left\{ \forall x \in X. \vee_{i\in[d]} \left(x_i = s_i\right) \mid s_1 \in \widetilde{S}_1, \ldots, s_d \in \widetilde{S}_d \right\}$$

Put simply, each constraint, defined by a support vector, $(s_1, \ldots s_d)$, requires that each sample x in the feature space would have at least a single coordinate, $j \in [d]$, where it identifies with the values of the support vector (i.e., $x_j = s_j$). As noted to earlier, other parameterizations of Valiant's constraint space consider a *set* of possible values per clause (e.g., a clause can be of form $x_i \in S'_i$ instead of $x_i = s'_i$), leading to an exponential growth in the mining complexity.

Mining Process. Given the training set after discretization, $\widetilde{X}_{train}$, Valiant's algorithm begins with $T := \Gamma$, the space of all possible constraints, and returns returns T, the set of the mined constraints [49]. The algorithm iterates on each (discretized) sample $\widetilde{x} \in \widetilde{X}_{train}$, and checks, for each potential constraint $t \in T$, whether $\widetilde{x}$ satisfies it or not. If $\widetilde{x}$ does not satisfy t , then t is discarded from T. In the worst case, when no constraint is discarded, we get a time complexity of $O\left(\left|\widetilde{X}_{train}\right| \Pi_{i=1}^{d} \left|\widetilde{S}_i\right|\right)$.

Advantages of DCs Over Valiant's. Besides the empirical evidence above, we opt for DCs for their desirable properties over Valiant's constraints. First, DCs using soft constraints can potentially identify valuable constraints that may be excluded by Valiant's due to a few noisy sample. Furthermore, as opposed to Valiant's, mining DCs does not necessitate data discretization and is applied to the raw form of the samples. This absence of discretization enhances the representativeness of the constraints; for example, Valiant's fails to model dependencies within the discretized ranges, which could be vital for addressing small perturbations. Finally, the ability to work directly with raw data also enables DCs to incorporate additional, important assertions on the literals, such as setting precise perturbation bounds.

A.2. Ranking DCs

As mentioned in §4.1, in our framework, we use only a subset of the mined DCs, based on a heuristic ranking scheme. Here, we elaborate on this scheme, establishing it based on metrics from data-integrity literature.

3. We use `sklearn`'s `KBinsDiscretizer` with k-means.

4. Amount of bins per dataset: k_{bin}=4 for *bank*, k_{bin}=6 for *phishing*, and k_{bin}=4 for *adult*.

In this work, we use the following standard metrics that quantify different aspects of DCs (e.g., the "interestingness" constraints [14] as well as their satisfaction by in-domain samples [29], [37], [51]):

- **Succinctness** [14]. Following "Occam's Razor," an insightful DC would be short and concise. This metric stands for how close the DC's amount of predicates to the minimum length of predicates. The closer, the better.
- **Coverage** [14]. Even if a DC is satisfied, the amount of satisfied inner-predicates varies. This amount of satisfied predicates can be seen as the support in the data. *Coverage* measures, for a single DC, the weighted average of the amount of satisfied predicates, over the different sample pairs in the data.
- **Pairs violation** [29], [37], [51] measures the proportion of sample pairs violated by the DC (out of all sample pairs). Also known as the *violation rate* when considered in *soft* DCs.
- **Sample violation** [29] quantifies the proportion of samples for which exist *any* pair of that violates the DC, over all samples.

To use these metrics in our ranking scheme, the following process is applied after mining the DCs:

1) We sample a subset (3K) samples from the training set.
2) We select (10K) the DCs to rank, according to an efficiently calculated metric–succinctness.
3) Each DC is assigned a score according to a linear combination of the four metrics. The coefficients for the linear combination are selected by manually inspecting the ranking provided by different values.
4) For each DC, we choose the most satisfying *other* tuples, calculated as the satisfaction rate of the DCs with the *other* tuple.
5) We choose the n_{dcs} top-ranked DCs, and, for each, we choose the n_{tuples} top-ranked *other* tuples.

Appendix B. Additional Experimental Setup Details

MoEvA2's Norm Function Throughout our experiments (§5), when using MoEvA2 [46], we set it to run while minimizing the ℓ_2-norm, although we are measuring a variation of ℓ_∞-norm. This choice was taken after finding that the feasible success of MoEvA2 is higher under this choice of norm function.

We ran an experiment similar to our original analysis (§6.3), and compare the performance of two instances of MoEvA2, one with the distance objective of ℓ_2 distance and the other one with the ℓ_∞ objective. We report the result in Fig. 10, where we observe that, although not intuitive, incorporating the ℓ_2 objective provides better performance under our standardized-ℓ_∞ metric.

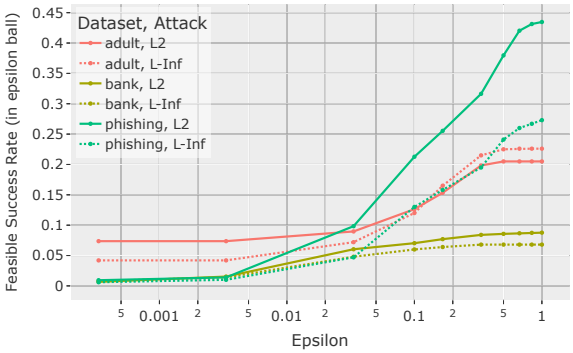


Figure 10: Comparing the feasible success rates of MoEvA2 with DCs under ℓ_∞- and ℓ_2-norms.

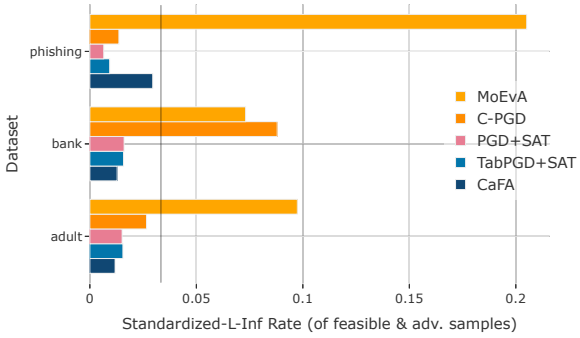


Figure 11: The *standardized-*ℓ_∞ cost of feasible and successful adversarial samples (lower is better). The vertical line corresponds to CaFA's cost bound ($\epsilon = \frac{1}{30}$).

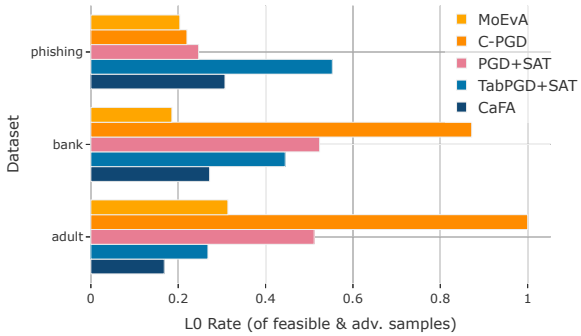


Figure 12: The ℓ_0 cost of feasible and successful adversarial samples (lower is better). To enable cross-dataset comparisons, we normalized values by the number of features.

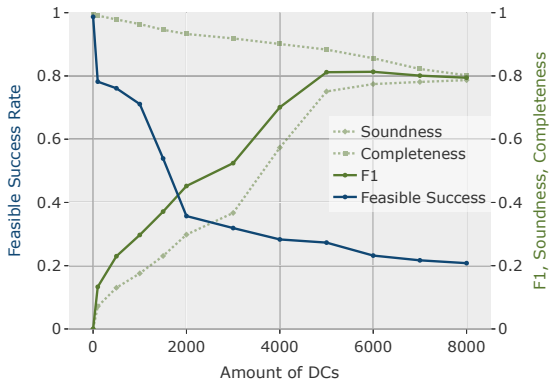


Figure 13: Constraints' quality and CaFA's feasible success rates for different choices of DC sets on the *bank* dataset.

Appendix C. Additional Experimental Results

C.1. Cost Analysis

Here, we include a more fine-grained comparison of attack costs. The measures were taken on the subset of successful and feasible adversarial samples, having considered the trade-off between the feasible success rate and the costs in §6.3. We report the comparison over the *standardized-*ℓ_∞ cost in Fig. 11 and over the ℓ_0 cost in Fig. 12.

C.2. Impact of Constraint Choice

Bank Dataset. A key trade-off lies between the feature-space constraints and attacks' feasible success. In general, the larger the constraint sets, the more challenging attacks are. To better understand this relationship, we explored how attacks' feasible success rates behave when varying the number of constraints (thus, also attaining different soundness-completeness tradeoffs). We first ran the analysis on the *bank* dataset, running CaFA with varying constraint sets by controlling the n_{dcs} parameter. For each constraint set, we measured feasible success, empirical soundness, and completeness. The results are shown in Fig. 13.

We observed that, *within* the considered set sizes, the constraint set quality (indicated by the F1 score) mostly increases with the set size, reaching its highest value when considering 5K constraints, the number of constraints used in most of our experiments. This increase in F1 scores correlates with a decrease in attack performance. For instance, when using a constraint set with a 13% F1 score, CaFA achieved a feasible success rate of ∼78%. Conversely, a higher quality constraint set with an 80% F1 score leads to a decline of feasible success rate to ∼30%. These findings emphasize the challenging landscape posed by high-quality constraints.

Phishing Dataset. Thus far, we have evaluated how the choice of constraints impact soundness, completeness, and feasible success rates on the *bank* dataset. To further validate the findings, we report on a similar evaluation with the *phishing* dataset. In this experiment, we used the exact setting and measures considered for the *bank* dataset. However, as no golden constraints were derived for the *phishing* dataset in prior work, we derived those manually

ID	Constraint	Comp. Rate (%)
#1	$x.NumNumericChars < x.UrlLength$	100.0%
#2	$(x.NumSensitiveWords * 2) < x.UrlLength$	100.0%
#3	$x.PctNullSelfRedirectHyperlinks = 1 \implies x.PctExtHyperlinks = 0$	100.0%
#4	$(x.PctNullSelfRedirectHyperlinks + x.PctExtHyperlinks) \leq 1$	99.9%
#5	$x.PctExtNullSelfRedirectHyperlinksRT = 1 \implies$ $x.PctNullSelfRedirectHyperlinks + x.PctExtHyperlinks < 0.3$	100.0%

TABLE 7: Manually derived golden constraints for the *phishing* dataset. We describe each constraint relative to a feature-space vector $x \in X$, and report the percentage of samples from the dataset that comply with it.

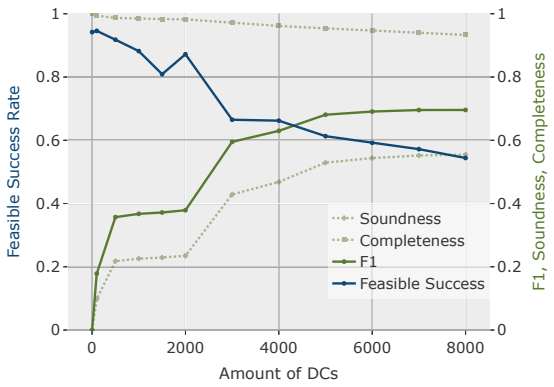


Figure 14: Constraints' quality and CaFA's feasible success rates for different choices of DC sets on the *phishing* dataset.

(see Tab. 7).

Fig. 14 reports the results. Consistent with the trends observed with *bank* dataset, we note that the F1 score increases as the size of the constraint set expands. We also observe an inverse relationship between the size of the constraint set and the attack's feasible success. Specifically, this evaluation further justifies our choice of DCs configuration, as determined by our analysis of the *bank* DCs (§6.1)—this configuration, with n_{dcs}=5K and n_{tuples}=1 led to the (roughly) highest of F1 score on the *phishing* dataset too.

C.3. Case Study: Evading *Phishing* Detector

In this study, exemplifying one of 11 samples[5] inspected in the case study (§6.6), we demonstrate a translation of adversarial example attacks from the feature-space to the real-world problem-space, using a real phishing website as an example. Due to the absence of raw data of the *phishing* dataset [13], we rely on a recently identified phishing website taken from the PhishTank archive (Fig. 15). Using this sample, we highlight the importance of CaFA's threat model (§3) in a realistic scenario.

First, we extracted the website's features, by implementing the feature-extraction logic (the dataset does not include code for feature extraction). Subsequently, we employed CaFA to generate an adversarial sample in the feature space (see Tab. 8). This sample served as a *recipe* for modifying

5. PhishTank sample IDs: 8302119, 8307185, 8314314, 8309078, 8309085, 8310709, 8309529, 8309492, 8313454, 8313452, 8314312.

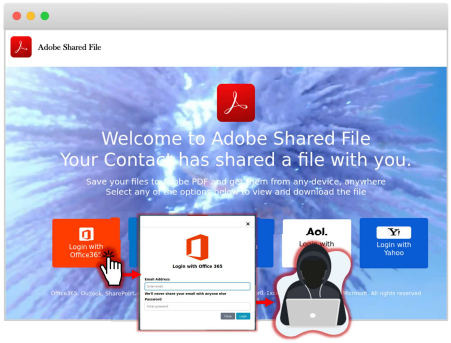


Figure 15: A phishing page for which we demonstrate an evasion. The page was taken PhishTank (ID: 8302119). The page tricks users to login to various services, such as Office 365, and sends their credentials to a malicious server.

	UrlLength	NumNumericChars	ExtMetaScriptLinkRT	SubmitInfoToEmai	...
x	112	15	0	False	...
x'	**113**	**14**	0	**True**	...

TABLE 8: A real-world *phishing* website's feature-space instance, and its corresponding perturbed sample created by CaFA. Perturbed features are **boldfaced** (we omit non-perturbed features).

the original URL and HTML code to fool the ML-based detector.

For example, following the *recipe* (Tab. 8) we adjusted the website's URL and HTML. Initially, we were required to increase the length of the URL by one character (*UrlLength*), while decreasing a single numerical character from it (*NumNumericChars*). Secondly, for the HTML, we were required to introduce the *mailto* function (*SubmitInfoToEmail*). To avoid affecting other features and to keep change imperceptible to users, we simply inserted it in an unreachable function in the JavaScript section. These minor alterations were sufficient to deceive the detector, while the website remained functional and visually identical.

C.4. Feasible Success Rates With DCs

In §6.2 we evaluated various attacks on the constrained-feature-space imposed by DCs. We report the results of this experiment in details in Tab. 9.

C.5. Feasible Success Rates With Valiant's

In §6.5, we evaluated CaFA on the *phishing* dataset while integrating Valiants' constraints to enable comparison with the work of Sheatsley et al. [43]. Tab. 10 reports results on all datasets.

Model	Dataset	Attack	C.Acc.?	$\ell_{0\|M\wedge C}\downarrow$	$\ell_{\infty\|M\wedge C}\downarrow$	$M\uparrow$	$C\uparrow$	$M\wedge C\uparrow$
MLP	*phishing*	TabPGD(1/30, Struc.)	×	5.54 $_{\pm 0.04}$	0.007$_{\pm 0.0003}$	99.80%$_{\pm 0.01\%}$	19.43%$_{\pm 0.2\%}$	19.33%$_{\pm 0.3\%}$
		TabPGD(1/30, Struc.) + SAT-Solver(1/30, DCs)		5.52 $_{\pm 0.06}$	0.009$_{\pm 0.0002}$	78.53%$_{\pm 1.2\%}$	88.46%$_{\pm 0.4\%}$	**67.03%**$_{\pm 1.4\%}$
		TabPGD(1/30, Struc.) + Tab-CW-L0	×	2.07$_{\pm 0.01}$	0.0005$_{\pm 0.0003}$	94%$_{\pm 0.6\%}$	49.56%$_{\pm 1.1\%}$	47.43%$_{\pm 0.9\%}$
		TabPGD(1/30, Struc.) + Tab-CW-L0 + SAT-Solver(1/30, DCs)		3.07$_{\pm 0.02}$	0.02$_{\pm 0.001}$	63.5%$_{\pm 0.9\%}$	93.3%$_{\pm 1.0\%}$	58.5%$_{\pm 1.4\%}$
		PGD	×	-	-	100%$_{\pm 0\%}$	0.30%$_{\pm 0.2\%}$	0.30%$_{\pm 0.2\%}$
		PGD + SAT-Solver(1/30, DCs)		-	-	95.8.%$_{\pm 0.3\%}$	6.5%$_{\pm 0.3\%}$	2.3%$_{\pm 0.4\%}$
		C-PGD(DCs)		-	-	99.36%$_{\pm 0.2\%}$	3.70%$_{\pm 0.7\%}$	3.40%$_{\pm 0.7\%}$
		MoEvA2(Struc.)	×	1.87$_{\pm 0.004}$	0.17$_{\pm 0.003}$	53.6%$_{\pm 0.1\%}$	79.26%$_{\pm 0.003\%}$	38.66%$_{\pm 0.2\%}$
		MoEvA2(Struc. and DCs)		2.03$_{\pm 0.01}$	0.20$_{\pm 0.004}$	57.0%$_{\pm 1.0\%}$	80.16%$_{\pm 0.5\%}$	43.46%$_{\pm 1.5\%}$
	bank	TabPGD(1/30, Struc.)	×	6.173$_{\pm 0.14}$	0.012$_{\pm 0.0005}$	77.10%$_{\pm 1.8\%}$	10.16%$_{\pm 0.4\%}$	9.9%$_{\pm 0.5\%}$
		TabPGD(1/30, Struc.) + SAT-Solver(1/30, DCs)		6.67$_{\pm 0.13}$	0.015$_{\pm 0.0006}$	54.46%$_{\pm 2.1\%}$	56.73%$_{\pm 0.2\%}$	**28.43%**$_{\pm 1.2\%}$
		TabPGD(1/30, Struc.) + Tab-CW-L0	×	2.90$_{\pm 0.26}$	0.007$_{\pm 0.001}$	78.13%$_{\pm 1.9\%}$	16.2%$_{\pm 0.6\%}$	16.0%$_{\pm 0.7\%}$
		TabPGD(1/30, Struc.) + Tab-CW-L0 + SAT-Solver(1/30, DCs)		4.07$_{\pm 0.3}$	0.012$_{\pm 0.0006}$	44.7%$_{\pm 2.3\%}$	66.36%$_{\pm 0.2\%}$	**27.26%**$_{\pm 1.3\%}$
		PGD	×	-	-	100.0%$_{\pm 0.0\%}$	4.93%$_{\pm 0.4\%}$	4.93%$_{\pm 0.4\%}$
		PGD + SAT-Solver(1/30, DCs)		7.85$_{\pm 0.18}$	0.16$_{\pm 0.0001}$	30.9%$_{\pm 1.2\%}$	48.46%$_{\pm 2.0\%}$	14.1%$_{\pm 0.9\%}$
		C-PGD(DCs)		-	-	100.0%$_{\pm 0.0\%}$	10.9%$_{\pm 0.4\%}$	10.9%$_{\pm 0.4\%}$
		MoEvA2(Struc.)	×	1.29$_{\pm 0.1}$	0.006$_{\pm 0.0008}$	20.73%$_{\pm 0.5\%}$	66.40%$_{\pm 0.7\%}$	8.43%$_{\pm 0.05\%}$
		MoEvA2(Struc. and DCs)		2.78$_{\pm 0.04}$	0.07$_{\pm 0.01}$	26.73%$_{\pm 1.2\%}$	47.5%$_{\pm 0.4\%}$	8.8%$_{\pm 0.05\%}$
	adult	TabPGD(1/30, Struc.)	×	4.44$_{\pm 0.13}$	0.011$_{\pm 0.0008}$	86.63%$_{\pm 1.2\%}$	16.7%$_{\pm 0.3\%}$	16.3%$_{\pm 0.2\%}$
		TabPGD(1/30, Struc.) + SAT-Solver(1/30, DCs)		5.09$_{\pm 0.09}$	0.015$_{\pm 0.0006}$	72.63%$_{\pm 1.0\%}$	54.03%$_{\pm 0.8\%}$	**32.5%**$_{\pm 0.8\%}$
		TabPGD(1/30, Struc.) + Tab-CW-L0	×	2.32$_{\pm 0.18}$	0.007$_{\pm 0.007}$	87.53%$_{\pm 1.5\%}$	20.8%$_{\pm 0.5\%}$	20.36%$_{\pm 0.6\%}$
		TabPGD(1/30, Struc.) + Tab-CW-L0 + SAT-Solver(1/30, DCs)		3.19$_{\pm 0.06}$	0.011$_{\pm 0.0003}$	65.16%$_{\pm 2.4\%}$	61.7%$_{\pm 1.7\%}$	**32.4%**$_{\pm 1.3\%}$
		PGD	×	-	-	100.0%$_{\pm 0.0\%}$	3.50%$_{\pm 0.1\%}$	3.50%$_{\pm 0.1\%}$
		PGD + SAT-Solver(1/30, DCs)		9.71$_{\pm 0.54}$	0.01$_{\pm 0.002}$	65.06%$_{\pm 0.007\%}$	21.16%$_{\pm 0.008\%}$	8.50%$_{\pm 1.4\%}$
		C-PGD(DCs)		-	-	100.0%$_{\pm 0.0\%}$	25.53%$_{\pm 2.8\%}$	25.36%$_{\pm 2.8\%}$
		MoEvA2(Struc.)	×	3.30$_{\pm 0.24}$	0.03$_{\pm 0.003}$	52.96%$_{\pm 0.7\%}$	49.33%$_{\pm 0.8\%}$	18.4%$_{\pm 0.5\%}$
		MoEvA2(Struc. and DCs)		5.95$_{\pm 0.23}$	0.09$_{\pm 0.004}$	58.10%$_{\pm 0.9\%}$	47.06%$_{\pm 0.6\%}$	20.5%$_{\pm 1.5\%}$
TabNet	*adult*	TabPGD(1/30, Struc.) + Tab-CW-L0	×	1.35	0.001	82.0%	22.2%	22.0%
		TabPGD(1/30, Struc.) + Tab-CW-L0 + SAT-Solver(1/30, DCs)		3.81	0.011	71.5%	60.0%	**43.7%**
		MoEvA2(Struc.)	×	1.92	0.048	56.9%	58.5%	27.2%
		MoEvA2(Struc. and DCs)		3.74	0.088	61.2%	57.9%	30.7%
	phishing	TabPGD(1/30, Struc.) + Tab-CW-L0	×	3.28	0.01	83.7%	32.6%	30.0%
		TabPGD(1/30, Struc.) + Tab-CW-L0 + SAT-Solver(1/30, DCs)		3.7	0.05	46.1%	93.6%	**44.1%**
		MoEvA2(Struc.)	×	1.90	0.16	55.7%	71.5%	33.5%
		MoEvA2(Struc. and DCs)		1.99	0.22	52.7%	79.0%	37.3%
	bank	TabPGD(1/30, Struc.) + Tab-CW-L0	×	2.52	0.003	52.5%	10.0%	9.2%
		TabPGD(1/30, Struc.) + Tab-CW-L0 + SAT-Solver(1/30, DCs)		4.58	0.009	48.0%	42.6%	**17.6%**
		MoEvA2(Struc.)	×	1.16	0.027	21.0%	66.8%	7.8%
		MoEvA2(Struc. and DCs)		2.12	0.193	27.8%	62.3%	12.6%

TABLE 9: Evaluation of our adversarial tabular attack variants (CaFA), compared to other attacks. For each attack, we mention the used parameters/information in parentheses by $AttackName(\epsilon, Constraints)$ (where ϵ stands for the *standardized-*ℓ_∞, and *constraints* names the type of constraints incorporated). We also state whether the attack used access to constraints (C. Acc.?) Then, we compare the ℓ_0 difference from the original sample (ℓ_0) averaged over the *realizable and successfully mislcassified* adversarial samples, the average *standardized-* ℓ_∞ on the same subset (l_∞), the miss-classification rate (M), the compliance with the DCs (C) and finally the combination of the latter two, which means the rate of realizable adversarial samples ($M \wedge S$). The best-performing attack, for each dataset and model, is marked with **bold**, and under the setting with *no* access to constraints we mark the best-performing attack with underline.

Dataset	Attack	Full Attacks		Disabling Access	
		Comp.	Comp. & Mis.	Comp.	Comp. & Mis.
phishing	PGD	100%	67.1%	3.3%	3.3%
	TabPGD	100%	82.5%	54.9%	54.7%
	TabPGD+CWL0	100%	**87.6%**	77.7%	77.7%
bank	PGD	100%	26.9%	54.3%	54.3%
	TabPGD	100%	**88.5%**	69.9%	66.9%
	TabPGD+CWL0	100%	87.8%	71.8%	68.2%
adult	PGD	100%	51.2%	24.5%	24.5%
	TabPGD	100%	82.7%	91.8%	79%
	TabPGD+CWL0	100%	**83.2%**	92%	79.9%

TABLE 10: Attacking with Valiant's constraints, reporting the compliance with the constraints (Comp.) and feasible success rate (Comp. and Mis.) on the full attacks and when disabling access to Valiant's constraints.

Appendix D.
Meta-Review

The following meta-review was prepared by the program committee for the 2024 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

D.1. Summary

In this paper, the authors investigate the vulnerability of machine learning models in domains with constraints. The authors propose an approach built from a series of algorithms for robustness evaluations, namely, mining constraints, enforcing datatype constraints during crafting, and projecting examples to comply with the learned domain constraints. Specifically, the authors propose using denial constraints to mine constraints from examples, integrating datatype constraints during the adversarial crafting process, and using SAT solvers (when necessary) to project invalid examples onto a constraint-complaint space. In their evaluation, the authors investigate robustness evaluations with 3 datasets, against existing baselines, and observe various performance curves of constraint-learning configurations showing constraint-complaint adversarial examples can be often found.

D.2. Scientific Contributions

- Independent Confirmation of Important Results with Limited Prior Research.
- Creates a New Tool to Enable Future Science.
- Provides a Valuable Step Forward in an Established Field.

D.3. Reasons for Acceptance

1) This paper provides a valuable step forward in an established field. Specifically, the authors develop CaFA to craft adversarial examples while satisfying mined constraints.
2) Experimental results demonstrate that CaFA achieves a higher (feasible) attack success rate with lower attack costs compared to existing attacks, including PGD, C-PGD, and MoEvA2.