Adversarial Robustness for Tabular Data through Cost and Utility Awareness

Anonymous Author(s)
Affiliation
Address
email

Abstract

Many machine learning applications (credit scoring, fraud detection, etc.) use data in the *tabular domains*. Adversarial examples can be especially damaging for these applications. Yet, existing works on adversarial robustness mainly focus on machine-learning models in the image and text domains. We argue that due to the differences between tabular data and images or text, existing threat models are inappropriate for tabular domains. These models do not capture that cost can be more important than imperceptibility, nor that the adversary could ascribe different value to the utility obtained from deploying different adversarial examples. We show that due to these differences the attack and defence methods used for images and text cannot be directly applied to the tabular setup. We address these issues by proposing new cost and utility-aware threat models tailored to capabilities and constraints of attackers targeting tabular domains. We show that our approach is effective on two tabular datasets corresponding to applications for which adversarial examples can have economic and social implications.

1 Introduction

Adversarial examples are inputs deliberately crafted by an adversary to cause a classification mistake. They pose a threat in applications for which such mistakes can have a negative impact in deployed models (e.g., a financial loss [1] or a security breach [2–4]). The literature on adversarial examples largely focuses on image [5–10] and text domains [11–16]. Yet, many of the applications where adversarial examples are most damaging or helpful are not images or text. Fraud and abuse detection systems [17], risk-scoring systems [1], operate on *tabular data*: A cocktail of categorical, ordinal, and numeric features. As opposed to images, each of these features has its own different semantics. The properties of the image domain have shaped the way adversarial examples and adversarial robustness are approached in the literature [8], and have greatly influenced adversarial robustness research in the text domain. In this paper, we argue that, in tabular domains, adversarial examples are of a different nature and adversarial robustness has a different meaning.

We argue that two high-level differences need to be addressed: (a) "imperceptibility", the main constraint in existing image and text adversarial examples, is ill-defined and can be irrelevant for tabular data; and (b) existing adversarial examples assume all adversarial inputs have the same value for the adversary, while in tabular domains different adversarial examples can bring different gain to the adversary. Authors in the literature commonly formalize the concept of "an example deliberately crafted to cause a misclassification" as a *natural example*, i.e., an example coming from the data distribution, that is *imperceptibly* modified by an adversary so that the classifier's decision changes. Typically, they formalize imperceptibility as closeness according to a mathematical distance such as L_p [18, 19]. In tabular data, however, imperceptibility is not necessarily relevant. Let us consider the

Submitted to the ML Safety Workshop at the 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute

following fraud detection toy example: An adversary aims to create a fraudulent financial transaction (e.g., using stolen credit card credentials). in an app such as PayPal. Assume a fraud detector takes as input two features: (1) transaction `amount`, and (2) `device` from which the transaction was sent.

In this example, *imperceptibility is not well-defined.* Is a modification to the `amount` feature from \$200 to \$201 imperceptible? What increase or decrease would we consider perceptible? The issue is even more apparent with categorical data, for which standard distances such as L_2, L_∞ cannot even capture imperceptibility: Is a change of the `device` feature from Android to an iPhone imperceptible?

Even if imperceptibility was well-defined, *imperceptibility might not be relevant.* Should we only be concerned about adversaries making "imperceptible" changes, e.g., modifying `amount` from \$200 to \$201? What about attack vectors in which the adversary evades detection while changing the transaction by a "perceptible" amount –from \$200 to \$2,000?

We argue that in tabular data the primary constraint should be *adversarial cost*, rather than imperceptibility. Instead of looking at how visually or semantically similar feature vectors are, the focus should be on *how costly it is for an adversary to enact a modification*. Costs capture the effort of the adversary, e.g., financial or computational. "How much money does the adversary have to spend to evade the detector?" better captures the possibility that an adversary deploys an attack than establishing a threshold on the L_p distance the adversary would tolerate.

Different tabular adversarial examples are of different value to the adversary. In the literature, except for Zhang and Evans [20], most formalizations of adversarial robustness implicitly consider that all adversarial examples are equal in their importance [6, 10, 21–23]. In tabular data domains, however, different adversarial examples can bring very different *gain* to the adversary. In the fraud detection example, if a fraudulent transaction with transaction `amount` of \$2,000 successfully evades the detector, it could be significantly more profitable than a transaction with `amount` of \$200.

Using cost as the primary constraint for adversarial examples provides a natural way to incorporate the variability in adversarial gain. The adversary is expected to care about the profit that they would obtain from the attack, i.e., the difference between the cost associated with crafting an adversarial example, and the gain from successfully using it. We call this profit the *utility* of the attack. We show how utility can be incorporated into the design of attacks to ensure their economic profitability, and into the design of defences to ensure protection against adversaries that focus on profit.

In this paper, we introduce a framework to build adversarial examples tailored to tabular data. Our contributions are:

- We propose two *adversarial objectives* for tabular data that address the limitations of traditional adversary examples: a *cost-bounded* objective that substitutes standard imperceptibility constraints with adversarial costs; and a novel *utility-bounded* objective in which the adversary adjusts their expenditure on different adversarial examples proportionally to the potential gains from deploying them.
- We perform an empirical evaluation of attacks and defences with respect to proposed objectives in realistic conditions demonstrating their applicability to real-world security scenarios.

2 Adversarial Objectives in Tabular Data

Notations. The input domain's *feature space* $\mathbb{X}$ is composed of m features: $\mathbb{X} \subseteq \mathbb{X}_1 \times \mathbb{X}_2 \times \cdots \times \mathbb{X}_n$. For an example $x \in \mathbb{X}$, we denote the value of its i-th feature as x_i. Features x_i can be categorical, ordinal, and numeric. Each example is associated to a binary label $y \in \{0, 1\}$. We assume the adversary's *target* to be a binary classifier $f(x) \in \{0, 1\}$ that aims to predict the class y to which an example x belongs. In terms of capabilities, we assume the adversary can only perform modifications that are within the domain constraints. E.g in the fraud-detection example, the adversary can change the transaction amount, but the value must be positive. For a given initial labelled example (x, y), we denote the set of feasible adversarial examples that can be reached within the capabilities of the adversary as $\mathcal{F}(x, y) \subseteq \mathbb{X}$.

Cost-Bounded Objective. In the standard way to obtain an adversarial example [10], the adversary aims to construct an example that maximizes the loss $\ell(\cdot, \cdot)$ incurred by the target classifier, while keeping the L_p-distance from the initial example bounded:

$$\max_{x' \in \mathcal{F}(x,y)} \ell(\eta(x'), y) \quad \text{s.t. } \|x' - x\|_p \le \varepsilon \tag{1}$$

This objective implicitly assumes that the adversary wants to keep the adversarial example as similar to the initial example as possible in terms of the examples' feature values.

Formally, we associate a cost to the modifications needed to generate any adversarial example $x' \in \mathcal{F}(x, y)$ (from the original example (x, y)). We encode this cost as a function $c : \mathbb{X} \times \mathbb{X} \to \mathbb{R}^+$. We assume the generation cost is zero if and only if no change is enacted: $c(x, x) = 0$.

We assume that the cost-bounded adversary has a budget ε. The adversary aims to find any example that flips the classifier's decision *and* that is within the cost budget:

$$\max_{x' \in \mathcal{F}(x)} \mathbf{1}[f(x') \ne y] \quad \text{s.t. } c(x, x') \le \varepsilon \tag{2}$$

Alternatively, the adversary can optimize a standard surrogate objective which ensures that the optimization problem can be solved in practice:

$$\max_{x \in \mathcal{F}(x,y)} \ell(\eta(x), y) \quad \text{s.t. } c(x, x') \le \varepsilon, \tag{3}$$

Utility-Bounded Objective. The cost-bounded adversarial objective solves the issue of imperceptibility not being a suitable constraint for tabular data. It does not, however, tackle the problem of heterogeneity of examples: the adversary cannot assign different importance to different adversarial examples.

We propose to solve it by introducing the *gain* of an attack. The gain, $g(x^*) : \mathbb{X} \to \mathbb{R}^+$, represents the reward (e.g., the revenue) that the adversary receives if their attack using adversarial example x^* is successful. For example, in fraud detection gain would be a transaction amount, i.e. how much money a fraudster can steal.

We also introduce the concept of *utility* of a successful attack as the net benefit of launching the attack. We define the utility $u_{x,y}(x^*)$ of an attack mounted with adversarial example x^* as simply the gain minus the costs:

$$u_{x,y}(x^*) \triangleq g(x^*) - c(x, x^*), \tag{4}$$

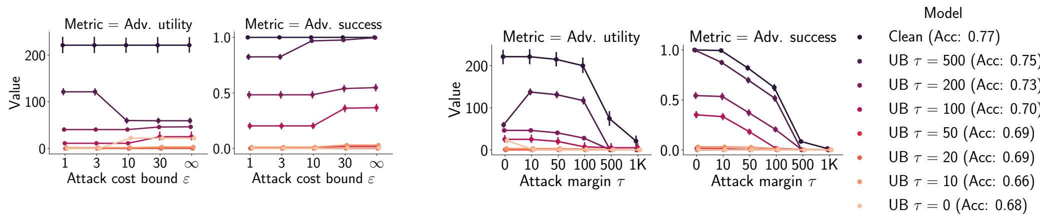
where (x, y) is the initial example.

The adversary can learn whether an example x^* evades the classifier or not (i.e., whether $f(x^*) \ne y$). Then, they can decide to deploy an attack with an adversarial example x^* only if the utility of the attack exceeds a given *margin* $\tau \ge 0$. Otherwise, the adversary discards this adversarial example. Formally, we can model this process by using a *utility constraint* instead of a cost constraint:

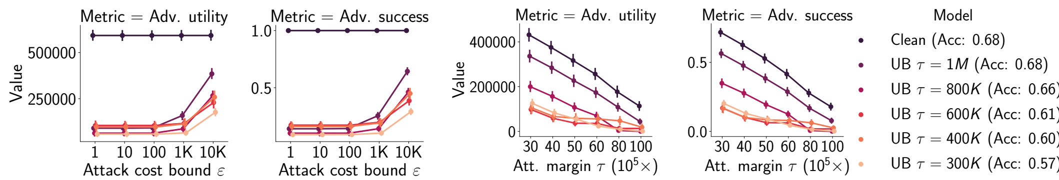
$$\max_{x \in \mathcal{F}(x,y)} \mathbf{1}[f(x) \ne y] \quad \text{s.t. } u_{x,y}(x) \ge \tau \tag{5}$$

Related work on adversarial costs. Our generic cost-bounded objective is not the only possible approach to model attacks in tabular domains. For example, works on adversarial robustness in the context of decision tree-based classifiers often use per-feature constraints as adversarial constraints [24–26]. At the low level, these constraints are formalized either as bounds on L_∞ distance [25, 26], or using functions determining constraints for each specific feature value [24]. In these approaches feature constraints are independent. Such independence simplifies the problem, e.g., the usage of L_∞ independent constraints enables to split a multidimensional optimization problem into a combination of simple one-dimension tasks [25]; or to limit the set of points affected by the split change [24].

Related work on utility-oriented adversaries. The literature on *strategic classification* also considers utility-oriented objectives [27–29] for their agents. In this body of work, however, agents are not considered adversaries, and the gain is typically limited to $\{+1, -1\}$ reflecting the classifier decision. Our model supports arbitrary gain, which enables us to model broader interests of the adversary such as revenue. Only the work by Sundaram et al. [30] supports gains different from $+1$ or -1, but they focus on PAC-learning guarantees in the case of linear classifiers, whereas our goal is to provide practical attack and defence algorithms for a wider family of classifiers.



(a) `IEEECIS`



(b) `HomeCredit`

Figure 1: *Utility-Bounded adversarial training* for different adversarial utility margins τ. Evaluation against cost-bounded (left) and utility-bounded (right) adversaries. We show the adversary's success and utility (y-axis) versus the adversary's attack budget ϵ or desired margin τ (x-axis). On `HomeCredit`, the UB training decreases the performance of both UB and CB attacks, being robustness better against the former. Even when enabling a large profit margin ($\tau = 1M$) the attack success rate decreases by 40%, at the same time not affecting the accuracy.

3 Algorithms and Evaluation

In the full version of this work, we introduce algorithms for attacks and defences within the proposed adversarial models. We briefly summarize them next, with details provided in the Appendix.

Attack Strategies For the evaluation of our adversarial models, we implement attacks within both adversarial objectives using a greedy search algorithm. We describe the algorithm and its design choices in Appendix A. As a comparison baseline, we adapt the PGD algorithm [10], a common algorithm for generating adversarial examples, to our cost model, similarly to Ballet et al. [31]. In Appendix C.2 we show that the greedy algorithm is efficient and outperforms a PGD-based baseline.

Defence With Adversarial Training To train adversarially robust models, we relax the constraint sets of the original problems, simplifying them to weighted L_1 ball constraints. With such relaxed constraints, a PGD-based adversarial training [10] with projection onto the weighted L_1 ball becomes feasible. We detail this method in Appendix B. For the evaluation of the method, we use two datasets: `HomeCredit` [32] and `IEEECIS` [33], for which we estimate realistic cost and gain models (see Appendix E. In Fig. 1, we show the results of the evaluation for models trained against the utility-bounded adversary. These models show decent performance against cost-bounded, close to "classical", adversaries. In Appendix C.3, we detail the experimental setup, and show the comparisons of training against both adversarial objectives, and a detailed study of accuracy-robustness tradeoffs.

4 Conclusions

In this paper, we revisited the problem of adversarial robustness when the target machine-learning model operates on tabular data.We introduced a new framework to design attacks and defences that account for the constraints existing in tabular adversarial scenarios: adversaries are limited by a budget to modify features, and adversaries may assign different utilities to different examples. Evaluating these attacks and defences on three real datasets we showed that our novel utility-based defence mechanism, not only generates models robust against utility-aware adversaries, but also against adversaries with a limited budget. On the contrary, performing adversarial training considering a cost-bounded adversary—as traditionally done in the literature—is a poor defence against adversaries focused on utility in some scenarios.

References

[1] Salah Ghamizi, Maxime Cordy, Martin Gubri, Mike Papadakis, Andrey Boytsov, Yves Le Traon, and Anne Goujon. Search-based adversarial testing and improvement of constrained credit scoring systems. In *ESEC/FSE*, 2020.

[2] Ambra Demontis, Marco Melis, Battista Biggio, Davide Maiorca, Daniel Arp, Konrad Rieck, Igino Corona, Giorgio Giacinto, and Fabio Roli. Yes, machine learning can be more secure! A case study on android malware detection. *CoRR*, 2017.

[3] Kathrin Grosse, Praveen Manoharan, Nicolas Papernot, Michael Backes, and Patrick D. McDaniel. On the (statistical) detection of adversarial examples. *CoRR*, 2017.

[4] Bojan Kolosnjaji, Ambra Demontis, Battista Biggio, Davide Maiorca, Giorgio Giacinto, Claudia Eckert, and Fabio Roli. Adversarial malware binaries: Evading deep learning for malware detection in executables. *CoRR*, 2018.

[5] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *CoRR*, 2013.

[6] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *CoRR*, 2014.

[7] Nicolas Papernot, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *Euro S&P*, 2016.

[8] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: A simple and accurate method to fool deep neural networks. In *CVPR*, 2016.

[9] Nicholas Carlini and David A. Wagner. Towards evaluating the robustness of neural networks. In *S&P*, 2017.

[10] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *CoRR*, 2017.

[11] Puyudi Yang, Jianbo Chen, Cho-Jui Hsieh, Jane-Ling Wang, and Michael I Jordan. Greedy attack and Gumbel attack: Generating adversarial examples for discrete data. *JMLR*, 2020.

[12] Yutong Wang, Yufei Han, Hongyan Bao, Yun Shen, Fenglong Ma, Jin Li, and Xiangliang Zhang. Attackability characterization of adversarial evasion attack on discrete data. In *KDD*, 2020.

[13] Boxin Wang, Hengzhi Pei, Han Liu, and Bo Li. AdvCodec: Towards a unified framework for adversarial text generation. *CoRR*.

[14] Qi Lei, Lingfei Wu, Pin-Yu Chen, Alexandros G Dimakis, Inderjit S Dhillon, and Michael Witbrock. Discrete adversarial attacks and submodular optimization with applications to text classification. *CoRR*, 2018.

[15] Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. Hotflip: White-box adversarial examples for text classification. In *ACL*, 2018.

[16] Bin Liang, Hongcheng Li, Miaoqiang Su, Pan Bian, Xirong Li, and Wenchang Shi. Deep text classification can be fooled. In *IJCAI*, 2018.

[17] Michele Carminati, Luca Santini, Mario Polino, and Stefano Zanero. Evasion attacks against banking fraud detection systems. In *RAID*, 2020.

[18] Mahmood Sharif, Lujo Bauer, and Michael K. Reiter. On the suitability of l_p-norms for creating and preventing adversarial examples. *CoRR*, 2018.

[19] Wei Emma Zhang, Quan Z. Sheng, Ahoud Alhazmi, and Chenliang Li. Adversarial attacks on deep learning models in natural language processing: A survey.

[20] Xiao Zhang and David Evans. Cost-sensitive robustness against adversarial examples. *CoRR*, 2018.

[21] Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric P. Xing, Laurent El Ghaoui, and Michael I. Jordan. Theoretically principled trade-off between robustness and accuracy. In *ICML*, 2019.

[22] Eric Wong, Frank Schmidt, Jan Hendrik Metzen, and J. Zico Kolter. Scaling provable adversarial defenses. *CoRR*, 2018.

[23] Ali Shafahi, Mahyar Najibi, Amin Ghiasi, Zheng Xu, John P. Dickerson, Christoph Studer, Larry S. Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! In *NeurIPS*, 2019.

[24] Yizheng Chen, Shiqi Wang, Weifan Jiang, Asaf Cidon, and Suman Jana. Cost-aware robust tree ensembles for security applications. In *USENIX*, 2021.

[25] Maksym Andriushchenko and Matthias Hein. Provably robust boosted decision stumps and trees against adversarial attacks. 2019.

[26] Hongge Chen, Huan Zhang, Duane Boning, and Cho-Jui Hsieh. Robust decision trees against adversarial examples. In *ICML*, 2019.

[27] Moritz Hardt, Nimrod Megiddo, Christos H. Papadimitriou, and Mary Wootters. Strategic classification. In *ITCS*, 2016.

[28] Jinshuo Dong, Aaron Roth, Zachary Schutzman, Bo Waggoner, and Zhiwei Steven Wu. Strategic classification from revealed preferences. In *EC*, 2018.

[29] Smitha Milli, John Miller, Anca D Dragan, and Moritz Hardt. The social cost of strategic classification. In *FAT**, 2019.

[30] Ravi Sundaram, Anil Vullikanti, Haifeng Xu, and Fan Yao. Pac-learning for strategic classification. In Marina Meila and Tong Zhang, editors, *ICML*, 2021.

[31] Vincent Ballet, Jonathan Aigrain, Thibault Laugel, Pascal Frossard, Marcin Detyniecki, et al. Imperceptible adversarial attacks on tabular data. In *Robust AI in FS NeurIPS Workshop*, 2019.

[32] Kaggle. Home credit default risk, 2019. URL `https://www.kaggle.com/c/home-credit-default-risk`.

[33] Kaggle. IEEE-CIS fraud detection, 2019. URL `https://www.kaggle.com/c/ieee-fraud-detection`.

[34] Bogdan Kulynych, Jamie Hayes, Nikita Samarin, and Carmela Troncoso. Evading classifiers in discrete domains with provable optimality guarantees. *arXiv preprint arXiv:1810.10939*, 2018.

[35] Roni Stern, Rami Puzis, and Ariel Felner. Potential search: A bounded-cost search algorithm. In *ICAPS*, 2011.

[36] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Sys. Sci. and Cybernetics*, 1968.

[37] Roni Stern, Ariel Felner, Jur van den Berg, Rami Puzis, Rajat Shah, and Ken Goldberg. Potential-based bounded-cost search and anytime non-parametric A*. *Artificial Intelligence*, 2014.

[38] Richard E. Korf. Iterative-Deepening-A*: An optimal admissible tree search. In *Joint Conference on Artificial Intelligence*, 1985.

[39] Rina Dechter and Judea Pearl. Generalized best-first search strategies and the optimality of A*. *J. ACM*, 1985.

[40] Samir Khuller, Anna Moss, and Joseph Naor. The budgeted maximum coverage problem. *Inf. Process. Lett.*, 1999.

[41] Laurence A. Wolsey. Maximising real-valued submodular functions: Primal and dual heuristics for location problems. *Math. Oper. Res.*, 1982.

[42] Eden Levy, Yael Mathov, Ziv Katzir, Asaf Shabtai, and Yuval Elovici. Not all datasets are born equal: On heterogeneous data and adversarial examples. *CoRR*, 2020.

[43] Alex Kantchelian, J. D. Tygar, and Anthony D. Joseph. Evasion and hardening of tree ensemble classifiers. In *ICML*, 2016.

[44] Francesco Cartella, Orlando Anunciacao, Yuki Funabiki, Daisuke Yamaguchi, Toru Akishita, and Olivier Elshocht. Adversarial attacks for tabular data: Application to fraud detection and imbalanced data. *SafeAI Workshop at AAAI*, 2021.

[45] Eric Wong, Leslie Rice, and J. Zico Kolter. Fast is better than free: Revisiting adversarial training. In *ICLR*, 2020.

[46] Konstantinos Slavakis, Yannis Kopsinis, and Sergios Theodoridis. Adaptive algorithm for sparse system identification using projections onto weighted l1 balls. In *ICASSP*, 2010.

[47] Guillaume Perez, Sebastian Ament, Carla Gomes, and Michel Barlaud. Efficient projection algorithms onto the weighted l1 ball, 2020.

[48] Stefano Calzavara, Claudio Lucchese, Gabriele Tolomei, Seyum Assefa Abebe, and Salvatore Orlando. Treant: training evasion-aware decision trees. *Data Min. Knowl. Discov.*, 2020.

[49] Daniël Vos and Sicco Verwer. Efficient training of robust decision trees against adversarial examples. In Marina Meila and Tong Zhang, editors, *ICML*, 2021.

[50] Zafar Gilani, Ekaterina Kochmar, and Jon Crowcroft. Classification of twitter accounts into automated agents and human users. In *ASONAM*, 2017.

[51] Experian. What is piggybacking credit, 2019. URL `https://www.experian.com/blogs/ask-experian/what-is-piggybacking-credit/`.

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

[53] Ira Pohl. Heuristic search viewed as path finding in a graph. *Artif. Intell.*, 1970.

[54] Pratyush Maini, Eric Wong, and Zico Kolter. Adversarial robustness against the union of multiple perturbation models. In *International Conference on Machine Learning*, 2020.

A Finding adversarial examples in tabular domains

In this section, we propose practical algorithms for finding adversarial examples suitable to achieve the adversarial objectives we introduce in Section 2.

A.1 Graphical Framework

The optimization problems in Section 2 can seem daunting due to the large cardinality of $\mathcal{F}(x, y)$ when the feature space is large. To make the problems tractable, we transform them into graph-search problems, following the approach by Kulynych et al. [34]. Consider a *state-space graph* $\mathcal{G}(x) = (V, E)$. Each node corresponds to a feasible example in the feature space, $V = \mathcal{F}(x, y) \cup \{x\}$. Edges between two nodes x and x' exist if and only if they differ in value of one feature: there exists $i \in [n]$ such that $x_i \neq x'_i$, and $x_j = x'_j$ for all $j \neq i$. In other words, the immediate descendants of a node in the graph consist of all feasible feature vectors that differ from the parent in exactly one feature value.

Using this state-space graph abstraction, the objectives in Section 2 can be modelled as graph search problems. Even though the graph size is exponential in the number of feature values, the search can be efficient, because the search does not need the graph to be complete. Thus, we can construct the graph on the fly.

Building the state-space graph is straightforward when features take discrete values. To encode continuous features in the graph we discretize them by only considering changes to a continuous feature i that lie within a finite subset of its domain $\mathbb{X}_i$ — in particular, on a discrete grid. The search efficiency depends on the size of the grid. As the grid gets coarser, finding adversarial examples becomes easier. This efficiency comes at the cost of potentially missing adversarial examples that are not represented on the grid but could fulfil the adversarial constraints with less cost or higher utility.

A.2 Attacks as Graph Search

In the remainder of the paper we make the following assumptions about the adversarial model:

Assumption 1 (Modular costs)**.** *The adversary's costs are modular: they decompose by features. Formally, changing the value of each feature i from x_i to x'_i has the associated cost $c_i(x_i, x'_i) > 0$, and the total cost of modifying x into x' is a sum of individual feature-modification costs:*

$$c(x, x') = \sum_{i}^{n} c_i(x_i, x'_i) \tag{6}$$

The state-space graph can encode modular costs by assigning weights to the graph edges. An edge between x and x' has an associated weight of $c_i(x_i, x'_i)$, where i is the index of the feature that differs between x and x'. For pairs of examples $x^{(0)}$ and $x^{(t)}$ that differ in more than one feature, the cost $c(x^{(0)}, x^{(t)})$ is the sum of the edge costs along the shortest path from $x^{(0)}$ to $x^{(t)}$.

Assumption 2 (Constant gain)**.** *For any initial example (x, y), the adversary cannot change the gain:*

$$\forall x' \in \mathcal{F}(x, y) : \quad g(x) = g(x') \tag{7}$$

This follows the approach in utility-oriented strategic classification (as detailed in Section 2). This assumption is not formally required for our attack algorithms, described next in this section, but we focus on this setting in our empirical evaluations in Appendix C.

Strategies to find adversarial examples. Under the constant per-instance gain, and modular-cost assumptions, the cost-bounded and utility-bounded adversaries look for any adversarial example that is within a (per-example) cost bound. These adversarial goals can be seen as instances of *bounded-cost search* [35].

We start with the *best-first search* (BFS) [36, 34], a flexible meta-algorithm that generalizes many common graph search algorithms. In its generic version (Algorithm 1) BFS keeps a bounded priority queue of *open nodes*. It iteratively pops the node v with the highest score value from the queue (best first), and adds its immediate descendants to the queue. This is repeated until the queue is empty. The algorithm returns the node with the highest score out of all popped nodes.

The BFS algorithm is parameterized by the *scoring function* $s : V \times V = \mathbb{X} \times \mathbb{X}$ and the size of the priority queue B. Different choices of the scoring function yield search algorithms suited for solving different graph-search problems, such as Potential Search for bounded-cost search [35, 37], and A* [38, 39] for finding the minimal-cost paths. When $B = \infty$, the algorithm might traverse the full graph and is capable of returning the optimal solution. As the size of B decreases, the optimality guarantees are lost. When $B = 1$ BFS becomes a *greedy* algorithm that myopically optimizes the scoring function. When $1 < B < \infty$ we get a *beam search* algorithm that keeps B best candidates at each iteration.

To achieve the adversarial objectives in Section 2, we propose to use a concrete instantiation of BFS, what we call the *Universal Greedy (UG)* algorithm. Inspired by heuristics for cost-bounded optimization of submodular functions [40, 41], we set the scoring function to balance the increase in the classifier's score and the cost of the change:

$$s(v, t) = -\frac{\eta(t) - \eta(v)}{c(v, t)} \tag{8}$$

The minus sign appears because BFS expands the lowest scores first, and we need to maximize the score. We set the beam size to $B = 1$ (greedy), which enables us to find high-quality solutions to *both* cost-bounded and utility-bounded problems at reasonable computational costs (see Appendix C).

Algorithm 1 Best-First Search (BFS)

function $\mathsf{BFS}_{B,s,\varepsilon}(x)$
  `open` $\leftarrow$ $\textsc{MinPriorityQueue}_B(x, 0)$
  `closed` $\leftarrow \{\}$
  while `open` is not empty **do**
    $v \leftarrow$ `open`.$\textsc{pop}()$
    if $v \notin$ closed **then**
      $\textsc{closed} \leftarrow \textsc{closed} \cup \{v\}$
    if $\eta(v) \geq \delta$ **then return** v
    $S \leftarrow \textsc{expand}(v)$
    for $t \in S$ **do**
      if $t \notin$ `closed` and $c(x, t) \leq \varepsilon$ **then**
        `open`.$\textsc{add}(t, s(v, t))$

A.3 Related Work on Attack Strategies

Tabular domains. Several works have proposed attacks on tabular data. Ballet et al. [31] propose to apply existing continuous attacks to tabular datasets. The authors focus on crafting imperceptible adversarial examples using standard methods from the image domain. They adapt these methods such that less "important" features (low correlation with the target variable) can be perturbed to a higher degree than other features. This corresponds to a special case within our framework, in which the feature-modification costs depend on the feature importance with the difference that Ballet et al. cannot guarantee that the proposed example will be feasible.

Levy et al. [42] suggest constructing a surrogate model capable of mimicking the target classifier. Part of this surrogate model is a feature embedding function transforming tabular data to a homogeneous continuous domain which aims to keep adversarial perturbations in the feasible set. Then, they apply projected gradient descent to produce adversarial examples in the embedding space and map the resulting examples to the tabular domains. As opposed to our methods, Levy et al. cannot provide any guarantee that the produced adversarial example will lay in the feasible set.

Finally, Kantchelian et al. [43] propose a MILP-based and a coordinate-descent attack within different L_p cost models against random-forest models.

Our attack differs from these three methods since they use L_p or similar bounds that do not capture adversarial capabilities, whereas we use a cost bound that can capture realistic constraints.

In a concurrent work, Cartella et al. [44] propose to use a "custom" norm also based on feature importance, similarly to Ballet et al. [31]. Cartella et al., however, use an adapted zero-order optimization algorithm to find adversarial examples. Although their motivation is similar to ours, our cost model is more general as we do not tie the costs to feature importance.

Text domains. Our universal greedy attack algorithm is similar to the methods for attacking classifiers that operate on text [19, 11–16]. All these works, however, make use of adversarial constraints such as restrictions on the number of modified words or sentences. These constraints do not apply to tabular domains, as simply considering "number of changes" does not address the heterogeneity of features. Our algorithms also differ from these approaches in that we incorporate complex adversarial costs in the design of the algorithms. For example, the Greedy attack by Yang et al. [11], like us, uses the target classifier's confidence for choosing the best modifications to create adversarial examples and allow to account for the number of modifications. Our framework not only considers the volume of modifications but also their cost, better reflecting the adversary's constraints.

B Defending from Adversarial Examples in Tabular Domains

The conventional approach to mitigate the risks of adversarial examples is adversarial training [6, 10]. In adversarial training, the training procedure includes adversarial examples along with natural ones. In a standard approach by Madry et al. [10], for instance, these adversarial examples are constructed by modifying natural examples x with perturbations constrained in a L_p-ball $d(x, x') < \varepsilon$, where d is an L_p distance function.

The distance function and ε encode the *threat model* that adversarial training aims to defend against. The choice of the distance function depends on the characteristics of the input domain. In most previous works, the distance function aims at capturing imperceptibility within the given the bound ε. It is commonly assumed that if $d(x, x^*) < \varepsilon$, x^* is not substantially different from x, and the adversary would use x^* to attack. Otherwise, turning x to x^* results in a perceptible adversarial example that would be detected as malicious, and those examples are assumed to be outside of the threat model. As explained in Section 2, this approach does not apply to the tabular domains. In tabular domains, imperceptibility is not necessarily a relevant constraint. Instead, the adversary's actions are constrained by feasibility and the cost of the transformation. Moreover, the tabular input domain is often a mix of discrete and continuous features, as opposed to continuous or quantized in, e.g., image domains, where adversarial examples are mostly studied.

Another difference between the image and tabular domains is the efficiency of generating adversarial examples. In images, adversarial examples used for training are generated using efficient methods such as Projected Gradient Descent (PGD) [10] or the Fast Gradient Sign Method (FGSM) [6, 45]. These approaches produce adversarial examples fast, and enable the efficient implementation of adversarial training. Fast generation, however, is not possible for tabular domains. The algorithms to produce tabular adversarial introduced in Appendix A require thousands of inference operations over the target model. Under this condition, generating one example, which is all the adversary needs to perform an attack, may not be fast, but it is clearly feasible. Generating multiple adversarial examples per natural sample, however, in the dataset that the defender needs for adversarial training quickly becomes computationally infeasible, especially if the defender is computationally constrained. This computational cost constrains our capability of evaluation (Appendix C), for which we need to repeatedly run the defences. To make the generation of adversarial examples feasible during adversarial training, we introduce approximate versions of the attacks that rely on a relaxation of initial attack constraints.

B.1 Relaxing the Constraints

Following the setting of the standard Projected Gradient Descent (PGD) method [10], adversarial training for the cost-bounded adversary could be defined as follows:

$$\min_{\theta} \max_{x' \in \mathcal{F}(x,y)} \ell(\eta_\theta(x'), y) \quad \text{s.t. } c(x, x') \leq \varepsilon, \tag{9}$$

where η_θ is a parametric classifier and θ are its parameters.

To keep the computational requirements low, we relax the problem to optimize over a convex set, which enables us to adapt the PGD method. Let us define B_ε to be the constraint region of Eq. (9):

$$B_\varepsilon(x, y) \triangleq \{(x', y) \in \mathcal{F}(x, y) \mid c(x, x') \leq \varepsilon\}$$

We construct a relaxation of B_ε in two steps:

$$B_\varepsilon \underset{(1)}{\longrightarrow} \bar{B}_\varepsilon \underset{(2)}{\longrightarrow} \tilde{B}_\varepsilon$$

(1) *Continuous relaxation.* We map B_ε into a continuous space using an *encoding function* $\phi : \mathbb{X}^n \to \mathbb{R}^m$, and a *relaxed cost function* $\bar{c} : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}^+$. Continuous relaxation is defined as:

$$\bar{B}_\varepsilon \triangleq \{(\phi(x'), y) \mid \bar{c}(\phi(x), \phi(x')) \leq \varepsilon\}, \tag{10}$$

where $(x', y) \in \mathcal{F}(x, y)$. The pair $(\phi, \bar{c})$ is designed to satisfy the following condition:

$$\forall (x', y) \in B_\varepsilon(x, y) : \ \bar{c}(\phi(x), \phi(x')) \leq c(x, x'), \tag{11}$$

ensuring that every example $(x', y) \in B_\varepsilon(x, y)$ is mapped to an element in the relaxed set, $\phi(x') \in B_\varepsilon(\phi(x), y)$. We denote the encoded value $\phi(x)$ as $\overline{x}$ for convenience.

(2) *Convex cover.* To enable adversarial training using PGD, we need that elements of the relaxed set can be projected onto the constraint region. For this purpose, we cover $\bar{B}_\varepsilon$ with a convex superset $\tilde{B}_\varepsilon$, e.g., a convex hull of $\bar{B}_\varepsilon$. The convex superset $\tilde{B}_\varepsilon$ needs to be constructed such that there exists an efficient algorithm for *projection*. For a given (x, y), and a point $t \in \mathbb{R}^m$, we want to be able to efficiently solve $\min_{t' \in \tilde{B}_\varepsilon(x,y)} \|t - t'\|_2$.

Encoding and cost functions As we assume that the cost of modifications is modular (see Appendix A.2), we define the encoding (ϕ) and cost ($\overline{c}$) functions to be modular too:

$$\phi(x) = [\phi_1(x_1), ..., \phi_n(x_n)]$$

$$\overline{c}(\phi(x), \phi(x')) = \sum_{i=1}^{n} \overline{c}_i(\phi_i(x_i), \phi_i(x_i{}'))$$

With this formulation, the problem of constructing suitable ϕ and $\overline{c}$ functions is reduced to finding $\phi_i : \mathbb{X}_i \to \mathbb{R}^{m_i}$ and $\overline{c}_i$ for each feature. If for all i both ϕ_i and $\overline{c}_i$ fulfill (11), then the modular cost $\overline{c}(\overline{x}, \overline{x}')$ fulfills (11) as well.

In the following we introduce ϕ and $\overline{c}$ functions for categorical and numeric features.

Categorical features. As encoding function $\phi(x_i)$ for categorical features we use standard one-hot encoding.

As the cost function for categorical features, we define $\overline{c}_i$:

$$\overline{c}_i(\overline{x}_i, \overline{x}_i') = \min_{t \in \mathcal{F}(x,y)} c_i(x_i, t) \cdot \frac{1}{2} \|\bar{x}_i - \bar{x}_i'\|_1,$$

where $\mathcal{F}_i(x, y)$ is the set of feasible values of the feature i. For example, let x_i be a categorical feature with 4 possible values $X_i = \{a, b, c, d\}$, and let the minimal cost of change be 2. When $x_i = b$ and $x' = c$ ($\overline{x}_i = (0, 1, 0, 0)$, $\overline{x}_i' = (0, 0, 1, 0)$ after one-hot encoding). Then, $\overline{c}_i(\overline{x}_i, \overline{x}_i') = \frac{1}{2} \cdot 2 \cdot 2 = 2$.

This cost function enables us to perform the two-step relaxation described before. First, it satisfies (11), and therefore the constraint region $\bar{B}_\varepsilon$ includes all mapped examples of B_ε. Second, we can obtain the convex superset $\tilde{B}_\varepsilon$ as a continuous L_1 ball around the mapped values $\bar{x} \in \bar{B}_\varepsilon$.

Numeric features. A *numeric feature* is a feature with values belonging to an ordered subset of $\mathbb{R}$ (e.g. integer, real). In most cases, the identity function ($\phi(x_i) = x_i$) is sufficient for numerical features. However, more complex encoding functions could also be desirable. For example, when one needs to reduce numerical errors, which can be achieved by normalizing the feature values to $[-1, 1]$, or when the cost is non-linear.

In general, projecting onto arbitrary sets can be challenging. Specifically, the bounded region B_ε could be non-convex, e.g., hypothetically, if the cost is a pathological function such as the Dirichlet function. We therefore must limit the scope of possible adversarial cost functions that we can model during adversarial training to those that are compatible with efficient projection. For this, we introduce a cost model that covers a broad class of functions for which $c_i(x_i, x_i')$ can be expressed as $K_i \cdot |\psi(x_i) - \psi(x_i')|$, where K_i is a constant and $\psi(x)$ is an invertible function.

For instance, this model covers the following exponential cost model: $c(x, x') = K \cdot |e^x - e^{x'}|$. In this case, we can encode the features as $\phi(x_i) = \psi^{-1}(x_i) \triangleq \ln(x_i)$. This transformation enables us to account for certain non-linear cost functions c with respect to the input space using linear cost functions $\overline{c}$ in the relaxed space $\bar{B}_\varepsilon$.

We define the cost function for numerical features as a piecewise-linear function, with different coefficients for increasing or decreasing the feature value:

$$\overline{c}_j(\overline{x}_j, \overline{x}_j') = c_{j-}(x) \cdot [\bar{x}_j - \bar{x}_j']^+ + c_{j+}(x) \cdot [\bar{x}_j' - \bar{x}_j]^+ \tag{12}$$

where $[t]^+$ returns t if $t > 0$, and 0 otherwise, and $c_{j-}(x)$ and $c_{j+}(x)$ encode the costs for decreasing and increasing the value of the feature j, respectively, and can vary from one initial example x to another.

Note that in this model the final cost of a modification could depend on the way in which this modification is achieved. A direct modification from x to x'' could have different cost than first modifying x to x' and then x' to x'', i.e., $\overline{c}_i(\bar{x}, \bar{x}'') \neq \overline{c}_i(\bar{x}, \bar{x}') + \overline{c}_i(\bar{x}', \bar{x}'')$.

Total cost. Given the set of categorical feature indices, $\mathcal{C}$, and the set of numeric feature indices, $\mathcal{I}$, the total cost function is:

$$\begin{aligned}\overline{c}(\overline{x}, \overline{x}') = \sum_{i \in \mathcal{C}} \min_{t \in \mathcal{F}_i(x,y)} c_i(x_i, t) \cdot \frac{1}{2} \|\bar{x}_i - \bar{x}'_i\|_1 \\ + \sum_{j \in \mathcal{I}} c_{j-}(x) \cdot [\bar{x}_j - \bar{x}'_j]^+ + c_{j+}(x) \cdot [\bar{x}'_j - \bar{x}_j]^+\end{aligned} \tag{13}$$

B.2 Adversarial Training with Projected Gradient Descent

Using the cost model introduced before, we redefine the training optimization problem in Eq. (9) to generate adversarial examples over a specific instantiation of the convex set $\tilde{B}$, as follows:

$$\min_{\theta} \max_{\tilde{x}' \in \tilde{B}_\varepsilon(x,y)} \ell(\eta_\theta(\tilde{x}'), y), \tag{14}$$

where we specify $\tilde{B}_\varepsilon$ as:

$$\tilde{B}_\varepsilon(x, y) \triangleq \{\overline{x} + \delta \mid \delta \in \mathbb{R}^m \wedge \overline{c}(\overline{x}, \overline{x} + \delta) \leq \varepsilon\}. \tag{15}$$

Thus, we can rewrite Eq. (14):

$$\begin{aligned}\min_{\theta} \quad & \max_{\delta \in \mathbb{R}^m} \ell(\eta_\theta(\overline{x} + \delta), y) \\ & \text{s.t. } \overline{c}(\overline{x}, \overline{x} + \delta) \leq \varepsilon\end{aligned} \tag{16}$$

This objective can be optimized using standard PGD-based adversarial training [10]. Due to the construction of our cost function in Eq. (13), we can use existing algorithms for projecting onto a weighted L_1-ball [46, 47] with an appropriate choice of weights. As these approaches are standard, we omit them in the main body, and provide the details in Appendix D.

B.3 Adversarial Training against a Utility-Bounded Adversary

For the utility-bounded adversary we propose to use an objective similar to (16), applying individual constraints to different samples:

$$\begin{aligned}\min_{\theta} \quad & \max_{\delta \in \mathbb{R}^m} \ell(\eta_\theta(\overline{x} + \delta), y) \\ & \text{s.t. } \overline{c}(\overline{x}, \overline{x} + \delta) \leq \varepsilon(x) \triangleq [g(\overline{x} + \delta)]_+\end{aligned} \tag{17}$$

In this formulation, we use our assumption of invariant gain (see Appendix A.2), as $g(\overline{x} + \delta) = g(\overline{x})$.

This objective aims to decrease the adversary's utility by focusing the protection on samples with high gain. The main difference with respect to the cost-constrained objective in (16) is that here we use a different cost bound for different examples $\varepsilon(x)$. This formulation enables us to directly use the PGD-based adversarial training to defend against utility-bounded adversaries as well.

B.4 Related Work on Adversarial Training

To the best of our knowledge, there are no works on adversarial training for methods based on deep learning that tackle the tabular domains. We discuss existing methods and techniques with related goals.

Adversarial robustness of decision trees. Classifiers based on decision trees are prominently used in tabular domains. The adversarial robustness of such classifiers has been studied extensively [26, 25, 48, 24, 49]. These works assume independent per-feature adversarial constraints, e.g., based on the L_∞ metric. Our adversarial models, and thus our attacks and defences, are capable of capturing a broader class of adversarial cost functions that depend on feature modifications and better model the adversary's constraints as we explain in Section 2.

C Experimental Evaluation

In this section, we show that out graph-based attacks can be used by adversaries to obtain profit, and that our proposed defences are effective at mitigating these attack's harms.

C.1 Experimental setup

C.1.1 Datasets

We perform our experiments on three tabular datasets which represent real-world applications for which adversarial examples can have social or economic implications:

- `TwitterBot` [50]. The dataset contains information about more than 3,400 Twitter accounts either belonging to humans or bots. The task is to detect bot accounts. We assume that the adversary is able to purchase bot accounts and interactions on darknet markets, thus modifying the features that correspond to the account age, number of likes, and retweets.
- `IEEECIS` [33]. The dataset contains information about around 600K financial transactions. The task is to predict whether a transaction is fraudulent or benign. We model an adversary that can modify three features for which we can outline the hypothetical method of possible modification, and estimate its cost: payment-card type, email domain, and payment-device type.
- `HomeCredit` [32]. The dataset contains financial information about $300K$ home-loan applicants. The main task is to predict whether an applicant will repay the loan or default. We use 33 features, selected based on the best solutions to the original Kaggle competition [32]. Of these, we assume that 28 can be modified by the adversary, e.g., the loan appointment time. We also use a non-linear adversarial cost model for manipulating credit scores, inspired by the practice of credit piggybacking [51].

C.1.2 Models

We evaluate our attacks against three types of ML models commonly applied to tabular data. First, an L_2-regularized *logistic regression (LR)* with a regularization parameter chosen using 5-fold cross-validation. Second, *gradient-boosted decision trees (XGBT)*. Third, *TabNet* architecture [52], a *deep learning* model. TabNet is an attentive transformer neural network specifically designed for tabular data. We optimize the number of steps as well as the capacity of TabNet's fully connected layers using grid search.

C.1.3 Adversarial Features

We assume that the feasible set consists of all positive values of numerical features and all possible values of categorical features. For simplicity, we avoid features with mutual dependencies and treat the adversarially modifiable features as independent. We detail the choice of the modifiable features and their costs in Appendix E.2.

C.1.4 Metrics

To evaluate the effectiveness of the attacks and defences, we use three main metrics:

- *Adversary's success rate:* The proportion of correctly classified examples from a test set X_{test} for which adversarial examples successfully generated using the attack algorithm $\mathcal{A}(x, y)$ evade the classifier:
$$\Pr_{(x,y)\sim X_{\text{test}}} \left[f(\mathcal{A}(x,y)) \neq y \wedge f(x) = y \right].$$
- *Adversarial cost:* Average cost of successful adversarial examples:
$$\mathbb{E}_{(x,y)\sim X_{\text{test}}} [c(x, \mathcal{A}(x,y)) \mid f(\mathcal{A}(x,y)) \neq y \wedge f(x) = y].$$
- *Adversarial utility:* Average utility (see Eq. (4)) of successful adversarial examples:
$$\mathbb{E}_{(x,y)\sim X_{\text{test}}} [u_{x,y}(\mathcal{A}(x,y)) \mid f(\mathcal{A}(x,y)) \neq y \wedge f(x) = y].$$

In all cases, we only consider correctly classified initial examples which enables us to distinguish these security metrics from the target model's accuracy. We introduce additional metrics in the experiments when needed.

C.2 Attacks Evaluation

We evaluate the attack strategy proposed in Appendix A in terms of their effectiveness, and empirically justify its design.

C.2.1 Design Choices of the Universal Greedy Algorithm

When designing attack algorithms in the BFS framework (see Algorithm 1) there are two main design choices: the scoring function and the beam size. We explore different configurations and show that our parameter choices for the Universal Greedy attack produce high-quality adversarial examples.

Beam size. We define the beam size of the Universal Greedy attack to be one. The other options that we evaluate are 10 and 100. We evaluate by running three types of attacks: cost-bounded for three cost bounds ε, and utility-bounded at the breakeven margin $\tau = 0$. The margin $\tau = 0$ is equivalent to a cost-bounded attack with a variable cost bound equal to the gain of each initial example (denoted as "Gain" in the tables).

We compute two metrics: Attack success, and the success-to-runtime ratio. This ratio represents how much time is needed to achieve the same level of success rate using each choice of the beam size. This metric is more informative for our evaluation than runtime, as runtime is just proportional to the beam size.

For feasibility reasons, we use two datasets: `TwitterBot` and `IEEECIS`. We aggregate the metrics across the three models (LR, XGBT, TabNet), and report the average. The results on `TwitterBot` are equivalent to the results on `IEEECIS`, thus for conciseness we only report `IEEECIS` results.

We find that the success rates are equal up to the percentage point for all choices of the beam size. We show the detailed numeric results in Table 7 in the Appendix. As the smallest beam size of one is the fastest to run, it demonstrates the best success/time ratio, therefore, is the best choice.

Scoring function. Recall from Eq. (8) that the scoring function is the cost-weighted increase in the target classifier's confidence:

$$s(v,t) = -\frac{\eta(t) - \eta(s)}{c(s,t)} \,,$$

which aims to maximize the increase in classifier confidence at the lowest cost.

Suitable choices for the scoring function $s(v,t)$ could be:

- A^* *algorithm* [38, 39, 34]: $s(v,t) = c(v,t) + \lambda \cdot h(t)$, where $h(t)$ is a heuristic function, which estimates the remaining cost to a solution, and $\lambda > 0$ is a greediness parameter [53]. This scoring function balances the current known cost of a candidate and the estimated remaining cost. We choose the model's confidence for the positive class, $h(x) = \eta(x)$, as heuristic function. Intuitively, this works as a heuristic, because the lower the confidence for the positive class, the more likely we are close to a solution: an example classified as the target class.
- *Potential Search* (PS) [35, 37]: $s(v,t) = {h(t)}/{\varepsilon - c(v,t)}$, which additionally takes into account the cost bound ε, thus becoming more greedy (i.e., optimizing $s(v,t) = \lambda \cdot h(t)$ with $\lambda \approx {1}/{\varepsilon}$) when the cost of the current candidate leaves a lot of room within the ε budget. We also choose $h(x) = \eta(x)$ as heuristic function.
- *Basic Greedy*: $s(v,t) = -{\eta(t)}/{c(s,t)}$, which aims to maximize the classifier's confidence, yet balance it with the incurred cost. Unlike Eq. (8), this scoring function does not care about the relative increase of the confidence, only about its absolute value.

We evaluate the choice of the scoring function on the `TwitterBot` and `IEEECIS` datasets, with the beam size fixed to one. We run the cost-bounded and utility-bounded attacks in the same configuration as before, and measure two metrics averaged over the models: Attack success, and attack success/time ratio.

Table 1 shows the results. On `IEEECIS`, the Universal Greedy outperforms the other choices in terms of success rate and the success/time ratio. On the `TwitterBot` dataset, it outperforms the other choices in the utility-bounded and unbounded attacks. For cost-bounded attacks, the Universal Greedy offers very close performance to the best option, the Basic Greedy.

Cost bound → / Scoring func. ↓	Adv. success, % 10	30	Gain	∞
UG	**45.32**	**56.57**	**56.22**	**68.20**
A*	42.37	55.62	55.34	53.47
PS	**45.32**	55.14	56.18	N/A
Basic Greedy	42.37	55.46	55.38	53.82

Cost bound → / Scoring func. ↓	Success/time ratio 10	30	Gain	∞
UG	**3.78**	**4.80**	**2.53**	**2.06**
A*	3.29	3.83	1.89	1.15
PS	**3.78**	4.01	2.26	N/A
Basic Greedy	3.21	3.86	2.01	1.16

(a) `IEEECIS`

Cost bound → / Scoring func. ↓	Adv. success, % 1,000	10,000	Gain	∞
UG	80.24	**85.35**	**21.63**	**87.00**
A*	77.56	84.45	20.29	86.25
PS	79.95	85.19	21.48	N/A
Basic Greedy	**80.40**	85.04	**21.63**	86.85

Cost bound → / Scoring func. ↓	Success/time ratio 1,000	10,000	Gain	∞
UG	208.95	205.76	**64.99**	**205.31**
A*	206.33	201.93	62.25	201.31
PS	205.85	203.18	63.76	N/A
Basic Greedy	**210.20**	**206.20**	64.32	204.96

(b) Twitter bot

Table 1: *Effect of the scoring-function choice* for graph-based attacks on `IEEECIS`. In all settings, our Universal Greedy scoring function offers the best success rate and performance.

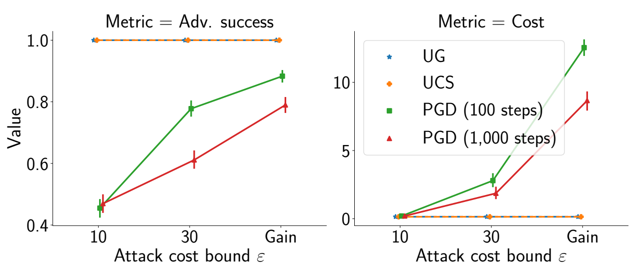


Figure 2: *Universal Greedy attack vs Baselines.* Left: Attack success rate (higher is better for the adversary). Right: Attack cost (lower is better for the adversary). For all cost bounds, our graph-based attack outperforms standard PGD and returns close to optimal-cost adversarial examples (obtained with Uniform-Cost Search, UCS).

C.2.2 Graph-based Attacks vs. Baselines

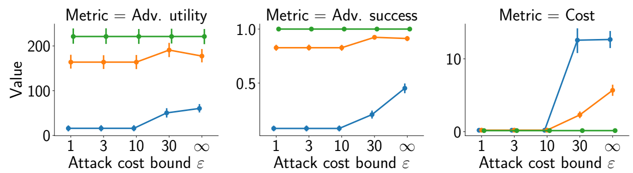
We compare the Universal Greedy (UG) algorithm against two baselines: previous work, and the minimal-cost adversarial examples.

Previous Work: PGD. Since the introduced cost model differs from the existing approaches to attacks on tabular data, we fundamentally cannot perform a fully apples-to-apples comparison against existing attacks (see Appendix A). To compare against the high-level ideas from prior work, we follow the spirit of the attack by Ballet et al. [31], which modifies the standard optimization problem from Eq. (1) to use correlation-based weights. We adapt the standard L_1-based PGD attack [10, 54] to (1) support categorical features through discretization, and (2) use weighted L_1 norm following our derivations in Appendix B.1. We provide a detailed description of this adaptation in Algorithm 4 in Appendix E.

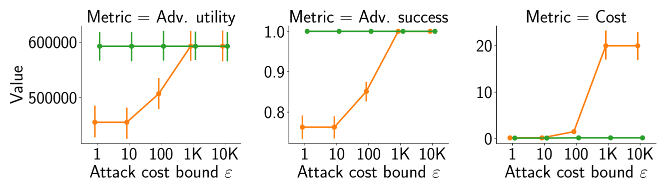
We run attacks using PGD with 100 and 1,000 steps, and compare it to UG (Appendix A) on the `TwitterBot` and `IEEECIS` datasets. As PGD can only operate on differentiable models, in this comparison we only evaluate the performance of the attacks against TabNet.

We run the cost-bounded attacks using two bounds ε, that are specific to each dataset (see Appendix E for the exact attack parameters). As before, we also run a utility-bounded attack at the breakeven margin $\tau = 0$. We measure the success rates of the attacks, as well as the average cost of the obtained adversarial examples. For conciseness, we do not report the results on `TwitterBot`, as they find they are equivalent to those on `IEEECIS`.

Fig. 2 shows that the UG attack consistently outperforms the PGD-based baseline both in terms of the success rate, and the costs. Our attacks are superior even when the PGD-based baseline produces feasible adversarial examples.



(a) IEEECIS. Model (test acc.): ● LR (0.62) ● XGBT (0.83) ● TabNet (0.77)



(b) HomeCredit. Model (test acc.): ● XGBT (0.65) ● TabNet (0.68)

Figure 3: Results of cost-bounded graph-based attacks against three types of models. Left pane: Adversarial utility (higher is better for the adversary). Middle and right panes: See Fig. 2. On IEEECIS, the attack can achieve utility from approximately \$10 to \$125 per attack depending on the target model. On HomeCredit, the average utility ranges between $\$400,000$ and $\$600,000$.

Minimal-Cost Adversarial Examples. As UG is a greedy algorithm, we additionally evaluate how far are the obtained adversarial examples from the optimal ones in terms of cost. For this, we compare the results from UG to a standard Uniform-Cost Search (UCS) [34]. UCS is an instantiation of the BFS framework (see Appendix A) with unbounded beam size, and the scoring function equal to the cost: $s(v,t) = c(v,t)$. In our setting, UCS is guaranteed to return optimal solutions to the following optimization problem:

$$\min_{x' \in \mathcal{F}(x,y)} c(x, x') \quad \text{s.t. } f(x') \neq y$$

Fig. 2 shows that UG has almost no overhead over the minimal-cost adversarial examples on TabNet ($1.03\times$ overhead on average). In fact, the average and median cost overhead is $1.80\times$ and $1\times$ over all models, respectively. There exist some outlier examples, however, with over $100\times$ cost overhead. We provide more information on the distribution of cost overhead in Appendix E.

C.2.3 Performance against Undefended Models

Having shown that the attacks outperform the baseline, and the design choices are sound, we demonstrate that the attacks bring some *utility* to the adversary. In this section, we evaluate the attacks in a non-strategic setting: the models are not deliberately defended against the attacks. For conciseness, we only evaluate cost-bounded attacks, as the next section provides an extensive demonstration of utility-bounded attacks.

In *all* evaluated settings, the attacks have non-zero success rates and achieve non-zero adversarial utility. Fig. 3 show the results of cost-bounded attacks for IEEECIS and HomeCredit datasets. For utility-bounded attacks, we present the results in Fig. 7 in the Appendix due to the space constraints. We omit the results for LR on HomeCredit as it does not perform better than the random baseline. An average adversarial example obtained using the cost-bounded objective brings a profit of \$125 to the adversary when attacking the IEEECIS TabNet model, and close to 100% of examples in the test data can be modified into successful adversarial examples.

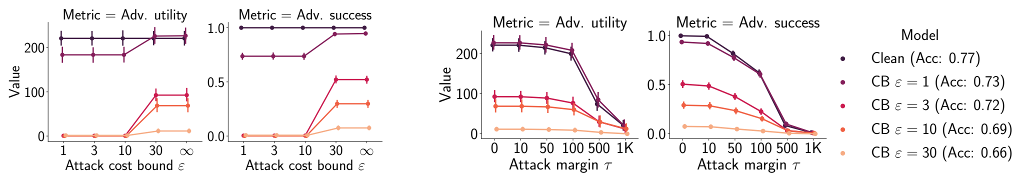
Although for all models we see non-zero success and utility, some models are less vulnerable than others—even without any protection. For example, the success rate of the adversary against LR on IEEECIS is much lower than against TabNet (at least 50p.p. lower). This model, however, is also comparatively inaccurate, with only 62% classification accuracy.

C.3 Evaluation of Our Defence Methods

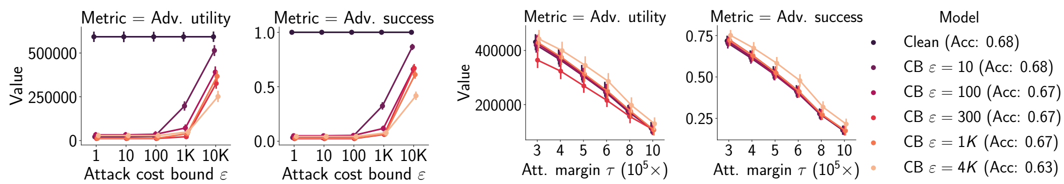
We evaluate the defence mechanisms proposed in Appendix B in two scenarios. First, a scenario in which the adversary's objective used by the defender for adversarial training—cost-bounded (CB) or

Table 2: Baseline performance

Accuracy	`TwitterBot`	`IEEECIS`	`HomeCredit`
Clean baseline	0.775	0.755	0.680
Robust baseline	0.773	0.685	0.556
Random baseline	0.566	0.500	0.501



(a) `IEEECIS`



(b) `HomeCredit`

Figure 4: *Cost-bounded adversarial training* for different adversarial budgets ε. Evaluation against cost-bounded (left), and utility-bounded (right) attacks. We represent the adversary's success and utility (y-axis) versus the adversary's attack budget ε or desired utility margin τ (x-axis). CB attacks only have substantial success and profit when the adversary invests more than the budget assumed by the defender. UB attacks are thwarted for `IEEECIS`, but CB training is not significantly effective on `HomeCredit`, and for some models even enables higher adversary's utility.

utility-bounded (UB)—matches the attack that will be deployed by the adversary. Second, a scenario in which the defender assumes the adversary's objective incorrectly and uses a different attack than the adversary when performing adversarial training.

Baselines. We set two comparison baselines which provide boundaries for which a defence can be considered effective.

On the accuracy side, we consider the *clean baseline*: a model trained without any defence. It provides the best accuracy, but also the least robustness against attacks. Any defence that does not achieve at least the clean baseline's *robustness* should not be considered, as the clean baseline would always provide better or equal accuracy, and hence a better robustness-accuracy trade-off.

On the robustness side, we consider the *robust baseline*: a model for which all features that can be changed by the adversary are masked with zeroes for training and testing. As this removes any adversarial input, this model is invulnerable to attacks within the assumed adversarial models. Any practical defence must outperform the robust baseline in terms of *accuracy*. Otherwise, the robust baseline would provide a better robustness-accuracy trade-off.

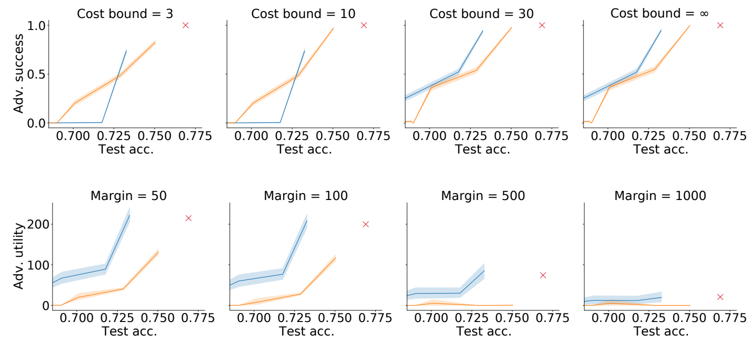
Table 2 shows the clean and robust baselines' accuracy for the three datasets. On `TwitterBot` the robust baseline performs almost as well as the clean model. As there is no space for a better defence for `TwitterBot`, we only evaluate our defences for the `IEEECIS` and `HomeCredit` models.

We train our attacks and defences using the parameters listed in Table 3 in Appendix E.

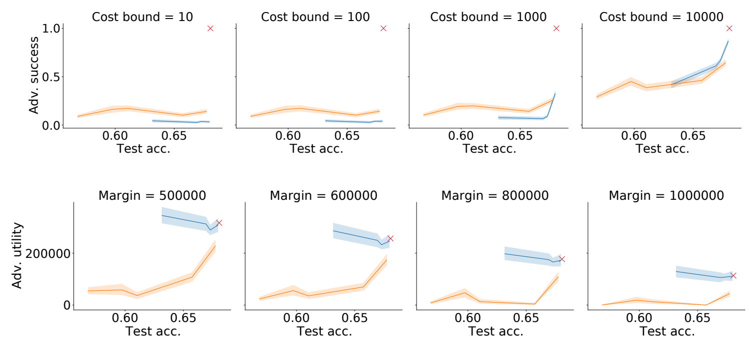
C.3.1 Defender matches the adversary

We first evaluate the case in which the adversarial training used to generate the defence is perfectly tailored to the adversary's objective.

Cost-bounded Defence vs. Cost-bounded Attack. We show in Fig. 4 the results when defender and adversary use CB objectives. For both `IEEECIS` and `HomeCredit` the CB trained defence



(a) IEEECIS: ● CB-trained models ● UB-trained models × Clean model



(b) HomeCredit: ● CB-trained models ● UB-trained models × Clean model

Figure 5: *Accuracy-robustness and utility-robustness trade-offs* for Cost-bounded and Utility-bounded adversarially trained models. The curves show accuracy (x-axis) and utility and success rate (x-axis) for the utility- and cost-bounded models presented in Fig. 4 and Fig. 1. When one curve is strictly below the other curve, it provides a better trade-off since it has better robustness for the same accuracy. Utility-bounded models consistently show better trade-offs for all utility-aware attacks. For CB attacks the situation is less consistent: for small cost-bounds CB defence outperforms utility-bounded one, while for the largest budgets utility bounded shows better results.

is effective when the adversary uses CB attacks: the adversary only finds successful adversarial examples with positive utility if they invest more than the budget assumed by the defender. If the defender greatly underestimates the adversary's budget of the adversary (e.g., training with $\varepsilon = 10$ when the adversary's budget is $\varepsilon = 1000$), the adversary obtains a high profit (close to 200K$). Therefore, an effective defence requires an adequate estimation of the adversary's capabilities.

Utility-bounded Defence vs. Utility-bounded Attack. Fig. 1 shows the results of our evaluation when the defender and the adversary use UB objectives. The defence is effective: it decreases both the success rate and the adversary's utility on both datasets. On IEEECIS, the adversary can only succeed when their desired profit τ is smaller than the τ used to train the defence. On HomeCredit, we observe a similar behaviour, although when training for margins τ less than 500K model does not completely mitigate adversaries that wish to have larger profits. When the defender allows for large adversary's profit margins (e.g., $\tau = 800K$ or $\tau = 1M$), the models become significantly robust with little accuracy loss.

C.3.2 Defender does not match the adversary

In the previous section, we show that if the defender correctly models the attacker's objective, our defences offer good robustness. Next, we evaluate the performance when the defender's model does not match the adversary's objective. This is likely in realistic deployments, as the defender might not have any a priori knowledge of the adversary's objective.

Utility-bounded Defence vs. Cost-bounded Attack. We show in Fig. 1 our evaluation results when a CB adversary attacks a defence trained assuming UB objectives. For both datasets, the robustness improves with respect to the clean baseline, even though robustness against CB adversaries is not the defence goal. The improvement is more pronounced as the defender tightens the profit margin (decrease in τ, being this effect much stronger on `HomeCredit` where even loose profit margins provide significant robustness. The adversary can increase their success by increasing their budget ϵ. Increasing the budget also increases the utility in `HomeCredit`. These experiments show that UB training improves robustness *even when the adversary has a different objective*.

Cost-bounded Defence vs. Utility-bounded Attack. When we confront a UB adversary against a CB defence, we observe a different behaviour (see Fig. 4). On `IEEECIS` CB adversarial training increases the robustness of the model against utility-oriented adversaries—with greater effect as the cost bound increases. However, when protecting against high adversary's budgets ($\varepsilon = 30$) the impact on accuracy is too large and the robust baseline becomes preferable.

For `HomeCredit` the situation is worse. While performance is always above the robust baseline, we observe little improvement with respect to the clean model. Even worse, for certain parameters the utility of the adversary can even increase after the adversarial training (see the model trained with a bound of $\varepsilon = 4000$). We conclude that CB training might offer no guarantees if the adversary has a different objective.

C.3.3 Robustness-Accuracy Trade-offs

In the previous sections, we evaluated the effectiveness of the defences depending on the adversary's and defender's objectives. We now evaluate the trade-offs between defence effectiveness in reducing the adversary's success and the utility of the attacks, and the accuracy of the model.

As adversarial training penalizes changes in the model's output caused by input feature perturbations, it results in certain features having less influence on the output. These features cannot be used for prediction to the same extent as features in the clean baseline, which leads to the degradation of the model's accuracy. On the positive side, these features can neither be used by the adversary—the robust baseline being the extreme in which all features prone to manipulation are zeroed—reducing the attack's success and utility.

We show in Fig. 5 the trade-off between adversarial success and utility on the one side, and model accuracy on the other side for `IEEECIS` (top) and `HomeCredit` (bottom) for all combinations of the defender and adversarial objectives. For CB adversaries (top row for each dataset), it is not clear which defence type is superior. Which defence provides better robustness for a given accuracy depends on the adversary's budget. On the contrary, for utility-bounded adversaries (bottom row for each datasets), we consistently observe better robustness (less adversarial utility for the same accuracy) for the utility-bounded defence compared to the cost-bounded. We conclude that in the absence of knowledge of the adversary's objective, utility-bounded defences are preferable. They outperform CB adversarial training when the adversary is utility-oriented, and offer comparable performance against CB attacks.

D Details on the Projection Algorithm and Adversarial Training

In this appendix, we describe our modifications to the traditional adversarial training pipeline.

D.1 Adversarial Training Procedure

Our training procedure is a version of the well-known adversarial training algorithm based on the PGD method [10].

For every sample in a batch $(\phi(x^{(i)}), y^{(i)})_{i=1}^{b}$, we generate adversarial examples (lines 2–7) by finding the perturbation $\delta^{(i)}$ (lines 4–7). The perturbations are normalized and multiplied by $\alpha = {}^{2\varepsilon}/_{n}$, to improve the stability of the algorithm (line 6); and then projected to fulfill our relaxed problem in Eq. (16) (line 7). We update the model weights (line 8), and return θ'.

Algorithm 2 Cost-bounded Adversarial Training Algorithm (single iteration)

Input: Model weights θ, batch of training examples $(\phi(x^{(i)}), y^{(i)})_{i=1}^{b}$, per-feature costs w_i, cost bound ε.
Output: Updated weights θ'

$\alpha := 2\frac{\varepsilon}{n}$
for i **in** $1..b$ **do**
    $\delta^{(i)} := 0$
    for t **in** $1..n$ **do**
        $\nabla^{(i)} := \nabla_{\delta_i} \ell(f_\theta(\phi(x^{(i)}) + \delta^{(i)}), y_i)$
        $\delta^{(i)} := \delta^{(i)} + \alpha \frac{\nabla^{(i)}}{\|\nabla^{(i)}\|_1}$
        $\delta^{(i)} := P_{w,\varepsilon}(\phi(x^{(i)}) + \delta^{(i)})$
$\theta' := \theta - \eta \nabla_\theta \frac{1}{b} \sum_{i=1}^{b} \ell(f_\theta(\phi(x^{(i)}) + \delta^{(i)}), y^{(i)})$
Return θ_{new}

D.2 Projection algorithm

We design an adapted projection algorithm to solve Eq. (14), presented in Algorithm 3. This algorithm is an extension of an existing sort-based weighted L_1 projection algorithm [46, 47]. It takes as input a sample $\overline{x}$ and a perturbed sample $\overline{x}'$, and returns a valid perturbation vector δ such that $\overline{x} + \delta$ lies within the cost budget. With respect to the algorithm by Perez et al. [47], we introduce the capability to assign different weights based on a feature type and perturbation sign (line 2, in blue) to support our cost function in Eq. (13).

We now prove the correctness of this algorithm.

Statement 1. *Algorithm 3 is a valid projection algorithm onto the set $\tilde{B}_\varepsilon$, as defined in Eq.* (15). *Concretely, for a given $\overline{x}, \overline{x}'$, the algorithm returns δ^* such that:*

$$\delta^* = P_{\tilde{B}_\varepsilon(x,y)}(\overline{x}') \triangleq \underset{\delta \in \mathbb{R}^m}{\arg\min} \; \|\overline{x}' - (\overline{x} + \delta)\|_2$$
$$\textit{s.t. } \overline{c}(\overline{x}, \overline{x} + \delta) \leq \varepsilon$$

Proof of Statement 1. First, if we keep either $c_{j+}(x)$ or $c_{j-}(x)$, the constraint becomes a weighted L_1 constraint, for which the complete proof is given by Perez et al. [47]. Then, we can recall the property that projection onto the weighted L_1 ball is equivalent to projection onto the simplex, if $\overline{c}(\overline{x}, \overline{x}') > \varepsilon$, and prove the similar property here.

Lemma 1. *For any $\overline{x}, \overline{x}', \varepsilon$,*

$$\delta^* = \underset{\delta:\, \overline{c}(\overline{x}, \overline{x}+\delta) \leq \varepsilon}{\arg\min} \|\overline{x}' - \overline{x} - \delta\|_2$$

$$\forall i \in [1..n] \implies \textit{sign}(\delta_i) = \textit{sign}(\overline{x}'_i - \overline{x}_i) \textbf{ or } 0$$

Proof. Proof by contradiction. Let us assume that the lemma does not hold and $\exists i : \text{sign}(\delta_i) = -\text{sign}(\overline{x}'_i - \overline{x}_i)$ and $\text{sign}(\delta_i) \neq 0$. Then, we can construct $\delta^* : \forall j \neq i, \delta^*_j = \delta_j$ and $\delta^*_i = -\delta_i$.

$$\|\overline{x}' - \overline{x} - \delta\|_2^2 = \|\overline{x}' - \overline{x} - \delta^*\|_2^2 - (\overline{x}'_i - \overline{x}_i - \delta_i)^2 + (\overline{x}'_i - \overline{x}_i - \delta^*_i)^2$$

Since $\text{sign}(\delta_i) = -\text{sign}(\overline{x}'_i - \overline{x}_i)$ and $\text{sign}(\delta_i) \neq 0$,

$$(\overline{x}'_i - \overline{x}_i - \delta_i)^2 > (\overline{x}'_i - \overline{x}_i - \delta^*_i)^2$$

Therefore,

$$\|\overline{x}' - \overline{x} - \delta^*\|_2^2 < \|\overline{x}' - \overline{x} - \delta\|_2^2$$

Which is a contradiction to the original statement. □

Algorithm 3 Cost-Ball Projection Algorithm

Input $\overline{x}, \overline{x}', c, \varepsilon, \mathcal{C}, \mathcal{I}$
Output $\delta^* = P_{\tilde{B}_\varepsilon(x,y)}(\overline{x}')$

$\delta = \overline{x}' - \overline{x}$

$w_i := \begin{cases} \min_{t \in \mathcal{F}_i(x,y)} c_i(x_i, t), & \text{if } i \in \mathcal{C} \\ c_{j-}(x), & \text{if } i \in \mathcal{I} \text{ and } \delta_i < 0 \\ c_{j+}(x), & \text{if } i \in \mathcal{I} \text{ and } \delta_i \geq 0 \end{cases}$

$z_i := \frac{\delta_i}{w_i}$

$\pi_z() :=$ Permutation $\uparrow (z)$

$z_i := z_{\pi_z(i)}$

$J := \max \left\{ j : \frac{-\varepsilon + \sum_{i=j+1}^{m} w_{\pi_z(i)} \delta_{\pi_z(i)}}{\sum_{i=j+1}^{m} w_{\pi_z(i)}^2} > z_j \right\}$

$\lambda := \frac{-\varepsilon + \sum_{j=J+1}^{m} w_{\pi_z(j)} \delta_{\pi_z(j)}}{\sum_{j=J+1}^{m} w_{\pi_z(j)}^2}$

$\delta_i^* :=$ sign$(\delta_i) \max(\delta_i - w_i \lambda, 0)$

Return δ_i^*

The highlighted parts indicate the differences with respect to the sort-based weighted L_1 projection algorithm [47]. The function $\pi_z(i)$ denotes an outcome of permutation. Permutation $\uparrow (z)$ is a sort permutation in an ascending order.

Algorithm 4 PGD-Based Attack

Input: P, initial example x, label y, costs w, cost bound ε.
Output: Adversarial example x^*

$\alpha := 2\frac{\varepsilon}{n}$

$\delta := 0$

for j **in** $1..n$ **do**

  $\nabla := \nabla_\delta \ell(f_\theta(\phi(x) + \delta), y)$

  $\delta := \delta + \alpha \frac{\nabla}{\|\nabla\|_1}$

  $\delta := P_{\mathcal{B}_{w,\varepsilon}}(\delta)$

$x^* = P_{\mathcal{F}}(\delta)$

Return x^*

Based on this lemma we can see that, to find the projection of $\overline{x}'$, we can replace $\overline{c}(\overline{x}, \overline{x}')$ with the following experssion:

$$\overline{c}^*(\overline{x}, \overline{x}') = \sum_{i \in \mathcal{C}} \frac{1}{2} \|\bar{x}_i - \bar{x}_i'\|_1 \min_{t \in \mathcal{F}_i(x,y)} c_i(x_i, x_i') + \sum_{j \in \mathcal{I}} c_{j*}(x) \cdot |\bar{x}_j' - \bar{x}_j|,$$

where c_{j*} is defined as follows:

$$c_{j*}(x) = \begin{cases} c_{j+}(x), & \text{if sign}(\overline{x}_j' - \overline{x}_j) \geq 0 \\ c_{j-}(x), & \text{if sign}(\overline{x}_j' - \overline{x}_j) < 0 \end{cases}$$

We can do so as both of these functions attain the same minimum value. $\square$

E Additional Experimental Details

In this appendix we provide the details of our evaluation aiming to improve the reproducibility of our results.

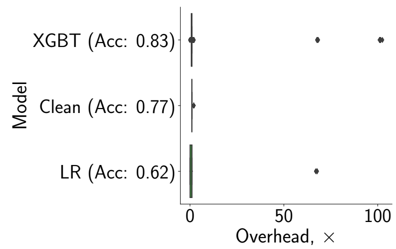


Figure 6: The distribution of cost overhead of adversarial examples obtained with UG over minimal-cost adversarial examples obtained with UCS on `IEEECIS`. Most UG adversarial examples have cost close to the minimal, although there exist outliers.

Table 3: `IEEECIS` and `HomeCredit` attack and defence parameters

Parameter	**Value range**
Adversarial Training (`IEEECIS`)	
Batch size	2048
Number of epochs	400
PGD iteration number	20
TabNet hyperparameters	$N_D = 16, N_A = 16, N_{steps} = 4$
ε *(for CB models)*	$[1, 3, 10, 30]$
τ *(for UB models)*	$[0, 10, 20, 50, 100, 200, 500]$
Attacks (`IEEECIS`)	
Max. iterations	100K
ε *(for CB attacks)*	$[1, 3, 10, 30]$
τ *(for UB attacks)*	$[0, 10, 50, 500, 1000]$
Adversarial Training (`HomeCredit`)	
Batch size	2048
Num. of epochs	100
TabNet hyperparameters	$N_D = 16, N_A = 16, N_{steps} = 4$
Num. of PGD iterations	20
ε *(for CB models)*	$[1, 10, 100, 1000, 10000]$
τ *(for UB models)*	$[300K, 400K, 500K, 600K, 800K]$
Attacks (`HomeCredit`)	
Num. of iterations	100
ε *(for CB attacks)*	$[1, 10, 100, 1K, 10K]$
τ *(for UB attacks)*	$[10K, 300K, 400K, 500K, 600K, 800K]$

E.1 Hyperparameter selection

We list our defence and attack parameters in Table 3. TabNet parameters are denoted according to the original paper [52]. We set the virtual batch size to 512. As training the clean baseline for HomeCredit was prone to overfitting, we reduced the training number of epochs to 100. Other hyperparameters were selected with a grid search.

E.2 Dataset Processing and Cost Models

E.2.1 `TwitterBot`

We use 19 numeric features from this dataset. We dropped 3 features, for which we cannot compute the effect of a transformation as we do not have access to the original tweets. We use the number of followers as the adversary's gain. We assign costs of features based on estimated costs to purchase Twitter accounts of different characteristics on darknet markets.

E.2.2 IEEECIS

We ascribe cost of changes, assuming that the adversary can change the device type and email address with a small cost. The device type can be changed with low effort using specific software on a mobile phone. Email domain can be changed with a registration of a new email address which typically cannot be automated. Although also low cost, it takes more time and effort than changing the device time. We reflect these assumptions ascribing the costs \$0.1 and \$0.2 to these changes. Changing the type of card requires obtaining a new card, which costs approximately \$20 in US-based darknet marketplaces in 2022, according to our research. We consider the transaction amount as a gain obtained by an adversary.

E.2.3 HomeCredit

The main goal of the adversary in this task is receiving a credit approval, therefore, illustrative purposes, we set credit amount to be a gain of one sample. All features which can be used by an adversary are listed in Table 6 with the costs we ascribe to them. We assumed six groups of features and estimated the cost as follows:

- *Group 1*: features that an adversary can change with negligible effort such as email address, weekday or hour of the application. We ascribe \$0.1 cost to these transformations.
- *Group 2*: features associated to income. We use these as a numerical features to illustrate the flexibility of our method. We assume that to increase income by 1\$, the adversary needs to pay 1\$.
- *Group 3*: features associated to changing a phone number. Based on the US darknet marketplace prices we estimate that buying a phone number costs \$10.
- *Group 4*: features related documents which can be temporally changed in favor of an adversary. For example, a car can be transferred from one person to another for the application period and returned to the original owner after it. We ascribe a cost of \$100 to obtain these documents.
- *Group 5*: features that requires either document forging or permanent changes to a person's status. For instance, buying a university diploma. These are expensive changes, and we estimate their cost in \$1 000.
- *Group 6*: features related to credit scores provided by unspecified external credit-scoring agencies. We estimate the cost of changes in this group with a manipulation model presented below.

Credit-score manipulation. In our feature set we include the features that contain credit scores from unspecified external credit-scoring agencies. One reported way of manipulating such credit scores is using credit piggybacking [51]. During piggybacking, a rating buyer finds a "donor" willing to share a credit for a certain fee. We introduce a model that captures costs of manipulating a credit score through piggybacking.

We assume that after one piggybacking manipulation the rating is averaged between "donor" and recipient, and that "donors" have the maximum rating (1.0). Then, the cost associated to increasing the rating from 0.5 to 0.75 is the same as that of increasing from 0.9 to 0.95. This cost cannot be represented by a linear function. Let the initial score value be x. The updated credit score after piggybacking is $x' = {}^{(x+1)}/_{2}$. If we repeat the operation n times, the score becomes:

$$x' = \frac{x + 2^n - 1}{2^n}$$

Thus, the number of required piggybacking operations can be computed from the desired final score x' as $n = \log_2 \frac{1-x}{1-x'}$, and the total cost is $c(x, x') = nC$, where C is the cost of one operation. We estimate to be \$10,000.

$$c(x, x') = C \log_2 \frac{1-x}{1-x'} = C(\log_2(1-x) - \log_2(1-x'))$$

To represent this cost function for adversarial training, we can use the encoding described in Appendix B.1, setting $\phi(x) = \log_2(1-x)$. Then, the cost function becomes $\bar{c}(x, x') = C|\phi(x) - \phi(x')|$, which is suitable for our defence algorithm. It is worth mentioning that this cost is a lower bound of

Table 4: Costs of changing a feature in `TwitterBot` dataset

Feature	Estimated cost, $
likes_per_tweet	0.025
retweets_per_tweet	0.025
user_tweeted	2
user_replied	2

Table 5: Costs of changing a feature in `IEEECIS` dataset

Feature	Estimated cost, $
DeviceType	0.1
P_emaildomain	0.2
card_type	20

Table 6: Costs of changing a feature in `HomeCredit`

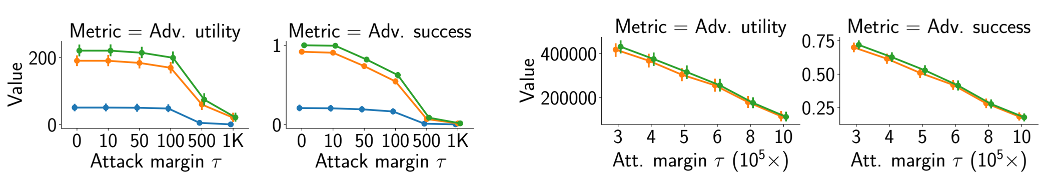
Feature	Estimated cost, $
NAME_CONTRACT_TYPE	0.1
NAME_TYPE_SUITE	0.1
FLAG_EMAIL	0.1
WEEKDAY_APPR_PROCESS_START	0.1
HOUR_APPR_PROCESS_START	0.1
AMT_INCOME_TOTAL	1
FLAG_EMP_PHONE	10
FLAG_WORK_PHONE	10
FLAG_CONT_MOBILE	10
FLAG_MOBIL	10
FLAG_OWN_CAR	100
FLAG_OWN_REALTY	100
REG_REGION_NOT_LIVE_REGION	100
REG_REGION_NOT_WORK_REGION	100
LIVE_REGION_NOT_WORK_REGION	100
REG_CITY_NOT_LIVE_CITY	100
REG_CITY_NOT_WORK_CITY	100
LIVE_CITY_NOT_WORK_CITY	100
NAME_INCOME_TYPE	100
CLUSTER_DAYS_EMPLOYED	100
NAME_HOUSING_TYPE	100
OCCUPATION_TYPE	100
ORGANIZATION_TYPE	100
NAME_EDUCATION_TYPE	1000
NAME_FAMILY_STATUS	1000
HAS_CHILDREN	1000

the real cost, as the adversary can only do an integer number of operations. Nonetheless, it perfectly fits our framework as Eq. (11) encompasses this cost model. This is not a fully realistic model, as we cannot know how exactly credit score agencies compute the rating. However, it is reasonable, and enables us to demonstrate how our framework's support of non-linear costs.

Cost bound → / Beam size ↓	Adv. success, % 10	30	Gain	∞
1	45.32	**56.57**	**56.22**	**68.20**
10	45.32	56.01	55.65	56.01
100	45.32	56.53	56.18	56.53

Cost bound → / Beam size ↓	Success/time ratio 10	30	Gain	∞
1	**3.78**	**4.80**	**2.53**	**2.06**
10	2.14	2.25	1.31	1.15
100	0.66	0.65	0.65	0.66

Table 7: Effect of beam size B in the Universal Greedy algorithm on the `IEEECIS` dataset. The success rates are close for all choices of the beam size, thus the beam size of one offers the best performance in terms of runtime.



(a) `IEEECIS`. Model (test acc.):
• LR (0.62) • XGBT (0.83) • TabNet (0.77)

(b) `HomeCredit`. Model (test acc.):
• XGBT (0.65) • TabNet (0.68)

Figure 7: Results of utility-bounded graph-based attacks against three types of models. Left pane: Adversarial utility (higher is better for the adversary). Right pane: See Fig. 2. On `IEEECIS`, the attack can achieve utility from approximately up to approximately \$200 per attack against TabNet and XGBT. On `HomeCredit`, the average utility ranges between \$400, 000 and \$200, 000.