UNIVERSITÉ DU LUXEMBOURG

PhD-FSTM-2024-076

The Faculty of Sciences, Technology and Medecine

Dissertation

Defence held on October 4^{th}, 2024 in Luxembourg

to obtain the degree of

Docteur de l'Université du Luxembourg en Informatique

by

Thibault Simonetto

Born on 9^{th} April 1997 in Metz (France)

Enhancing Machine Learning Robustness for Critical Industrial Systems:

Constrained Adversarial Attacks and Distribution Drift Solutions

Dissertation defence committee

Dr. Maxime Cordy, Dissertation Supervisor
Research Scientist, University of Luxembourg, Luxembourg, Luxembourg

Dr. Yves Le Traon, Chairman
Professor, University of Luxembourg, Luxembourg, Luxembourg

Dr. Tegawendé Bissyandé, Vice Chairman
Associate Professor, University of Luxembourg, Luxembourg, Luxembourg

Dr. Damien Teney, Member & Reviewer
Research Scientist, Idiap Research Institute, Martigny, Switzerland

Dr. Salah Ghamizi, Member & Reviewer
Research Associate, Luxembourg Institute of Science and Technology, Esch-sur-Alzette, Luxembourg

Abstract

Model robustness weaknesses significantly hinder the adoption of ML-enhanced systems in critical real-world contexts. Adversarial examples and the natural decay of model performance over time (known as distribution drift) are major challenges. Current methods of evaluation do not sufficiently take into account the real context and can lead to misleading evaluations. While methods to evaluate robustness against adversarial examples in Computer Vision (CV) and Natural Language Processing (NLP) are well-studied, their application to tabular data remains scarcely explored. These methods often produce unrealistic feature vectors that do not represent feasible domain objects. Additionally, existing approaches to mitigate performance decay often overlook practical issues like labeling and deployment delays, leading to an overestimation of their effectiveness. In collaboration with BGL BNP Paribas, we address these challenges by demonstrating the importance of realistic robustness evaluation for ML models in critical contexts. The contributions of the thesis are as follows. We improve the evaluation of critical industrial systems by proposing attacks that generate more realistic adversarial examples. Specifically, we propose three algorithms that produce adversarial examples that satisfy domain constraints (a necessary condition for the example to occur in reality). We develop new defenses to robustify models against such adversarial examples by combining synthetic data generators with adversarial training (a process in which adversarial examples are integrated into the training). These defenses together with the realistic evaluation of models' robustness form TabularBench, a benchmark providing 200+ models across 5 datasets trained with 14 different training methods. We are confident that our benchmark will accelerate the research of adversarial defenses for tabular ML. We apply our contributions to a real-world use case from BGL BNP Paribas and show how our defenses can robustify critical ML systems. As a final contribution of this thesis, we consider the robustness of critical systems to distribution drifts that produce performance decay. Inspired by a real-world use case from BGL BNP Paribas, we reassess model retraining strategies by considering industrial constraints such as labeling and deployment delays. Using a realistic protocol, we benchmark 17 retraining strategies and find that ignoring delays overestimates effectiveness and alters method rankings. The optimal drift detection

method without delay does not remain optimal if delays occur, highlighting the necessity for realistic evaluation protocols. Overall, the objective of this dissertation is to bridge gaps between the current assumption of evaluating and robustifying ML models in the literature and the reality of ML models deployed in critical industrial systems.

Acknowledgments

Throughout this thesis, I have received a great deal of support and assistance, and I am probably forgetting many in the following. First of all, I am deeply grateful to Dr. Maxime Cordy for the opportunity he gave me to pursue my PhD studies under his supervision. He provided me with valuable feedback and I have learned a lot from him. I am grateful for the opportunities to teach, conduct my thesis with an industrial partner, and develop product-oriented projects. I am also thankful for the trust he put in me in conducting projects beyond pure research.

I would like to thank my industrial partner BGL BNP Paribas and my colleagues in the Data Science lab for their interesting topics, use cases, and the trust they put in me to conduct research on their data and systems. I have learned a lot during this partnership, and it allowed me to base my research on real-world applications.

I am also thankful to my co-authors for their efforts and feedback that contributed to making this thesis stronger. I also want to thank all jury members for their interest in my research and the time invested in my dissertation.

I would like to express my gratitude to all my colleagues from the SerVal research group for the great motivation they were during the tougher periods of my thesis.

More personally, I thank my partner Natia Japoshvili, for her love and the incredible support she gave me during these last four years. I also thank my father, my mother, and my brother for the support they gave me throughout my studies.

Contents

1

Introduction

With Machine Learning (ML)-based systems becoming the preferred means of automating the processing of large amounts of data, the question of their robustness is of utmost importance. Yet, testing the robustness of such ML-based systems still poses challenges to practitioners. After introducing existing approaches to evaluate and improve the robustness of ML models, we introduce the challenges that appear when applying these methods to critical industrial systems. Finally, we present the three contributions of this dissertation and how they tackle the identified challenges.

Contents

1.1 Context

Machine Learning (ML), a subset of Artificial Intelligence, represents one of the most significant technological advancements of the last two decades enabling computers to learn from data without being explicitly programmed and driving innovation across various industries, such as finance [GCG+20], medical [RIZ+17] or cybersecurity [PPC+20]. The rapid adoption of ML can be attributed to several key factors. On the ML model's developer side, the availability of large amounts of data, the advancements in computing power lowering its cost, and the development of sophisticated algorithms enabled by open source framework have facilitated the industrialization of ML models. On the consumer side, recent development of generative models (e.g. ChatGPT) and their accessibility have contributed to the popularity of ML.

The industrial partner of this dissertation, BGL BNP Paribas Luxembourg has leveraged recent advances in Machine Learning to improve their daily operation. Specifically, BGL employs ML algorithms to identify and quarantine suspicious transactions for manual inspection. This system has proven effective, accurately classifying up to 80% of transactions, thereby minimizing human effort and reducing risks for the bank. However, obstacles related to the technology continue to pose difficulties for the spread of ML technologies both within the bank and in other areas. Critical industrial systems are software and hardware systems that are essential for the safe and reliable operation of industrial processes. The failure of these systems can result in catastrophic consequences, including loss of life, significant environmental damage, and substantial financial loss. A challenging issue that critical industrial systems face is the robustness of their ML models. We broadly define the robustness of ML systems as the degree to which a model's performance changes when confronted to a data distribution unseen during training. The extreme case of distribution change between training and production is caused by adversarial examples. Adversarial examples are carefully altered original examples that preserve the semantics for a human observer but cause the model to output wrong decisions. Another challenge of ML robustness is distribution drift, which are unforeseeable change in the testing distribution of examples over time. For instance, in the financial industry, the recent Covid-19 crisis has reshaped the economic landscape and therefore can cause distribution drift in ML financial systems. In this dissertation, we study these two phenomena in the context of critical industrial systems.

To enable the secure deployment of ML models in critical systems it is important to evaluate their robustness to adversarial examples. The standard approach to assess the robustness of a model is known as adversarial testing. To conduct adversarial testing, we use an adversarial attack to generate adversarial examples from the testing set and evaluate the accuracy of the model over the generated

adversarial examples. Adversarial attacks operate over the problem space or the feature space. Attacks operating in the problem space manipulate real-world objects such as malware [CSD19], which are then numerically represented in the feature space before being fed to the ML models. Such attacks usually suffer from long execution times [DGS+22] and require important engineering effort to adapt to different domains. Attacks operating over the feature space directly manipulate the feature representation of real-world objects, such as the pixel values of images in computer vision [GSS15; SZS+13a]. Recent studies [GCG+20; TWT+20] have shown that a major limitation of feature space attacks is the lack of realism of the examples they generate. Often, the resulting samples do not correspond to real-world objects. Although the generated example successfully evades the ML model in controlled experiments, they would not be feasible in industrial settings and therefore are not valid proxy to evaluate the robustness of industrial systems. The lack of realisticness arises from attacks perturbing examples without considering domain constraints. In the problem space, domain constraints define what are valid objects. For example, for BGL, a valid transaction must have a positive amount. This problem space constraints directly translate to the feature space. In the feature space, constraints arise as a direct translation of problem space constraints or as a result of the feature engineering process. For example, financial systems feature engineering involves computing statistical values on the history of transactions over a period of time. In a valid feature space example, the feature representing the minimum transaction amount should have a lower value than the feature representing the average transaction amount. Adversarial attacks usually only consider boundary constraints on the feature values and not the whole spectrum of domain constraints, hence they fail to generate valid adversarial examples.

Another important consideration for deploying reliable ML models in critical systems is their robustness to distribution drift, particularly, drift that causes a decline in the model's performance over time. Our industrial partner has observed such a distribution drift and estimates, that when left unsupervised, distribution drift causes such degradation of performance that a complete reengineering of the system is necessary every year. Multiple solutions to the problem of distribution drift have been proposed in the literature. The first family of methods involves retraining the model periodically (baseline) or when a drift is observed in the distribution. Methods have been proposed to detect drift in the input distribution [KVD+14; QAW+15; Inc22; VKV+22] or in the error rate of the models [Pag54; GMC+04; BCF+06; BG07a; FCR+15; RHS20]. Literature on data drift detectors often evaluates the capacity of the detector to identify drift in the data distribution but not how this detection method can help when integrated into an ML retraining schedule. On the other hand, the error drift detector evaluation protocol supposes that labels are available for retraining and that the model is directly available

after training. On the contrary, for our industrial partner, there is a delay in labeling incoming samples. Additionally, developed ML models must undergo a (partly) manual validation phase which also introduces delays. Another approach to managing distribution drift is online training introduced in [WK96]. Online training involves updating a model's parameters - and consequently its decision-making - iteratively each time a new example is introduced into the system. However, this approach is unsuitable for critical industrial systems, as it conflicts with the requirement to thoroughly test a system's robustness before deploying it in production.

1.2 Challenges in Evaluating and Building Robust ML systems

Adversarial attacks and distribution drift are two major threats to the robustness and therefore adoption of ML systems in critical domains. In computer vision and natural language processing, methods have been proposed to evaluate and robustify ML models against these two challenges. However, the applicability of these methods to the particularities of tabular data, often used in critical industrial systems remains scarcely explored. In particular, the following challenges arise when applying robustness evaluation and enhancement to real-world critical systems such as the one of BGL BNP Paribas.

1. **Realistic robustness evaluation of critical systems against adversarial examples.**

   As aforementioned, adversarial testing is a process where the model developer generates adversarial examples using the strongest adversarial attack available and measures the accuracy of the model under test for these examples. The challenge lies in the realisticness of the examples produced by the attack. In image classification, the domain of definition is straightforward: if all elements of the input vector (each corresponding to a pixel) are in the interval $[0, 255]$, the vector represents a valid image. However, critical industrial systems are often based on tabular data, for instance in finance [GCG+20] and cybersecurity [CO19a]. With tabular data, the input vector is composed of features and represents a domain object, such as a history of transactions. The transformation from the domain object space to the feature space naturally creates constraints in the feature space that must be satisfied for the examples to be valid and realistic. For example, in a transaction system, the feature *average amount in the last ten days* must be lower or equal to the feature *maximum amount in the last ten days*. To correctly assess the robustness of the machine learning model, one must consider such constraints in the adversarial evaluation and generation process. During robustness evaluation,

we can automatically reject adversarial examples that do not respect domain constraints. However, using traditional attacks that are not specifically designed to respect these constraints, all examples will be rejected leading to a false sense of robustness. This limitation calls for adversarial attacks specifically designed to handle domain constraints. Although constrained attacks and attacks in the problem space have been proposed, these attacks require tremendous engineering effort to apply to other use cases.

2. **Effective training methods to robustify ML systems.**

   A large variety of methods have been proposed to defend against adversarial examples. However, adversarial training [MMS+17] based defenses are recognized as the only reliable defenses against evasion attack [TCB+20; Car23]. Tramèr et al. [TCB+20] demonstrate that many recently published defenses, although evaluated on existing adaptive attacks, could still be broken by new adaptive attacks. More recently Carlini [Car23] showed the simplicity of creating such attacks by letting a large language model propose the attack. Adversarial training consists of adding adversarial examples during training to empirically robustify the models against such examples. In addition to adversarial training, data augmentation methods in combination with adversarial training have been shown effective in robustifying models [RGC+21]. However, these techniques were mostly applied to computer vision and little is known about their generalization to tabular data in critical industrial systems.

3. **Realistic evaluation of performance drift mitigation of ML systems in production.** The evaluation process of drift mitigation techniques also lacks realisticness. Earlier we determined that the only applicable method to mitigate distribution drift in ML systems is a schedule of retraining, periodic or based on drift detection. However, there remains a gap between the process of evaluation of drift detectors in the literature and the reality of the industrial process. The evaluation protocol of the literature often does not consider the entire lifecycle of the ML model. In particular, the data acquisition process is often optimistic and a label for incoming data is available shortly after an example is observed. Additionally, the validation process, which remains partially manual introduces an additional delay. We argue that for industrial systems, the most important challenge is not the lack of effectiveness of existing technical solutions, but in the realisticness of existing evaluation procedures. We hypothesize that this procedural gap leads to an overestimation of the capabilities of drift detectors to mitigate distribution drift that impacts performance drift.

1.3 Overview of contributions

This section presents the contributions of this dissertation to address the aforementioned challenges related to the realistic evaluation of the robustness of ML models deployed in critical environments.

- **A constrained adaptive attack to evaluate the robustness of critical industrial ML systems in the constrained feature space**.
  In Chapter 4, we propose a theoretical framework to address the challenge of realistically evaluating the robustness of ML systems in the constrained feature space. We instantiate our framework with two attacks: CPGD, a gradient-based attack inspired by PGD[MMS+17] using the penalty function, and MOEVA, a multi-objective genetic algorithm whose objective is to produce a misclassification while minimizing the penalty function. MOEVA is the first attack to reliably generate adversarial examples in the constrained feature space across various domains. Our results reveal that only MOEVA is effective across all our four use cases and that simply adding the penalty function to PGD is not sufficient to create a strong gradient-based attack. In Chapter 5 we propose Constrained Adaptive PGD (CAPGD), an effective gradient-based attack that overcomes the limitation of CPGD, notably by integrating recent advances from computer vision[CH20] such as step size adaptation and perturbation momentum, as well as a generic repair operator for equality constraints. After showing the complementarity of our approach, we propose CAA, the successive application of CAPGD and MOEVA, that further increases the success rate the robustness while being up to five times faster than MOEVA.
- **A collection of training methods to robustify models against adversarial attacks in tabular deep learning.** After building a strong attack to evaluate the robustness of deep learning models to adversarial examples, we focus on the robustification of these models (Chapter 6). We evaluate different training methods to robustify models. We first evaluate Madry adversarial training [MMS+17] approach to learn from adversarial examples, hence increasing the robustness of our models. We found that adversarial training is not sufficient to robustify a model. In the domain of computer vision, Rebuffi et al. [RGC+21] showed that training on synthetic data can also improve robustness when combined with adversarial training. Inspired by this line of work, we adapt six synthetic data generators to be used with adversarial training. Our empirical study reveals that combining adversarial training with data augmentation increases robustness, even under a large attack budget. Additionally, we validated our main claims by conducting an empirical study on the industrial use case of our partner. We thereby confirm that CAA reliably evaluates the robustness of ML models under

domain constraints and that adversarial training with data augmentation increases the robustness of the models. In practice, our method increases the robustness of their transaction system from 43.3% accuracy to 75.5% while maintaining similar clean performance (respectively MCC of 0.418 and 0.415). Building on these attacks and defenses constitutions, we introduce TabularBench (Chapter 7), the first benchmark specifically designed for evaluating adversarial attacks and defenses within a constrained feature space. We propose a standard protocol to evaluate the robustness of models and training methods across 5 datasets. We release a leaderboard based on more than 200 evaluations, a dataset zoo - a collection of real and synthetic datasets, - a model zoo - a collection of robust models ready to use for downstream application or further evaluation. The first version of this benchmark uses CAA, as the strongest available constrained adversarial attack. With this benchmark, we aim to support research in developing robustification methods against adversarial examples in the constrained feature space. Currently, the benchmark uses CAA as the strongest available attack in the constrained feature space, but we welcome contributions in the form of novel attacks.

- **An empirical study on the impact of industrial delays when mitigating distribution drifts.** (Chapter 8)
  We propose a novel evaluation protocol to bridge the gap between current assumptions in the evaluation of drift detectors in the literature and the reality of deploying such solutions in industrial contexts. In particular, our protocol considers labeling and deployment delays when evaluating retraining strategies, based on periodic retraining or drift detection. We show that our protocol better identifies the optimal retraining strategies fitting the practitioner's case. We conduct an empirical study on the effectiveness and efficiency of retraining strategies from the literature, both on BGL BNP Paribas use case and a publicly available counterpart. Notably, we show the impact of retraining window size, the importance of drift detector tuning, and how the delays affect the Pareto-optimal retraining strategies.

All in all the objective of this thesis is to bridge gaps between the current assumption of evaluating and robustifying ML models in the literature and the reality of ML models deployed in critical industrial systems. For the robustness of adversarial examples, we show that we can generate realistic adversarial examples to conduct adversarial testing by integrating domain knowledge in the generation process. Then we show how recent advances in synthetic data generation methods can improve the robustness of deep learning models to adversarial examples. Regarding the robustness of distribution drift, we propose a method to evaluate existing retraining strategies under real-world constraints, that improves decision-making on the strategy to adopt.

1.4 Structure

The structure of this thesis is as follows:

- **Chapter 2** provides the necessary background of this work.
- **Chapter 3** reports on the related work.
- **Chapter 4** presents our framework to generate constrained adversarial examples and proposes two attacks CPGD and MOEVA.
- **Chapter 5** presents CAA, an effective and efficient attack to conduct constrained adversarial testing by combining MOEVA and CAPGD, a gradient attack improvement of CPGD.
- **Chapter 6** evaluates defenses based on adversarial training and data augmentation.
- **Chapter 7** describes TabularBench, our benchmark for constrained adversarial attacks and defenses.
- **Chapter 8** proposes a novel evaluation protocol to bridge the gap between the evaluation process of drift detectors in the literature and the reality of industrial systems.
- **Chapter 9** concludes the dissertation.

2

Background

This chapter introduces the key concepts related to adversarial attacks and distribution drift, laying essential groundwork for the rest of the thesis.

Contents

Let us consider a classification problem defined over an input space Z and a set of class $\mathcal{Y} = \{1, ..., C\}$. Each input $z \in Z$ is an object of the considered application domain (e.g. malware [AGM+20], network data [CO19b], financial transactions [GCG+20]). We assume the existence of a feature mapping function $\varphi : Z \to \mathcal{X} \subseteq \mathbb{R}^n$ that maps Z to an n-dimensional feature space $\mathcal{X}$ over the feature set $F = \{f_1, f_2, ..f_n\}$.

We consider a C-class classifier $H : \mathcal{X} \to \mathcal{Y}$ that maps an n-feature vector $x = \{x_1, x_2, ..., x_n\} \in \mathcal{X}$ to a predicted class $y \in \mathcal{Y}$. The function $h : \mathcal{X} \to \mathbb{R}^C$ predicts a probability distribution over the set $\mathcal{Y}$. The predicted class $H(x)$ is the class with the highest probability, that is $H(x) = \arg\max_{0 \leq i \leq k} h_i(x)$.

2.1 Robustness against adversarial attacks

2.1.1 Adversarial attack

The objective of an adversary is to change the class $H(x)$ predicted by the classifier. In a targeted attack, the adversary seeks to change the classification to a specific, predefined class. Conversely, in an untargeted attack, the goal is simply to alter the model's prediction, regardless of the specific class it is changed to. For simplicity, and without loss of generality, we consider the untargeted case in the remaining of this section.

Un-constrained attacks

Given an original example x_0, the attack aims at generating an adversarial example $x_0+\delta$ that maximizes the probability of misclassification $H(x_0+\delta) \neq H(x_0)$. In unconstrained attacks, the adversary can freely perturb any feature.

The only restriction is that given a distance metric, $D : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, the distance between the clean example and the adversarial one is under a predefined threshold, $D(x, x + \delta) \leq \epsilon$. This restriction comes from computer vision, where D is traditionally an L_p norm, which ensures that the semantics of the perturbed image is preserved. In other domains, attacks can be adapted with specific distance. For instance in finance, [CAF+21] define a distance metric based on the feature importance and the probability of human checks. Also in finance, [KKT22] considers the cost and the utility (i.e. the gain) of updating each feature.

Therefore, the unconstrained attack objective is:

$$\begin{aligned} &\text{minimize } D(x, \hat{x}_0) \\ &\text{such that } H(\hat{x}_0) \neq H(x) \end{aligned} \tag{2.1}$$

Constrained feature space attacks

In image classification, the unconstrained feature space $\mathcal{X}$ is equivalent to the object space Z. For black-and-white images of a given resolution, the input

and object spaces are defined by the range of possible pixel values. In other domains such as finance and software security, the feature space is the result of the transformation from the object space to the feature space $\varphi : Z \to \mathcal{X}$. Each sample $z \in Z$ is an object of the considered application domain (e.g. malware, network data, transactions). Each object z respects some natural conditions in order to be valid. In the feature space, these conditions translate into a set of constraints over the feature values. For instance, the feature f_{loan} that corresponds to the loan amount is always positive. Plus, the function φ, which can be considered as the feature engineering steps, introduces additional constraints. Given a feature $f_{avg} \in F$ representing the average amount of money on the account over the last 6 months and f_{max} the maximum amount over the same period, this feature engineering creates a new constraint $f_{avg} \leq f_{max}$. We denote the set of constraint Ω and $\mathcal{X}_\Omega = x \in \mathcal{X} | \forall omega \in \Omega, x \vDash \omega$ the contained feature space. The objective of constrained feature space attacks is to generate examples in $\mathcal{X}_\Omega$. Hence, the constrained attack objective is:

$$\begin{aligned} &\text{minimize } D(x, \hat{x}_0) \\ &\text{such that } H(\hat{x}_0) \neq H(x) \\ &\qquad \hat{x}_0 \in \mathcal{X}_\Omega \end{aligned} \tag{2.2}$$

Problem space attack

Attacks in the problem space directly manipulate objects in the object space Z such as malware, URLs, or a set of transactions. The aim of the attack is to find a sequence of valid domain-object transformation $T = T_k \circ T_{k-1 \circ \ldots \circ T_1}$ such that $T(z)$ that satisfies object-space constraints γ. The constraints γ concern the adversarial goal (preservation of the semantics) and domain-specific restriction (plausibility) [PPC+20]. The objective of a problem-space attack is:

$$\begin{aligned} &\text{minimize } |T| \\ &\text{such that } T \in \mathcal{T} \\ &\qquad H(\varphi(T(z))) \neq H(\varphi(z)) \\ &\qquad T(z) \vDash \gamma \end{aligned} \tag{2.3}$$

with $\mathcal{T}$ the set of available transformation and $|T|$ the length of the sequence T. The primary drawback of these attacks is that they are domain-specific and demand significant engineering effort to be adapted to a new domain.

2.1.2 Robustification against adversarial attacks

There are different approaches to augment the robustness of a model to adversarial examples. We focus on two empirical methods that generate synthetic examples in the training process of the classifier H. First, adversarial training and

its variants which integrate adversarial examples in the training process to robustify. Second, data augmentation methods that, in combination with adversarial training have been shown effective.

2.1.3 Adversarial training

Adversarial training proposed by [SZS+13b] is the standard way to robustify models. The idea is to generate adversarial examples during training using a strong and efficient attack (e.g. FGSM[GSS15]) and use them to update the model parameters. We consider a classifier H with parameters θ trained with a loss function l. The most established way [MMS+17] to integrate these examples in the learning process of a neural network is to generate for each training batch, the worst-case adversarial examples (i.e. examples with the highest loss) and update the model's learnable parameters θ to minimize the classification error over these adversarial examples.

Hence, adversarial training solves the min-max optimization problem:

$$\min_{\theta} \mathbb{E}_{(x,y)\sim\mathcal{D}}\Big[\max_{\delta\in\mathcal{S}} l(\theta, x+\delta, y)\Big] \tag{2.4}$$

where (x, y) are training data samples from $\mathcal{D}$ distribution, $\delta \in \mathcal{S}$ is the perturbation in the set of allowed maximum perturbations, l is the loss of the model, and θ the model's learnable parameters. In practice, in adversarial training, this optimization problem is solved at each training batch.

This works well for models that can incrementally learn from data such as neural networks. For traditional models, **adversarial retraining** [CWC20] is more suited. In adversarial retraining, we use an attack A to generate adversarial examples over the original training set D to produce an adversarial training data $D^A = \{(\hat{x}_i, y_i)_{i=1}^N\}$. We train the models on the join of the original and adversarial training sets $D^* = D \cup D^A$. Adversarial retraining differs from adversarial training as it generates adversarial examples on a single version of the model. In contrast, adversarial training is iterative and at each step, new adversarial examples are generated against a potentially stronger model over time.

2.1.4 Data augmentation

In [RGC+21], Rebuffi et al. show that adversarial training can suffer from robust overfitting, where the robust test accuracy decreases during training, and so does the robustness of the model. The authors propose to use adversarial training with clean and synthetic data to reduce overfitting and augment the robustness of the model. In practice, during training, at each generation, we generate a set of synthetic examples D^+, and we use $(D_i \subseteq D) \cup D^+$, where D_i is the training set at iteration i. We then optimize the same min-max problem as in 2.4.

2.2 Distribution drift

This section introduces concepts related to distribution shift and drift detectors.

2.2.1 Definition and source of distribution drift

Inspired by the definition provided by Lu et al. in their survey [LLD+19], we define distribution shift as follows:

Given a time period $[0, t]$, the set of samples, denoted $S_{0,t} = \{d_0, \ldots, d_t\}$, where $d_i = (x_i, y_i)$ is one observation (or a data instance), $x_i \in X$ is the feature vector, $y_i \in Y$ is the label and $S_{0,t}$ follows a certain distribution $F_{0,t}(X, Y)$. Distribution drift occurs at timestamp $t + 1$, if $F_{0,t}(X, Y) \neq F_{t+1,\infty}(X, Y)$, denoted $\exists t : P_t(X, Y) \neq P_{t+1}(X, Y)$.

According to this definition, distribution drift can be defined as the change in the joint probability of X and Y at time t. The joint probability $P_t(X, Y)$ can be decomposed as $P_t(X, Y) = P_t(X) \times P_t(Y|X)$, therefore, the distribution drift can have three sources. We describe the three sources of distribution drift and explain how each source can influence the performance of the model.

1. **Covariate shift** $P_t(X) \neq P_{t+1}(X)$ while $P_t(Y|X) = P_{t+1}(Y|X)$, that is, only the input space distribution $P_t(X)$ changes and the relation $P_t(Y|X)$ remains unchanged. This kind of drift does not affect the true decision boundary. Therefore, it may or may not affect the performance of the model depending on the difference in the distributions and the generalization capability of the model. It remains interesting to study drifts in the $P(X)$ distribution as it does not require the label and is model-agnostic. For example, an increase in the average transaction amounts can lead to this kind of drift.

2. **Concept drift** $P_t(Y|X) \neq P_{t+1}(Y|X)$ while $P_t(X) = P_{t+1}(X)$, that is the distribution of input $P_t(X)$ remains unchanged, but the true decision boundary updates. Therefore, the decision boundary learned by the model is outdated and this causes a drop in accuracy in the region where the boundary has changed. For example, the changing economic context in which a financial system evolves can cause this type of drift.

3. **Label shift** $P_t(Y) \neq P_{t+1}(Y)$ and $P_t(X|Y) = P_{t+1}(X|Y)$, that is the distribution of label $P_t(X)$ changes but the distribution of features given the class $P_t(X|Y)$ does not change. For example, the change in the distribution of labels, such as the proportion of accepted and refused transactions, can lead to this type of drift.

While the primary focus in critical systems is on performance drift, identifying sources of distribution shift, such as changes in data distribution and decision

boundaries, serves as a valuable clue to identify early performance drift. Changes in data distribution and decision boundaries are identified by drift detectors.

2.2.2 Drift detectors

A drift detector is a method that observes a stream of data over time and determines for every new data point if the current distribution of the data has changed compared to a reference data set.

We survey the literature for available drift detectors and identify three types of detectors depending on the data distribution they observe: data-based detectors, error-based detectors, and predictive detectors.

Data drift detectors observe the distribution $P(X)$. For each batch of incoming examples, it determines whether the current distribution $P(X_{curr})$ is different from the reference distribution $P(X_{ref})$. The reference distribution is typically the one used to train the ML model, while the reference (also referred to as test) window is typically the latest data such that $|X_{curr}| = |X_{ref}$, where $|\cdot|$ denotes the cardinality of a set.

Error-based detectors observe the distribution of an arbitrary score function that takes as input the output of the model's prediction function $h(x)$ and the ground truth y. This score function is typically the Class Error (CE) (that is, returns 1 if the prediction was wrong else 0) or the Probability Error (PE) (that is, $1 - h_y(x)$, where y is the correct class label). Error-based detectors directly observe the error of the model; hence, we expect them to have a lower probability of false positives (i.e. detecting a drift that does not affect the performance of the model) than data-based or predictive detectors. However, by definition, error-based detectors require ground truth labels, which are not always available. The availability of the label highly depends on the use case and domain. For instance, in recommendation systems (e.g. on video platforms) the ground truth of the prediction is available as soon as the user provides feedback (e.g. click or do not click on the recommended video). On the contrary, the labeling of images remains costly, for example, in the medical domain [VC22].

Predictive detectors observe a metric computed during the inference of the model. Unlike error-based detectors, predictive detectors do not use the true label to predict the drift. For example, using an ensemble such as random forests, we can use the predicted probability from each tree to compute the uncertainty of the ensemble [SH20]. The ensemble's uncertainty can then be used as a metric for detecting drift.

3

Related Work

This chapter discusses the related work of tabular machine learning and its robustness to adversarial examples and distribution drift.

Contents

3.1 Tabular Deep Learning

Tabular data remains the most commonly used form of data [SA21], especially in critical applications such as medical diagnosis [UMC20; SRR+21], financial applications [GCG+20; CXY+20; CAF+21], user recommendation systems [ZYS+19], customer churn prediction [AAM+17; TXZ+20], cybersecurity [CO19b; AGM+20], and more. Improving the performance and robustness of tabular machine learning models for these applications is becoming critical as more ML-based solutions are cleared to be deployed in critical settings.

[BLS+22] showed that traditional deep neural networks tend to yield less favorable results in handling tabular data when compared to more shallow machine learning methods, such as XGBoost. They suggested four main reasons specific to tabular data, namely **low-quality** training data, complex irregular **spatial dependencies** between features, **sensitivity to preprocessing**, and **imbalanced importance** of features. To overcome these challenges, the tabular ML community proposed various optimizations that can be sorted across 3 families:

Data transformation methods such as **VIME** or Value Imputation for Mask Estimation [YZJ+20] uses self and then semi-supervised learning through deep encoders and predictors. The self-supervised encoder (a multilayer perceptron) is trained to learn a feature representation z, to be used for the downstream task. This encoder takes as input a corrupted sample x' input based on a clean sample x and a mask m. The encoder is trained on two tasks, independent of the downstream task. The first task is mask vector estimation whose objective is to predict corrupted value in a sample and recover the mask m from the feature representation z. The second, feature vector estimation, is to recover the original values of the sample x that have been corrupted. During semi-supervised learning, the predictor is trained using a supervised loss on labeled samples and a consistency loss on unlabeled samples. The consistency loss is computed as the difference between the prediction of corrupted (unlabeled) samples from the same clean samples but different masks.

Specialized architectures designed specifically for heterogeneous tabular data. **TabTransformer** is a transformer-based model [HKC+20]. It uses self-attention to map the categorical features to an interpretable contextual embedding, and the paper claims this embedding improves the robustness of models to noisy inputs. **TabNet** is another transformer-based model [AP21]. Similarly to a decision tree, it uses multiple subnetworks in hierarchical sequence. At each decision step, it uses sequential attention to choose which features to reason. TabNet aggregates the outputs of each step to obtain the decision.

Regularization models propose novel loss functions and training processes. **RLN** or Regularization Learning Networks [SS18] uses an efficient hyperparameter tuning scheme to minimize a counterfactual loss. The authors train a regularization

coefficient to weights in the neural network to lower the sensitivity and produce very sparse networks. **STG** or Stochastic Gates [YLN+20] uses stochastic gates for feature selection in neural network estimation problems. The method is based on probabilistic relaxation of the l_0 norm of features or the count of the number of selected features.

Recent approaches like RLN [SS18] and TabNet [AP21] are catching up and even outperforming shallow models in some settings. We argue that DNNs for Tabular Data are sufficiently mature and competitive with shallow models and require therefore a thorough investigation of their safety and robustness.

3.2 Adversarial attacks against tabular data

Adversarial attacks on tabular data require special attention to the specific characteristics of the data, such as feature types and the relationships between features, due to the underlying semantic structure.

3.2.1 Realistic Adversarial Examples

Initially applied to computer vision, adversarial examples have also been adapted and evaluated on tabular data. [BAL+19] considered feature importance to craft the attacks. [KKT22] suggested considering both the cost and benefit of perturbing each feature. [MLK+22] considered mutability, type, boundary, and data distribution constraints, Domain constraints are emerging as critical elements for developing effective adversarial attacks. This last approach is closest to the trend in adversarial machine learning in critical scenarios such as malware and finance [PPC+20; DGS+22; GCG+20].

Our work follows this last hypothesis and focuses on constrained feature-space attacks to realistically assess the robustness of deep tabular learning models.

While domain constraints satisfaction is essential for successful attacks, research on robustness for industrial settings (eg [GCG+20] with a major bank) also demonstrated that imperceptibility remains important for critical systems with human-in-the-loop mechanisms, which could deflect attacks with manual checks from human operators. Imperceptibility is domain-specific, and multiple approaches have been suggested [BAL+19; KKT22; DGS+22]. None of these approaches was confronted with human assessments or compared with each other, and in our study, we decided to use the most established L_2 norm. Our algorithms and approaches are generalizable to further distance metrics and imperceptibility definitions.

Overall, none of the existing attacks for tabular machine learning supports the feature relationships inherent to realistic tabular datasets, as summarized in Table 3.1. Nevertheless, we evaluate all the approaches that support continuous values and where a public implementation is available to confirm our claims: LowProFool, BF*, CPGD, and MOEVA.

Table 3.1: Evasion attacks for tabular machine learning. Attacks with a public implementation in bold. Cont. = continuous, Disc = discrete, Cat. = categorical, and Rel. = relation.

Attack	Supported features			Supported constraints		
	Cont.	Disc.	Cat.	Cat.	Disc.	Rel.
LowProFool (LPF) [BAL+19]	✓	X	X	X	X	X
[CAF+21]	✓	✓	✓	✓	✓	X
[GHS+21]	✓	✓	✓	✓	✓	X
[XHR+23]	X	X	✓	✓	X	X
[WHB+20]	X	X	✓	✓	X	X
[BHZ+23]	X	X	✓	✓	X	X
BF*/BFS [KHS+18; KKT23]	✓	✓	✓	✓	✓	X
[MLK+22]	✓	✓	✓	✓	✓	X
CPGD, MOEVA (OURS)	✓	✓	✓	✓	✓	✓
CAPGD, CAA (OURS)	✓	✓	✓	✓	✓	✓

3.2.2 Adversarial attacks for constrained domains

In the constrained adversarial attacks literature, most of the studies upgrade existing attacks to support some constraints and only a few propose new algorithms tailored to each domain specifically. One of these novel methods for crafting adversarial attacks that respect domain constraints was proposed by [KHS+18]. The authors use a graphical framework where each edge represents a feasible transformation. The nodes of the graph are the transformed examples. The downside of this method is that it works only with linear models, and it comes with high computation costs. [CO19b] builds an iterative gradient-based adversarial attack that considers groups of feature families. The modifications that do not respect the constraints are updated until they enter a feasible region. The implementation provided by the authors is closely related to the botnet domain, therefore it can not be reused in new domains. [LLY+20] considers simple linear constraints for cyber-physical systems. They generate practical examples using a best-effort search algorithm. However, the solution is not scalable to practical applications that have more complex non-linear constraints. Other researchers tune existing attacks to support constraints. [SPW+20] extends the traditional JSMA algorithm to support constraints resolution. They use the concept of a primary feature in a group of interdependent features. Meanwhile, Tian et al. [TWT+20] introduced C-IFGSM an updated version of FGSM that considers feature correlations. They embed the correlations into a constraint matrix which is used to calculate the Hadamard product with the sign of the gradient in order to determine the direction

and magnitude of the update. [TPS20] crafted constrained adversarial examples for NIDS by extending the optimization function in the Carlini and Wagner attack. They group flow-based features by their modification feasibility and assign weights to each group based on the level of difficulty of the modification. The authors in [EBM+21] introduce non-uniform perturbation in the PGD attack to enable adversarial retraining with more realistic examples. They use Pearson's correlation coefficient, Shapley values, and masking to build a matrix that constrains the direction and magnitude of change similarly to [TWT+20]. All the attacks above are upgraded to handle simple and general constraints but do not deal with more complex domain-specific constraints. Therefore, the samples they generate are more restricted but not totally feasible.

[GCG+20] proposed a black-box, genetic-based approach to generate constrained adversarial examples for an industrial credit scoring system. This work is the closest to one of the attacks we propose in this thesis. However, their evaluation is limited to the financial domain.

Therefore, there is a need for attacks that can generate realistic examples while respecting domain constraints in tabular data.

3.2.3 Defending against adversarial examples.

The standard method to defend against adversarial examples is Madry adversarial training [MMS+17]. Madry et al. define the problem of adversarial robustness as a saddle point problem where the training algorithm updates the parameters by minimizing the loss over the adversarial that attempts to maximize this loss. In [MMS+17] they showed the importance of using a strong attack during adversarial training [GSS15], and proposed Projected Gradient Descent.

Rice et al. [RWK20] demonstrated that adversarial training is prone to *robust overfitting*, a phenomenon where test set accuracy rapidly declines while train set accuracy continues to improve. Rice et al. [RWK20] suggest early stopping as the primary approach to counteract robust overfitting and show that models trained with early stopping are more robust than those using other regularization techniques like data augmentation. In contrast, Rebuffi et al. [RGC+21] argue that combining data augmentation with regularization methods, such as weight averaging [IPG+18], can significantly enhance robustness.

A large variety of defenses have been proposed against adversarial examples, leveraging different techniques as gradient masking [XZZ19], model ensemble [VS19; PXD+19; SRR20], adversarial detection [YHG+19; YKR19] or preprocessing steps [YZK+19]. However, Tramèr et al. [TCB+20] demonstrate that several of these defenses, although evaluated on existing adaptive attacks, could still be broken by new adaptive attacks. More recently Carlini [Car23] showed the simplicity of creating such attacks by letting a large language model propose the attack. Hence, adversarial training based methods are so far the only reliable defense against

evasion attacks.

3.3 Distribution drift mitigation

3.3.1 Drift detectors

Table 3.2: Summary of drift detectors

Detector	Type	Detection mode
No detection	Baseline	None
Periodic detector	Baseline	Periodic
Statistical test [VKV+22]	Data	Periodic
Divergence metric [Inc22]	Data	Periodic
PCA-CD [QAW+15]	Data	Periodic
ADWIN [BG07a]	Error	Stream
DDM [GMC+04]	Error	Stream
EDDM [BCF+06]	Error	Stream
HDDM-A [FCR+15]	Error	Stream
HDDM-W [FCR+15]	Error	Stream
KSWIN [RHS20]	Error	Stream
Page-Hinkley [Pag54]	Error	Stream
Uncertainty [SH20]	Predictive	Periodic
Aries [HGX+23]	Predictive	Periodic

A drift detector is a method that observes a stream of data over time and determines for every new data point if the current distribution of the data has changed compared to a reference data set. There is a large literature on drift detectors but to the extent of our knowledge, none of the papers propose to study the applicability of drift detectors in a real-world scenario with delays.

We survey the literature for available drift detectors and identify three types of detectors: data-based detectors, error-based detectors, and predictive detectors as explained in Section 2.2.2. The alibi python library [VKV+22] observes the $P(X)$ distribution feature by features using the K-S test for numerical features, and the chi-squared test for categorical ones. To handle multivariate data, the tool uses the Bonferroni [Dun61] or False Discovery Rate (FDR) correction [BH95]. The evidently library [Inc22] uses Wasserstein distance for numerical features and the Jensen-Shannon divergence for categorical features. If the distance exceeds a predefined threshold, the feature is flagged as shifted. For multivariate data, the tool proposes to detect shifts if more than a predefined number of feature

drifts. In [QAW+15], the method PCA-CD proposes to apply Principal Component Analysis (PCA) to the data X. They show that PCA not only preserves the ability to detect changes in the mean and standard deviation of individual features but also allows one to detect changes in the feature correlations. PCA-CD computes a divergence score on and uses the Page-Hinkley method [Pag54] to detect drift in this metric. [KVD+14] proposed a theoretically elegant approach for multivariate data using kdq-trees. However, this approach does not scale well to high-dimension datasets. Data-based drift detectors have been shown effective at detecting drift in the data distribution in their respective papers. However, little is known about their effectiveness mitigate performance degradation over time, when used to trigger model retrains.

Error-based detectors observe the distribution of an arbitrary (univariate) score function that takes as input the prediction of the model and the ground truth. The simplest approach, DDM [GMC+04], observes the error rate and detects drifts if the error rate surpasses a certain threshold above the minimum error rate observed over time. Similarly, [BCF+06] observes the error rate but detects drift by keeping track of the average distance between two errors. In [FCR+15], HDDM-A and HDDM-W observe the error rate and use respectively Hoeffding's and McDiarmid's inequalities to detect drift. [BG07a] observes an arbitrary score function using an adaptive windowing mechanism. KSWIN [RHS20] uses the Kolmogorov-Smirnov on a sliding window. The Page-Hinkley[Pag54] method uses the mean and a cumulative sum and detects drif when the cumulative sum exceeds a predefined threshold. Error-based drift detectors have been shown effective in detecting change in the performance of models. However, their applicability in industrial contexts with labeling and deployment delay constraints remains unexplored.

Finally, one can take advantage of recent advances in uncertainty estimation and test accuracy prediction to build drift detectors. For instance, Aries [HGX+23] compares the distance to the model decision boundary of the test samples with that of the training sample to estimate the accuracy of test examples. Based on this accuracy, we simulate the error rate using a uniform distribution controlled by the estimated accuracy. We use ADWIN to monitor the error rate. The uncertainty drift detector uses Bayesian approaches [SH20] to compute the epistemic, allegoric, and total uncertainty. We use ADWIN to observe the uncertainty distribution. Although using such an approach to detect drift is a natural application, to the extent of our knowledge it has not been explored.

Table 3.2 summarizes the drift detectors that can be applied in our industrial context.

3.3.2 Delays in time series evaluation

Masud et al. [MGK+10] study the problem of novel class detection while considering the true label delay constraints. They show how delaying the classification of incoming samples helps to detect new classes before the true label is available, hence providing a more accurate prediction.

[PCvD+22] investigated recently the reliability of data and error-based drift detectors. However, the experimental protocol used does not consider label and deployment in production delays, which is the core of our study.

In [PA16], Plasse et. al introduced a taxonomy to describe the labeling delay mechanism. They showed how delayed labels can be used to pre-update classifiers in real-world applications. However, the study didn't tackle validation delays and their impact on the model's deployment or the drift monitoring process.

[Žli10] analyzed the factors that allow distribution drift detection before the label is available. However, no experiments are conducted on the impact of the findings on the performance of ML predictions, which is the aim of our novel protocol.

3.3.3 Domain Generalization

The domain generalization (DG) problem was first formally introduced by Blanchard et al. [BLS11]. Unlike other related learning problems such as domain adaptation or transfer learning, DG considers the scenarios where target data is *inaccessible* during model learning. Hence, adaptation to distribution shift falls under the umbrella of domain generalization. [ZLQ+22] introduced a categorization of techniques commonly used to address the DG challenge, including domain alignment training, synthetic data augmentation, and self-supervised learning. Our work stems from the need of our industrial partner to improve the monitoring of the deployed models and to effectively trigger their well-established retraining procedures. While we believe that DG techniques could also improve the performance of the models against distribution shift, all these approaches are orthogonal to our investigations for an efficient retraining schedule under delay.

3.3.4 Online learning

Online learning is an orthogonal line of research to our work, and labeling delays remain scarcely explored. [GGB20] proposed an evaluation procedure that updates the model online with new samples and labels following the labeling delay. Similar to our study, they considered the case where the predictions can be refined when a newer version of the model becomes available. They also performed a continuous evaluation of the latest samples encountered the improve the accuracy of the models. However, online deployment of new models is impossible in our financial industrial setting. In addition, retroactive evaluation of new samples as

they propose is unrealistic in our production setting.

4

A Unified Framework for Adversarial Attacks in Constraint Feature Space

Existing adversarial attacks generate unconstrained and unrealistic examples. In this chapter, we propose a framework for constrained adversarial examples. We instantiate our framework with two attacks, CPGD (gradient-based) and MOEVA (search-based) to enhance the evaluation of machine learning models' robustness.

Contents

4.1 Introduction

In this chapter, we address the first challenge of this thesis, which is to generate valid adversarial examples in the constrained feature space.

Research on adversarial examples initially focused on image recognition [DDS+04; SZS+13a] but has, since then, demonstrated that the adversarial threat concerns many domains including cybersecurity [PPC+20; SPW+20], natural language processing [ASE+18], software security [YAY20], cyber-physical systems [LLY+20], finance [GCG+20], manufacturing [MH20], and more.

A peculiarity of these domains is that the ML model is integrated into a larger software system that takes as input domain objects (e.g. financial transactions, malware, network traffic). Therefore altering an original example in any direction may result in an example that is *infeasible* in the real world. This contrasts with images that generally remain valid after slight pixel alterations. Hence, a successful adversarial example should not only fool the model and keep a minimal distance to the original example but also satisfy the inherent *domain constraints*.

As a result, generic adversarial attacks that were designed for images – and are unaware of constraints – equally fail to produce feasible adversarial examples in constrained domains [GCG+20; TWT+20]. A blind application of these attacks would distort model robustness assessment and prevent the study of proper defense mechanisms.

Problem-space attacks are algorithms that directly manipulate *problem objects* (e.g. malware code [AGM+20; PPC+20], audio files [DJL+20], wireless signal [SL19]) to produce adversarial examples. While these approaches guarantee by construction that they generate feasible examples, they require the specification of domain-specific transformations [PPC+20]. Their application, therefore, remains confined to the particular domain they were designed for. Additionally, the manipulation and validation of problem objects are computationally more complex then working with feature vectors.

An alternative to problem-space attacks is feature-space attacks that enforce the satisfaction of the domain constraints. Some approaches for constrained feature space attacks modify generic gradient-based attacks to account for constraints [SPW+20; TWT+20; EBM+21] but are limited to a strict subset of the constraints that occur in real-world applications (read more in Section 3.2, where we discuss the related work thoroughly). Other approaches tailored to a specific domain manage to produce feasible examples [CO19b; LLY+20; GCG+20] but would require drastic modifications throughout all their components to be transferred to other domains. To this date, there is a lack of generic attacks for robustness assessment of domain-specific models and a lack of cross-domain evaluation of defense mechanisms.

In this chapter, we propose a unified framework for constrained feature-space

attacks that applies to different domains without tuning *and* ensures the production of feasible examples. Based on our review of the literature and our analysis of the covered application domains, we propose a *generic constraint language* that enables the definition of (linear and non-linear) relationships between features. We, then, automatically translate these constraints into two attack algorithms that we propose. The first is *Constrained Projected Gradient Descent* (CPGD) – a white-box alteration of PGD that incorporates differentiable constraints as a penalty in the loss function that PGD aims to maximize, and post-processes the generated examples to account for non-differentiable constraints. The second is Multi-Objective EVolutionary Adversarial Attack (MOEVA) – a grey-box multi-objective search approach that treats misclassification, perturbation distance, and constraints satisfaction as three objectives to optimize. The ultimate advantage of our framework is that it requires the end-user only to specify what domain constraints exist over the features. The user can then apply any of our two algorithms to generate feasible examples for the target domain.

We have conducted a large empirical study to evaluate the utility of our framework. Our study involves four datasets from finance and cybersecurity, and two types of classification models (neural networks and random forests). Our results demonstrate that our framework successfully crafts feasible adversarial examples. Specifically, MOEVA does so with a success rate of up to 100%, whereas CPGD succeeded on the finance dataset only (with a success rate of 9.85%).

In turn, we investigate strategies to improve model robustness against feasible adversarial examples. We show that adversarial retraining on feasible examples can reduce the success rate of CPGD down to 2.70% and the success rate of the all-powerful MOEVA down to 85.20% and 0.80% on the finance and cybersecurity datasets, respectively.

To summarize, the contributions of this chapter are:

1. A generic constraint language that enables the definition of relationships between features in tabular data.

2. Constrained Projected Gradient Descent (CPGD), a gradient-based attack derived from PGD that incorporates differentiable constraints.

3. MOEVA, a grey-box genetic-based attack that treats misclassification, perturbation distance, and constraints satisfaction as objectives to optimize.

4. A large-scale empirical study revealing that attacks unaware of domain constraints mostly fail to generate adversarial examples but dedicated methods, in particular MOEVA can generate constrained examples.

4.2 Problem Formulation

We formulate below the problem for binary classification. We generalize to multi-class classification problems in Appendix B.

4.2.1 Constraint Language

Let us consider a classification problem defined over an input space Z and a binary set $\mathcal{Y} = \{0, 1\}$. Each input $z \in Z$ is an object of the considered application domain (e.g. malware [AGM+20], network data [CO19b], financial transactions [GCG+20]). We assume the existence of a feature mapping function $\varphi : Z \longrightarrow \mathcal{X} \subseteq \mathbb{R}^n$ that maps Z to an n-dimensional feature space $\mathcal{X}$ over the feature set $F = \{f_1, f_2, ..f_n\}$. For simplicity, we assume $\mathcal{X}$ to be normalized such that $\mathcal{X} \subseteq [0, 1]^n$. That is, for all $z \in Z$, $\varphi(z)$ is an n-sized feature vector $x = (x_1 \dots x_n)$ where $x_i \in [0, 1]$ and is the j-th feature. Each object z respects some natural conditions in order to be valid. In the feature space, these conditions translate into a set of constraints over the feature values, which we denote by Ω. By construction, any feature vector x generated from a real-world object z satisfies all constraints $\omega \in \Omega$.

Based on our review of the literature, we have designed a constraint language to capture and generalize the types of feature constraints that occur in the surveyed domains. Our framework allows the definition of constraint formulae according to the following grammar:

$$\begin{aligned} \omega &:= \omega_1 \wedge \omega_2 \mid \omega_1 \vee \omega_2 \mid \psi_1 \succeq \psi_2 \\ \psi &:= c \mid f \mid \psi_1 \oplus \psi_2 \mid x_i \end{aligned} \tag{4.1}$$

where $f \in F$, c is a constant real value, $\omega, \omega_1, \omega_2$ are constraint formulae, $\succeq \in \{<, \leq, =, \neq, \geq, >\}$, $\psi, \psi_1, \dots, \psi_k$ are numeric expressions, $\oplus \in \{+, -, *, /\}$, and x_i is the value of the i-th feature of the original input x.

One can observe from the above that our formalism captures, in particular, feature boundaries (e.g. $f > 0$) and numerical relationships between features (e.g. $f_1/f_2 < f_3$) – two forms of constraints that have been extensively used in the literature [CO19b; GCG+20; TWT+20; LLY+20].

4.2.2 Threat Model and Attack Objective

Let a function $H : \mathcal{X} \to \mathcal{Y}$ be a binary classifier and function $h : \mathcal{X} \to \mathbb{R}$ be a single output predictor that predicts a continuous probability score. The function $h : \mathcal{X} \to \mathbb{R}^C$ predicts a probability distribution over the set $\mathcal{Y}$. The predicted class $H(x)$ is the class with the highest probability, that is $H(x) = \arg\max_{0 \leq i \leq k} h_i(x)$.

In our threat model, we assume that the attacker has knowledge of h and its parameters, as well as of F and Ω. We also assume that the attacker can directly

modify a subset of the feature vector $x = (x_1 \dots x_m)$, with $m \leq n$. We refer to this subset as the set of **mutable features**. The attacker can only feed the example to the system if this example satisfies Ω.

Given an original example x, the **attack objective** is to generate an adversarial example $x + \delta$ such that $H(x + \delta) \neq H(x)$, $\delta < \epsilon$ for a maximal perturbation threshold ϵ under a given *p-norm*, and $x + \delta \in \mathcal{X}_\Omega$. While domain constraints guarantee that an example is feasible, (e.g. total credit amount must be equal to the monthly payment times the duration in months), we limit the maximum perturbation to produce imperceptible adversarial examples. By convention, one may prefer $p = \infty$ for continuous features, $p = 1$ for binary features, and $p = 2$ for a combination of continuous and binary features. We refer to such examples $x + \delta$ as a *constrained adversarial example*. We also name *constrained adversarial attack* algorithms that aim to produce the above optimal constrained adversarial example. We propose two such attacks.

4.3 Constrained Projected Gradient Descent

Past research has shown that multi-step gradient attacks like PGD are among the strongest attacks [KGB16]. PGD adds iteratively a perturbation δ that follows the sign of the gradient ∇ with respect to the current adversary x_t of the input x. That is, at iteration $t + 1$ it produces the input

$$x^{t+1} = \Pi_{x+\delta}(x^t + \alpha sgn(\nabla_{x^t} l(\theta_h, x^t, y))) \tag{4.2}$$

where θ_h the parameters of our predictor h, Π is a clip function ensuring that $x + \delta$ remains bounded in a sphere around x of a size ϵ using a norm p, and $\nabla_x l$ is the gradient of the loss function tailored to our task, computed over the set of mutable features. For instance, we can use cross-entropy losses with a mask for classification tasks. We compute the gradient using the first-order approximation of the loss function around x.

However, as our experiments reveal (see Table 4.2 and Section 4.6), a straight application of PGD does not manage to generate any example that satisfies Ω. This raises the need to equip PGD with the means of handling domain constraints.

An out-of-the-box solution that we have experimented with is to pair PGD with a mathematical programming solver, i.e. Gurobi [Gur22]. Once PGD managed to generate an adversarial example (not satisfying the constraint), we invoke the solver to find a solution to the set of constraints close to the example that PGD generated (and under a perturbation sphere of ϵ size). Unfortunately, this solution does not work out either because the updated examples do not fool the classifier anymore or the solver simply cannot find an optimal solution given the perturbation size.

In face of this failure, we conclude that this gradient-based attack cannot generate constrained adversarial examples if we do not revisit its fundamentals in

Table 4.1: From constraint formulae to penalty functions. τ is an infinitesimal value.

Constraints formulae	Penalty function
$\omega_1 \wedge \omega_2$	$\omega_1 + \omega_2$
$\omega_1 \vee \omega_2$	$\min(\omega_1, \omega_2)$
$\psi_1 \leq \psi_2$	$\max(0, \psi_1 - \psi_2)$
$\psi_1 < \psi_2$	$\max(0, \psi_1 - \psi_2 + \tau)$
$\psi_1 = \psi_2$	$\mid \psi_1 - \psi_2 \mid$

light of the new attack objective. We, therefore, propose to develop a new method that considers the satisfaction of constraints as an integral part of the perturbation computation.

Concretely, we define a penalty function that represents how far an example x is from satisfying the constraints. More precisely, we express each constraint ω_i as a penalty function $penalty(x, \omega_i)$ over x such that x satisfies ω_i if and only if $penalty(x, \omega_i) <= 0$. Table 4.1 shows how each constraint formula (as defined in our constraint language) translated into such a function. The global distance to constraint satisfaction is, then, the sum of the non-negative individual penalty functions, that is, $penalty(x, \Omega) = \sum_{\omega_i \in \Omega} penalty(x, \omega_i)$.

The principle of our new attack, CPGD, is to integrate the constraint penalty function as a negative term in the loss that PGD aims to maximize. Hence, given an input x, CPGD looks for the perturbation δ defined as

$$\arg\max_{\delta : \|\delta\|_p \leq \epsilon} \{ l(h(x+\delta), y) - \sum_{\omega_i \in \Omega} penalty(x+\delta, \omega_i) \} \tag{4.3}$$

The challenge in solving (4.3) is the general non-convexity of $penalty(x, \Omega)$. To recover tractability, we propose to approximate (4.3) by a convex restriction of $penalty(x, \Omega)$ to the subset of the convex penalty functions. Under this restriction, all the penalty functions used in (4.3) are convex, and we can derive the first-order Taylor expansion of the loss function and use it at each iterative step to guide CPGD. Accordingly, CPGD produces examples iteratively as follows:

$$\begin{aligned} x^{t+1} = \Pi_{x+\delta}(R(x^t + \alpha sgn(\nabla_{x^t} l(h(x^t), y) \\ - \sum_{\phi_i} \nabla_{x^t} penalty(x^t, \phi_i)))) \end{aligned} \tag{4.4}$$

with $x^0 = x$, and R a repair function. At each iteration t, R updates the features of the example to repair the non-convex constraints whose penalty functions are not back-propagated with the gradient $\nabla_{x^t} l$ (if any).

4.4 Multi-Objective Generation of Constrained Adversarial Examples

As an alternative to CPGD, we propose MOEVA, a multi-objective optimization algorithm whose fitness function is driven by the attack objective described in Section 4.2.

4.4.1 Objective Function

We express the generation of constrained adversarial examples as a multi-objective optimization problem that reflects three requirements: misclassification of the example, maximal distance to the original example, and satisfaction of the domain constraints. By convention, we express these three objectives as a minimization problem.

The first objective of a constrained attack is to cause misclassification by the model. When provided an input x, the binary classifier H outputs $h(x)$, the prediction probability that x lies in class 1. If $h(x)$ is above the classification threshold t, the model classifies x as 1; otherwise as 0. Without knowledge of t, we consider $h(x)$ to be the distance of x to class 0. By minimizing $h(x)$, we increase the likelihood that the H misclassifies the example irrespective of t. Hence, the first objective that MOEVA minimizes is $g_1(x) \equiv h(x)$.

The second objective is to minimize perturbation between the original example x^0 and the adversarial example, to limit the perceptibility of the crafted perturbations. We use the conventional L_p distance to measure this perturbation. The second objective is $g_2(x) \equiv L_p(x - x^0)$.

The third objective is to satisfy the domain constraints. Here, we reuse the penalty functions that we defined in Table 4.1. The third and last objective function is thus $g_3(x) \equiv \sum_{\omega_i \in \Omega} penalty(x, \omega_i)$.

Accordingly, the constrained adversarial attack objective translates into MOEVA a three-objective function to minimize with three validity conditions, that is:

$$\begin{array}{ll} \text{minimise } g_1(x) \equiv h(x) & \text{s.t. } g_1(x) < t \\ \text{minimise } g_2(x) \equiv L_p(x - x^0) & \quad\; g_2(x) \leq \epsilon \\ \text{minimise } g_3(x) \equiv \sum\limits_{\omega_i \in \Omega} \text{penalty}(x, \omega_i) & \quad\; g_3(x) = 0 \end{array} \tag{4.5}$$

and this three-objective function also forms the fitness function that MOEVA uses to assess candidate solutions.

4.4.2 Genetic Algorithm

We instantiate MOEVA as a multi-objective genetic algorithm, namely based on R-NSGA-III [VDB18]. We describe below how we specify the different components

of this algorithm. It is noteworthy, however, that our general approach is not bound to R-NSGA-III. In particular, the three-objective function described above can give rise to other search-based approaches for constrained adversarial attacks.

Algorithm 1 Generation process of MOEVA

1: **Input:** $\mathbf{x_0}$, an original example
2: $G\_objectives[g_1, g_2, g_3]$, the 3 objective functions
3: N_{gen}, number of generations
4: L, population size
5: C, number of children
6: **Output:** A population P of adversarial examples minimizing the $G\_objectives$ functions
7: $P \leftarrow \text{init}(\mathbf{x_0}, L)$
8: **for** $j = 1$ to N_{gen} **do**
9: $P_{parents} \leftarrow \text{select\_random}(P, C)$
10: $P_{offspring} \leftarrow \text{twoPointsCrossover}(P_{parents})$
11: $P \leftarrow P \cup \text{polyMutate}(P_{offspring})$
12: $P_{objectives} \leftarrow \text{evaluate}(P, G\_objectives)$
13: $P \leftarrow \text{survive}(P, P_{objectives}, L)$
14: **end for**
15: **return** P

Algorithm 1 formalizes the generation process of MOEVA.

Population initialization (l. 7). The algorithm first initializes a population P of L solutions. Here, an individual represents a particular example that MOEVA has produced through successive alterations of a given original example x. We specify that the initial population comprises L copies of x. The reason we do so is that we noticed, through preliminary experiments, that this initialization was more effective than using random examples. This is because the original input inherently satisfies the constraints, which makes it easier to alter it into adversarial inputs that satisfy the constraints as well.

Population evolution (l. 8-14). The algorithm proceeds iteratively and makes the population evolve into new "generations", until it reaches a predefined number N_{gen} of iterations/generations. At each iteration, MOEVA evaluates each individual in the current population through the three-objective function defined above. Hence, we know at each stage if the algorithm has managed to successfully generate constrained adversarial examples. The evolution of the population then proceeds in three steps:

1. *Crossover (l. 9-10):* We create new individuals using two-point binary crossover [DSO07]. This crossover is useful for our approach to preserve

constraint satisfaction, as the "children" individuals keep the feature values of their "parents". We randomly select the parents from the current population.

2. *Mutation (l. 11):* To introduce diversity in the population, we randomly alter the features of each "child" (resulting from crossover) using polynomial mutation [DSO07]. Our mutation operator enforces the satisfaction of constraints that involve a single feature, e.g. it preserves boundary constraints and does not change immutable features. The set of children that result from this mutation process is then added to the current population.

3. *Survival (l. 12-13):* MOEVA next determines which individuals it should keep in the next generation. Being based on R-NSGA-III, our algorithm uses non-dominance sorting and reference directions to make this selection, based on our three objective functions. That is, we place non-dominated individuals in a first Pareto front and repeat (without replacement) until we reach N Pareto fronts. Individuals in these N Pareto fronts form the next generation and the others are discarded. If there are less than L individuals (i.e., the population size), then, the algorithm fills the population with individuals from the $N+1$-th front, selected using reference directions – an approach that aims to maximize diversity in the selection [BDD+21].

After the specified number of generations, the algorithm returns the last population together with the evaluation of the three objective functions.

4.5 Experimental Settings

4.5.1 Datasets and Constraints

Because images are devoid of constraints and fall outside the scope of our framework, we evaluate CPGD and MOEVA on four datasets coming from inherently constrained domains. These datasets bear different sizes, features, and types (and numbers) of constraints. We evaluate both neural networks (NN) and random forest (RF) classifiers. More details about datasets and models are in Appendix C.

LCLD. It is inspired by the Lending Club Loan Data [Kag19]. Therein, examples are credit requests that can be accepted or rejected. We trained a neural network and a random forest that both reach an AUROC score of 0.72. We have identified constraints that include 94 boundary conditions, 19 immutable features, and 10 feature relationship constraints (3 linear, 7 non-linear). For example, the installment (I), the loan amount (L), the interest rate (R), and the term (T) are linked by the relation $I = L * R(1+R)^T/((1+R)^T - 1)$.

CTU-13. It is a feature-engineered version of CTU-13, proposed by [CO19b]. It includes a mix of legit and botnet traffic flows from the CTU campus. We trained a neural network and a random forest to classify legit and botnet traffic, which both achieved an AUROC of 0.99. We identified 324 immutable features and 360 feature relationship constraints (326 linear, 34 non-linear). For example, the maximum packet size for TCP ports should be 1500 bytes.

Malware. It comprises features extracted from a collection of benign and malware PE files [AGM+20]. We trained a random forest with an AUROC of 0.99. We identified 88 immutable features and 7 feature relationship constraints (4 linear, 3 non-linear). For example, the sum of binary features set to 1 that describe API imports should be less than the value of the feature api_nb, which represents the total number of imports on the PE file.

URL. It comes from [HY21] and contains a set of legitimate or phishing URLs. The random forest we use has an AUROC of 0.97. We have identified 14 relation constraints between the URL features, including 7 linear constraints (e.g. hostname length is at most equal to URL length) and 7 are if-then-else constraints.

4.5.2 Experimental Protocol and Parameters

In all datasets, a typical attack would be interested in fooling the model to classify a malicious class (rejected credit, botnet, malware, and phishing URL) into a target class (accepted, legit, benign, and legit URL). By convention, we denote by 1 the malicious class and by 0 the target class.

We evaluate the success rate of the attacks on the trained models using, as original examples, a subset of the test data from class 1. In LCLD we take 4000 randomly selected examples from the candidates, to limit computation cost while maintaining confidence in the generalization of the results. For CTU-13, Malware, and URL, we use respectively all 389, 1308, and 1129 test examples that are classified in class 1.

Since all datasets comprise binary, integer, and continuous features, we use the L_2 distance to measure the perturbation between the original examples and the generated examples.

We detail and justify in Appendices C and D the attack parameters including perturbation threshold ϵ, the number of generations and population size for the genetic algorithm attack, and the number of iterations for the gradient attack.

4.6 Experimental Results

4.6.1 Attack Success Rate

Table 4.2 shows the success rate of PGD, PGD + SAT, CPGD, and MOEVA on the two neural networks that we have trained on LCLD and CTU-13; and the

Table 4.2: Success rate (C&M) of the attacks on the neural network (NN) and random forest (RF) models, in % of the original examples. M is the success rate disregarding constraint satisfaction; C is the ratio of original examples where the attack found examples that satisfy the constraints and are within the perturbation bound.

Model	Dataset	Attack	C	M	C&M
Neural Network	LCLD	PGD	0.00	22.20	0.00
		PGD + SAT	2.43	0.00	0.00
		CPGD	61.68	22.03	9.85
		MOEVA	100.00	99.90	97.48
	CTU-13	PGD	0.00	100.00	0.00
		PGD + SAT	100.00	0.00	0.00
		CPGD	0.00	17.57	0.00
		MOEVA	100.00	100.00	100.00
Random Forest	LCLD	Papernot	0.00	11.86	0.00
		MOEVA	99.98	61.84	41.51
	CTU-13	Papernot	79.36	13.02	0.0
		MOEVA	100.00	7.62	5.41
	Malware	Papernot	0.00	51.99	0.00
		MOEVA	100.00	100.00	39.30
	URL	Papernot	84.23	11.25	8.50
		MOEVA	100.00	32.06	31.89

success rate of the Papernot attack and MOEVA on the random forests that we have trained on each dataset. More precisely, we use the extension of the original Papernot attack [PMG16] that [GCG+20] proposed to make this attack applicable to random forests.

PGD and PGD + SAT fail to generate any constrained adversarial examples. The problem with PGD is that it fails to satisfy the domain constraints. While the use of a SAT solver fixes this issue, the resulting examples are classified correctly. CPGD can create LCLD examples that satisfy the constraints and examples that the model misclassifies, yielding an actual success rate of 9.85%. On the CTU-13 dataset, however, the attack fails to generate any constrained adversarial examples. The reason is that CTU-13 comprises 360 constraints, which translates into as many new terms in the function of which CPGD backpropagates through. As each function contributes with diverse, non-co-linear, or even opposed gradients, this ultimately hinders the attack. Similar phenomena have been observed in multi-label [SJH+18] and multi-task models [GCP+21]. By contrast, MOEVA, which enables a global exploration of the search space, is successful for 97.48% and 100% of the original examples, respectively.

MOEVA also manages to create feasible adversarial examples on the random forest models, with a success rate ranging from 5.41% to 41.51%. This indicates that our attack remains effective on such ensemble models, including with other datasets. Like PGD, the Papernot attack – unaware of constraints – cannot produce a single feasible example on LCLD, CTU-13, and Malware, whereas it has a low success rate (8.50%) on URL compared to MOEVA (31.89%).

Conclusion: While adversarial attacks unaware of domain constraints fail, incorporating constraint knowledge as an attack objective enables the successful generation of constrained adversarial examples.

4.6.2 Adversarial Retraining

We, next, evaluate if adversarial retraining is an effective means of reducing the effectiveness of constrained attacks.

We start with our models trained on the original training set. We generate constrained adversarial examples (using either CPGD or MOEVA) from original training examples that each model correctly classifies in class 1. To enable a fair comparison of both methods, for the LCLD (NN), we use only the original examples for which CPGD and MOEVA could both generate a successful adversarial example. In all other cases, we do not apply this restriction, since MOEVA is the only technique that is both applicable and successful. While MOEVA returns a set of constrained examples, we only select the individual that maximizes the confidence of the model in its (wrong) classification, similar to CPGD which maximizes the model loss.

Table 4.3: Success rate of CPGD and MOEVA after adversarial retraining and constraint augmentation (on neural networks). For a fair comparison, the model denoted by the same symbols (* or †) are trained with the same number of adversarial examples, generated from the same original samples.

Defense	Attack	LCLD	CTU-13
None	CPGD	9.85	0.00
	MOEVA	97.48	100.00
CPGD Adv. retraining *	CPGD	8.78	NA
	MOEVA	94.90	NA
MOEVA Adv. retraining *	CPGD	2.70	NA
	MOEVA	85.20	0.8
Constraints augment.	CPGD	0.00	NA
	MOEVA	80.43	0.00
MOEVA Adv. retrain. †	MOEVA	82.00	NA
Combined mechanisms †	MOEVA	77.43	NA

Table 4.4: Success rate of MOEVA on the random forest models.

Defense	LCLD	CTU-13	Malware	URL
None	41.51	5.41	39.30	31.89
Adv. retraining	3.90	4.67	37.69	22.14
Cons. augment.	19.73	6.63	28.52	20.99
Combined	0.77	4.67	28.98	15.94

Regarding the perturbation budget, we follow established standards [CAP+19] and provide the attack with an ϵ budget 4 times larger than the defense.

Table 4.3 (middle rows) shows the results for the neural networks, and Table 4.4 (second row) for the random forests. Overall, we observe that adversarial retraining remains an effective defense against constrained attacks. For instance, on LCLD (NN) adversarial retraining using MOEVA drops the success rate of CPGD from 9.85% to 2.70%, and its own success rate from 97.48% to 85.20%. The fact that MOEVA still works suggests, however, that the large search space that this search algorithm explores preserves its effectiveness. By contrast, on CTU-13 (NN), we observe that the success rate of MOEVA drops from 100% to 0.8% after adversarial retraining with the same attack.

We assess the effect of adversarial retraining more finely and show, in Figure

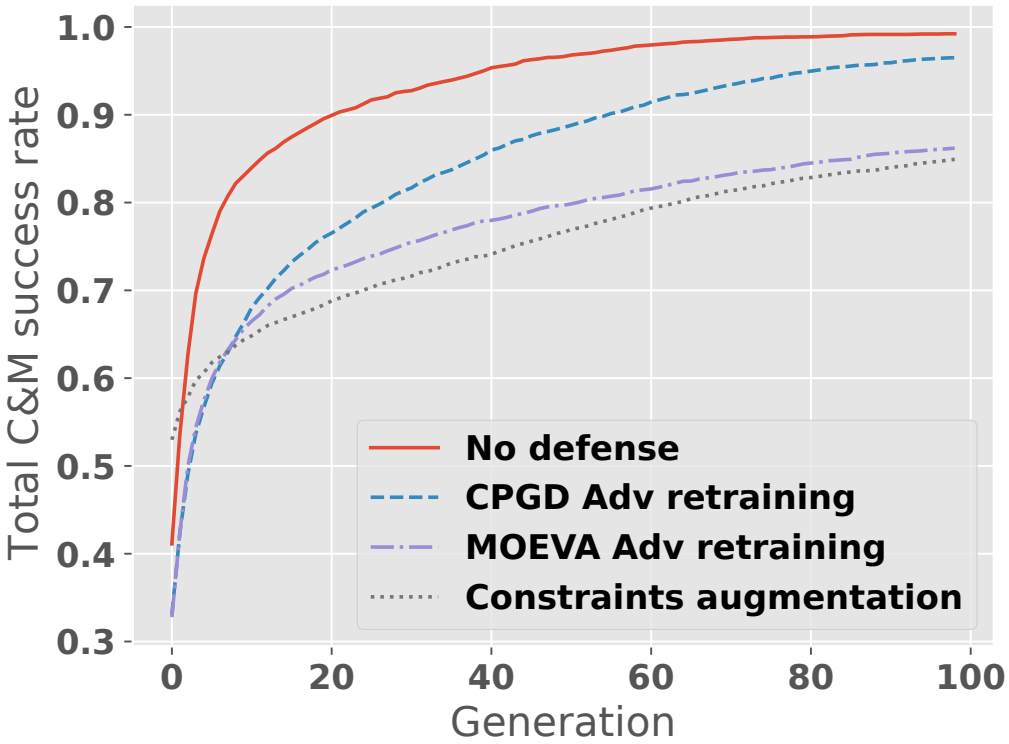


Figure 4.1: Success rate of MOEVA against the original LCLD neural network, the defended counterparts, and constraints engineering, over the generations.

4.1, the success rate of MOEVA on the LCLD neural network over the number of generations. Compared to the undefended model, adversarial retraining (using MOEVA) lowers the asymptotic success rate. Moreover, the growth of the success rate is much steeper for the undefended model. For instance, MOEVA needs ten times more generations to reach the same success rate of 84% against the MOEVA defended models than against the undefended model (100 generations versus 12 generations). Adversarial retraining using CPGD is less effective: while it reduces the success rate in the earlier generations, its benefits diminishes as the MOEVA attack runs for more generations.

Conclusion: Adversarial retraining remains an effective defense against constrained adversarial attacks. The benefits of adversarial retraining against MOEVA are achieved as soon as the earliest generations and persist throughout the attack process.

4.6.3 Impact of Constraints Engineering

We hypothesize that constraints – in particular, non-convex constraints can hinder the generation of adversarial examples. To verify this hypothesis, we propose a systematic method to add engineered constraints, and we evaluate the effectiveness of our attack against this augmented dataset.

To define new constraints, we first augment the original data with new features engineered from existing features. Let $\hat{f}$ denote the mean value of some feature of interest f over the training set. Given a pair of feature (f_1, f_2), we engineer a binary feature f_e as

$$f_e(x) \equiv (\hat{f}_1 \leq x_1) \oplus (\hat{f}_2 \leq x_2) \tag{4.6}$$

where x_i is the value of f_i in x and $\oplus$ denotes the exclusive or (XOR) binary operator. The comparison of the value of the input x for a particular feature f_i with the mean $\hat{f}_i$ allows us to handle non-binary features while maintaining a uniform distribution of value across the training dataset. We use the XOR operator to generate the new feature as this operator is not differentiable. We, then, introduce a new constraint that the value of f_e should remain equal to its original definition. That is, we add the constraint

$$\omega_e \equiv (f_e(x) = (\hat{f}_1 \leq x_1) \oplus (\hat{f}_2 \leq x_2)) \tag{4.7}$$

The original examples respect these new constraints by construction. In other words, for an adversarial attack to be successful, the attack should modify f_e if the modifications it applied to f_1 and f_2 would imply a change in the value of f_e.

To avoid combinatorial explosion, we add constraints only on pairs of the most important mutable features. We measure importance with the approximation

of Shapley value [SGK17], an established explainability method. In the end, we consider a number M of pairs such that $M = \arg\max_x \binom{x}{2} \leq \frac{N}{4}$ where N is the total number of features.

As a preliminary sanity check, we verified that constraint augmentation does not penalize clean performance and confirmed that the augmented models keep similar performance.

We evaluate our attacks following the same protocol, except that the models are trained on the augmented set of features. That is, we assume that the attacker has knowledge of the added features and constraints.

We show the results in Table 4.3 and 4.4 (third row). Constrained augmentation nullifies the low success rate of CPGD on LCLD – the gradient-based attack becomes unable to satisfy the constraints. Constraints augmentation also decreases significantly the success rate of MOEVA in all cases except the CTU-13 random forest. For instance, it drops from 97.48% to 80.43% for LCLD NN, and from 100% to 0% on CTU-13 NN.

In Figure 4.1, we assess the effect of constraints augmentation on the success rate of MOEVA on the LCLD neural network over the number of generations Compared to the original model, constrained augmentation lowers the success rate. The growth of success rate is much slower for the clean constrained engineered model.

Conclusion: Constraint augmentation hinders the generation of constrained adversarial attacks, as early as the first generations.

4.6.4 Combining mechanisms

We investigate whether the combination of constraint augmentation with adversarial retraining yields a decrease in MOEVA performance. A positive answer would indicate that constraint augmentation and adversarial retraining have complementary benefits.

We add to the models the same engineered constraints as we did previously. We also perform adversarial retraining on the augmented models, using all adversarial examples that MoEvA managed to generate on the training set. Then, we attack the defended models using MoEvA applied to the test set. For a fair comparison with adversarial retraining, we also apply this defense without constraint augmentation, using the same number of examples. We do not experiment with CPGD, which was already ineffective when only one defense was used. Neither do we consider the datasets for which one defense was enough to fully protect the model.

Tables 4.3 and 4.4 (last rows) present the results. On LCLD (NN), the combined mechanisms drop the attack success rate from 97.48% (on a defenseless model) to 77.23%, which is better than adversarial retraining (82.00%) and constraint

augmentation (80.43%) applied separately. On the RFs, the combination either offers additional reductions in attack success rate compared to the best individual defense (LCLD and URL) or has negligible effects (CTU-13 and Malware).

Conclusion: Constraint augmentation and adversarial training are complementary. Compared to their separate application, the combination can decrease the attack success rate by up to 5%.

4.7 Conclusion

In this chapter, we proposed the first generic framework for adversarial attacks under domain-specific constraints. We instantiated our framework with two methods: one gradient-based method that extends PGD with multi-loss gradient descent, and one that relies on multi-objective search. We evaluated our methods on four datasets and two types of models. We demonstrated their unique capability to generate constrained adversarial examples. We would like to encourage the evaluation of the robustness of machine learning models deployed in critical software systems. With this work, we enable practitioners to correctly evaluate the robustness of their models by generating realistic adversarial examples (i.e. that respect domain constraints). In addition to adversarial retraining, we proposed and investigated a novel mechanism that introduces engineered non-convex constraints. This strategy is as effective as adversarial retraining. Although adaptive attacks could potentially breach this approach, it still demonstrates the capacity of constraints to hinder adversarial attacks. We hope that our approach, algorithms, and datasets will be the starting point of further endeavors toward studying feasible adversarial examples in real-world domains that are inherently constrained. In particular, we explore in the next chapter, improvements to gradient-based attacks and the construction of a strong and effective attack that combines fast gradient attacks with effective search attacks.

5

Constrained Adaptive Attack: Effective Adversarial Attack Against Deep Neural Networks for Tabular Data

The reliable evaluation of Deep Neural Networks' robustness requires strong and efficient attacks. We propose Constrained Adaptive PGD (CAPGD) a fast gradient-based attack that is complementary to MOEVA. By combining these two attacks, we obtain Constrained Adaptive Attack a strong attack subsuming all other constrained attacks while being up to five times faster.

Contents

5.1 Introduction

This chapter continues to address the first challenge of this thesis which is to evaluate the robustness of critical systems against adversarial examples. Chapter 4 introduced a framework to generate constrained adversarial examples that we instantiated with two attacks CPGD and MOEVA. While we empirically showed the effectiveness of our framework and attacks, the success rate of CPGD remains low (at most 9.85%) and MOEVA remains computationally expensive.

While research has studied the robustness of deep learning models in Computer Vision (CV) and Natural Language Processing (NLP) tasks, many real-world applications instead deal with tabular data, including in critical fields like finance, energy, and healthcare. If classical "shallow" models (e.g. random forests) have been the go-to solution to learn from tabular data [HK20], deep learning models are becoming competitive [BLS+22]. This raises anew the need to study the robustness of these models.

However, robustness assessment for tabular deep learning models brings a number of new challenges that previous solutions - because they were originally designed for CV or NLP tasks - do not consider. One such challenge is the fact that tabular data exhibit *feature constraints*, i.e. complex relationships and constraints across features. The satisfaction of these feature constraints can be a non-convex or even non-differentiable problem; this implies that established evasion attack algorithms relying on gradient computation do not create valid adversarial examples (i.e., constraint satisfying) [GCG+20]. Meanwhile, attacks designed for tabular data also ignore feature type constraints [BAL+19] or, in the best case, consider categorical features without feature relationships [WHB+20; XHR+23; BHZ+23] and are evaluated on datasets that exclusively contain such features. This restricts their application to other domains that present heterogeneous feature types.

As of today, *Constrained Projected Gradient Descent* (CPGD) and *Multi-Objective Evolutionary Adversarial Attack* (MOEVA) remain the only published evasion attacks that support feature constraints [SDG+22]. CPGD is an extension of the classical gradient-based PGD attack with a new loss function that encodes how far the generated examples are from satisfying the constraints. Although theoretically elegant and practically efficient, this attack suffers from a low success rate due to its difficulty in converging toward both model classification and constraint satisfaction [SDG+22]. Conversely, MOEVA is based on genetic algorithms. It offers an outstanding success rate compared to CPGD and works on shallow and deep learning models. However, it is computationally expensive and requires numerous hyper-parameters to be tuned (population size, mutation rate, generations, etc.). This prevents this attack from scaling to larger models and datasets.

Overall, research on adversarial robustness for tabular machine learning in general (and tabular deep learning in particular) is still in its infancy. This is in

stark contrast to the abundant literature on adversarial robustness in CV [LGX+22] and NLP tasks [DGC23]. Given this limited state of knowledge, the objective of this chapter is to propose novel and effective attack methods for tabular models subject to feature constraints.

We hypothesize that gradient-based algorithms have not been explored adequately in Chapter 4 and that the introduction of dedicated adaptive mechanisms can outperform CPGD. To verify this, we design a new adaptive attack, named *Constrained Adaptive PGD* (CAPGD), whose only free parameter is the number of iterations and that does not require additional parameter tuning (Section 5.3). We demonstrate that the different mechanisms we introduced in CAGPD contribute to improving the success rate of this attack compared to CPGD, by 81% points. Across all our datasets, the set of adversarial examples that CAPGD generates subsumes all the examples generated by any other gradient-based method. Furthermore, CAPGD is 75 times faster than MOEVA, while the latter reaches the highest success rate across all datasets.

These results motivate us to design *Constrained Adaptive Attack* (CAA), an adaptive attack that combines our new gradient-based attack (CAPGD) with MOEVA for an increased success rate at a lower computational cost. Our experiments show that CAA reaches the highest success rate for all models/datasets we considered, except in one case where CAA is second-best. With this attack, we offer a strong baseline for future research on evasion attacks for tabular models, which should become the minimal test for robust tabular architectures and other defense mechanisms.

Our contributions can be summarized as follows:

1. We design a new parameter-free attack, CAPGD that introduces momentum and adaptive steps to effectively evade tabular models while enforcing the feature constraints. We show that CAPGD outperforms the other gradient-based attacks in terms of the capability to generate valid (constraint-satisfying) adversarial examples.

2. We propose a new efficient and effective evasion attack (CAA) that combines gradient and search attacks to optimize both effectiveness and computational cost.

3. We evaluate CAA in a large-scale evaluation over four datasets, five architectures, and two training methods (standard and adversarial training). Our results show that CAA outperforms all other attacks and is up to 5 times more efficient.

5.2 Problem formulation

We formulate the problem of evasion attacks under constraints. We assume the attack to be untargeted (i.e. it aims to force misclassification in any incorrect class); the formulation for targeted attacks is similar and omitted for space reasons.

We denote by $x \in \mathbb{R}^d$ an input example and by $y \in \{1, \ldots, C\}$ its correct label. Let $h : \mathbb{R}^d \to \mathbb{R}^C$ be a classifier and $h_{c_k}(x)$ the classification score that h outputs for input x to be in class c_k. Let $\Delta \subseteq \mathbb{R}^d$ be the space of allowed perturbations. Then, the objective of an evasion attack is to find a $\delta \in \Delta$ such that $argmax_{c \in \{1,\ldots,C\}} h_c(x + \delta) \neq y$.

In image classification, the set Δ is typically chosen as the perturbations within some l_p-ball around x, that is, $\Delta_p = \{\delta \in \mathbb{R}^d, ||\delta||_p \leq \epsilon\}$ for a maximum perturbation threshold ϵ. This restriction aims at preserving the semantics of the original input by assuming that small enough perturbations will yield images that humans perceive the same as the original images and would therefore classify the perturbed input into the same class (while the classifier predicts another class). This also guarantees that the example remains meaningful, that is, $x + \delta$ is not an image with random noise.

Tabular data are by nature different from images. They typically represent objects of the considered application domain (e.g. botnet traffic [CO22], financial transaction [GCG+20]). We denote by $\varphi : Z \to \mathbb{R}^d$ the feature mapping function that maps objects of the problem space Z to a d-dimensional feature space defined by the feature set $F = \{f_1, f_2, ...f_d\}$. Each object $z \in Z$ must inherently respect some natural condition to be valid (to be able to exist in reality). In the feature space, these conditions translate into a set of constraints on the feature values, which we denote by Ω. By construction, any input example x obtained from a real-world object z satisfies Ω, noted $x \models \Omega$.

Thus, in the case of tabular data, we additionally require the perturbation δ applied to x to yield a valid example $x + \delta$ satisfying Ω, that is, $\Delta_p(x) = \{\delta \in \mathbb{R}^d : ||\delta||_p \leq \epsilon \wedge x + \delta \models \Omega\}$.

To define the constraint language expressing Ω, we consider the four types of constraint introduced in Chapter 4. These four constraint types cover all the constraints of the datasets in our empirical study. Hence, *immutability* defines what features cannot be changed by an attacker; *boundaries* defines upper / lower bounds for feature values; *type* specifies a feature to take continuous, discrete, or categorical values; and *feature relationships* capture numerical relations between features. These four types of constraints can be expressed with the following grammar:

$$\begin{aligned} \omega &:= \omega_1 \wedge \omega_2 \mid \omega_1 \vee \omega_2 \mid \psi_1 \succeq \psi_2 \\ \psi &:= c \mid f \mid \psi_1 \oplus \psi_2 \mid x_i \end{aligned} \tag{5.1}$$

where $f \in F$ is a feature, c is a constant, $\omega, \omega_1, \omega_2$ are constraint formulae, $\succeq \in \{<, \leq, =, \neq, \geq, >\}$ is a comparison operator, $\psi, \psi_1, \ldots, \psi_k$ are numeric expressions, $\oplus \in \{+, -, *, /\}$ is a numerical operator, and x_i is the value of the i-th feature of the original input x.

5.2.1 Constrained Projected Gradient Descent

Constrained Projected Gradient Descent (CPGD) is an extension of the PGD attack [MMS+17] to generate adversarial examples satisfying constraints in tabular machine learning. It integrates constraint satisfaction into the loss function that PGD optimizes. This is achieved by translating each constraint ω into a differentiable function $penalty(x, \omega)$ that values to zero if $x \models \omega$; otherwise, the function represents how far x is from satisfying ω. We use the translation table of Chapter 4 to translate each construct of the constraint grammar into a penalty function.

Based on this, CPGD produces adversarial examples from an initial sample x_{orig} classified as y by iteratively computing:

$$x^{(k+1)} = R_\Omega(P_\mathcal{S}(x^{(k)} + \eta^{(k)} \nabla \mathcal{L}(x^{(k)}, y, h, \Omega))) \tag{5.2}$$

where $x^0 = x_{orig}$ (the original input), $P_\mathcal{S}$ is the projection onto $\mathcal{S} = \{x \in \mathbb{R}^d, ||x - x_{orig}||_p \leq \epsilon\}$, $\nabla \mathcal{L}$ is the gradient of loss function $\mathcal{L}$, defined as

$$\mathcal{L}(x, y, h, \Omega) = l(h(x), y) - \sum_{\omega_i \in \Omega} penalty(x, \omega_i). \tag{5.3}$$

In the original CPGD implementation, the step size $\eta^{(k)}$ follows a predefined decay schedule, $\eta^{(k)} = \epsilon \times 10^{-(1+\lfloor k/\lfloor K/M \rfloor \rfloor)}$, with $M = 7$, and $K = max(k)$. $\mathcal{L}'(x)$ abbreviates $\mathcal{L}(x, y, h, \Omega)$.

5.2.2 Experimental settings

Our experiments are driven by the following datasets, models, and attack parameters. More details about the datasets and models are given in Appendix 10.2.1.

Datasets To conduct our study, we selected tabular datasets that present feature constraints from their respective application domain. **URL** [HY21] is a dataset of legitimate and phishing URLs. With only 14 linear domain constraints and 63 features, it is the simplest of our empirical study. **LCLD** [Geo18] is a credit-scoring dataset with non-linear constraints. The **WiDS** [LRG+20] dataset contains medical data on the survival of patients admitted to the ICU. It has only 30 linear domain constraints. The **CTU** [CO22] dataset reports legitimate and botnet traffic from CTU University. The challenge of this dataset lies in its large number of linear domain constraints (360). We detail the datasets in the Appendix 10.2.1.

Architectures We evaluate five top-performing architectures from a recent survey on tabular ML [BLS+22]: **TabTransformer** [HKC+20] and **TabNet** [AP21] are transformer-based models. **RLN** [SS18] uses a regularization coefficient to minimize a counterfactual loss. **STG** [YLN+20] optimizes feature selection with stochastic gates, and **VIME** [YZJ+20] relies on self-supervised learning. These architectures achieve performance equivalent to XGBoost, the best shallow machine learning model for our use cases.

Perturbation parameters We use the L2-norm to measure the distance between original and perturbed inputs, because this norm is suitable for both numerical and categorical features. We set ϵ to 0.5 for all datasets. Each dataset has a critical (negative) class, respectively phishing URLs, rejected loans, flagged botnets, and not surviving patients. Hence, we only attack clean examples from the critical class that are not already misclassified by the model and report robust accuracy of models.

Evaluation metrics We measure the effectiveness of our attack using robust accuracy defined as the accuracy of valid examples generated by a given attack. If a clean example is misclassified, we do not perturb it. If the attack generates an invalid example, we consider it as correctly classified. We measure the efficiency of the attacks in computational time.

5.3 Our Constrained *Adaptive* PGD

The relative lack of effectiveness of CPGD as reported in its original publication leads us to investigate the cause of these weaknesses. We investigate four factors that may affect the success rate of the attack: (1) we conjecture that the fixed step size and predefined decay in CPGD might be suboptimal because the choice of the step size is known to largely impact the effectiveness of gradient-based attacks [MAT+18]; (2) CPGD is unaware of the trend, i.e. it does not consider whether the optimization is evolving successfully and is not able to react to it; (3) CPGD does not check constraint satisfaction between the iterations, which could "lock" the algorithm into a part of the invalid data space; (4) CPGD starts with the original example, whereas classical gradient-based attacks often benefit from random initialization.

5.3.1 CAPGD algorithm

We propose Constrained Adaptive PGD (CAPGD), a new constraint-aware gradient-based attack that aims to overcome the limitations of CPGD and improve its effectiveness. We detail CAPGD in Algorithm 1 in Appendix 10.2.1, and summarize its components below.

Step size selection We introduce a step-size adaptation. We follow the exploration-exploitation principle by gradually reducing the gradient step [CH20]. However, unlike CPGD, this reduction does not follow a fixed schedule but is determined by the optimization trend. If the value of the loss function grows, we keep the same step size; otherwise, we halve it. That is, we start with a step $\eta^{(0)} = 2\epsilon$, and we identify checkpoints $w_0 = 0, w_1, ..., w_n$ at which we decide whether it is necessary to halve the size of the current step. We halve the step size if any of the following two conditions holds:

1. Since the last checkpoint, we count how many cases since the last checkpoint w_{j-1} the update step has successfully increased $\mathcal{L}'$. The condition holds if the loss has increased for at least a fraction of ρ steps (we set $\rho = 0.75$):

$$\sum_{i=w_{j-1}}^{w_j-1} \mathbf{1}_{\mathcal{L}'(x^{(i+1)})>\mathcal{L}'(x^{(i)})} < \rho \cdot (w_j - w_{j-1}). \tag{5.4}$$

2. The step has not been reduced at the last checkpoint and the loss is less or equal to the loss of the last checkpoint:

$$\eta^{(w_{j-1})} \equiv \eta^{(w_j)} \wedge \mathcal{L}_{\max}^{(w_{j-1})} \equiv \mathcal{L}_{\max}^{(w_j)} \tag{5.5}$$

where $\mathcal{L}'(x)$ is the loss function, $\mathcal{L}_{\max}^{(w_j)}$ is the highest loss value in the first $j+1$ iterations.

Repair operator We also introduce a new "repair" operator denoted R_Ω that projects back the example produced at each iteration into the valid data space. The idea is to force the value of any feature f that occurs in constraints of the form $f = \psi$ (see Equation 5.1) to be ψ valued based on all other feature values in the example.

Initial state As for initialization, we apply the attack from two initial states: the original example x_{orig} and a random example sampled from $\mathcal{S}$ (the Lp-ball around x_{orig}). The goal behind this second initialization is to reduce the risk of being immediately locked into local optima that encompass only invalid examples. Our experiments later reveal the complementary of these two initializations.

Gradient step Finally, we introduce in CAPGD a momentum [DLP+18]. Let $\eta^{(k)}$ be the step size at iteration k, then we first compute $z^{(k+1)}$ before the updated example $x^{(k+1)}$.

$$\begin{aligned} z^{(k+1)} &= P_{\mathcal{S}}(x^{(k)} + \eta^{(k)}(\nabla\mathcal{L}'(x^{(k)})) \\ x^{(k+1)} &= R_\Omega(P_{\mathcal{S}}(x^{(k)} + \alpha \cdot (z^{(k+1)} - x^{(k)}) + (1-\alpha) \cdot (x^{(k)} - x^{(k+1)})) \end{aligned} \tag{5.6}$$

where $\alpha \in [0, 1]$ (we use $\alpha = 0.75$ following [CH20]) regulates the influence of the previous update on the current, and $P_{\mathcal{S}}$ is the projection onto $\mathcal{S} = \{x \in \mathbb{R}^d, ||x - x_{orig}||_p \leq \epsilon\}$.

Table 5.1: Robust accuracy against CAPGD and SOTA gradient attacks. A lower robust accuracy means a more effective attack (lowest in bold).

Dataset	Model	Clean	LPF	CPGD	CAPGD
URL	TabTr.	93.6	93.6	91.9	**10.9**
	RLN	94.4	94.4	92.8	**12.6**
	VIME	92.5	92.5	90.7	**56.3**
	STG	93.3	93.3	93.3	**72.6**
	TabNet	93.4	93.4	88.5	**19.3**
LCLD	TabTr.	69.5	69.2	69.5	**27.1**
	RLN	68.3	68.3	68.3	**0.2**
	VIME	67.0	67.0	67.0	**2.6**
	STG	66.4	66.4	66.4	**55.5**
	TabNet	67.4	67.4	67.4	**6.3**
CTU	TabTr.	**95.3**	**95.3**	**95.3**	**95.3**
	RLN	**97.8**	**97.8**	**97.8**	**97.8**
	VIME	**95.1**	**95.1**	**95.1**	**95.1**
	STG	**95.3**	**95.3**	**95.3**	**95.3**
	TabNet	**96.1**	**96.1**	**96.1**	**96.1**
WIDS	TabTr.	75.5	75.5	75.2	**48.0**
	RLN	77.5	77.5	77.3	**61.8**
	VIME	72.3	72.3	71.5	**51.4**
	STG	77.6	77.6	77.5	**65.1**
	TabNet	79.7	79.7	76.0	**10.2**

5.3.2 Comparison of CAPGD to gradient-based attacks

To evaluate the benefits of CAPGD, we compare it with CPGD as well as LowProFool, the only other public *gradient* attack for tabular models that can be extended to support all types of features.

CAPGD is more successful than existing gradient attacks. We compare the robust accuracy across our four datasets and five architectures with CPGD, LowProFool, and CAPGD, and report the results in Table 5.1. CAPGD significantly outperforms CPGD and LowProFool. It decreases the robust accuracy on URL, LCLD, and WIDS datasets to as low as 10.9%, 0.2%, and 10.2% respectively.

The results also reveal that gradient attacks are ineffective on the CTU dataset. These results demonstrate that gradient-based attacks are not enough and motivate us to consider combining CAPGD with search-based attacks, as investigated in Section 5.4.

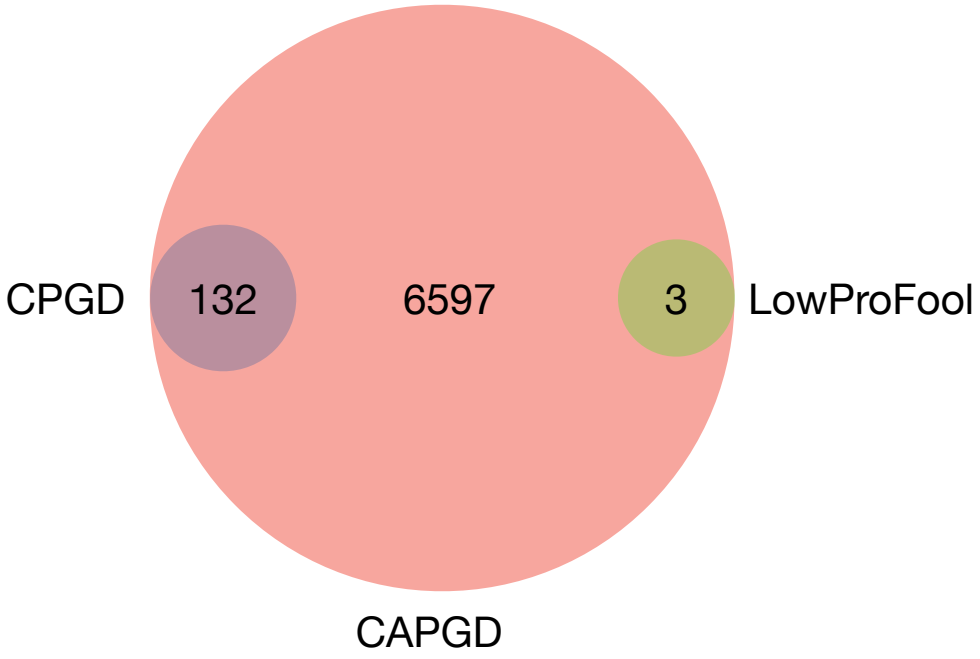


Figure 5.1: Visualization of the complementarity of CAPGD, CPGD, and LowProFool with the number of successful adversarial examples.

CAPGD subsumes all gradient attacks. We analyze in detail the original examples from which attacks could generate valid and successful adversarial examples. For each attack, we take the union of the sets of clean examples across 5 seeds. We generate the Venn diagram for CPGD, LowProFool, and CAPGD, for all datasets and model architectures. We sum the partition values in Figure 5.1. CAPGD generates adversarial examples for 6597 original examples from which none of the other gradient attacks could produce adversarial examples. In contrast, all successful adversarial examples by CPGD (132) and LowProFool (3) are also generated by CAPGD.

5.3.3 Components of CAPGD

All components of CAPGD contribute to its effectiveness. We conduct an ablation study on the components of CAPGD. We evaluate CAPGD without the repair operator at each iteration (NREP), without the initialization with clean example (NINI), without the initialization with random sampling (NRAN), and without the adaptive step (NADA). Table 5.2 reveals that removing a component of CAPGD reduces its effectiveness. Interestingly, not all components affect all datasets similarly. For instance, removing the repair at each gradient iteration only affects the LCLD datasets' success rate. For URL and WIDS, CAPGD-NREP remains in the confidence interval of CAPGD. Removing any other components always negatively affects CAPGD, up to an improvement of 32.1% of the robust accuracy for CAPGD-NADA on the WIDS dataset and TabNet model.

CAPGD components are complementary. None of the components of CAPGD negatively affects its capability of finding an adversarial example for a given clean example. In Figure 5.2, we analyze the coverage of each CAPGD variant A (Covered Attack) with regard to another variant B (Covering Attack). For A and B, we compute the set of clean examples C_A and C_B on which the attacks are successful. The percentage in the heatmaps represents the proportion of $C_A \cup C_B$ covered by C_B, that is

$$coverage = \frac{|C_B|}{|C_A \cup C_B|}$$

where $|C|$ is the cardinality of C. Attack B subsumes A when $coverage = 1$. In practice, to avoid random effect, we run the attack for $N = 5$ seeds, and take the *union* of clean examples on which we generate a successful example.

Figure 5.2 reveals that CAPGD subsumes all its variants with coverage over 99%, while none of the variants subsumes CAPGD. Therefore all components of CAPGD are necessary to obtain the strongest attack.

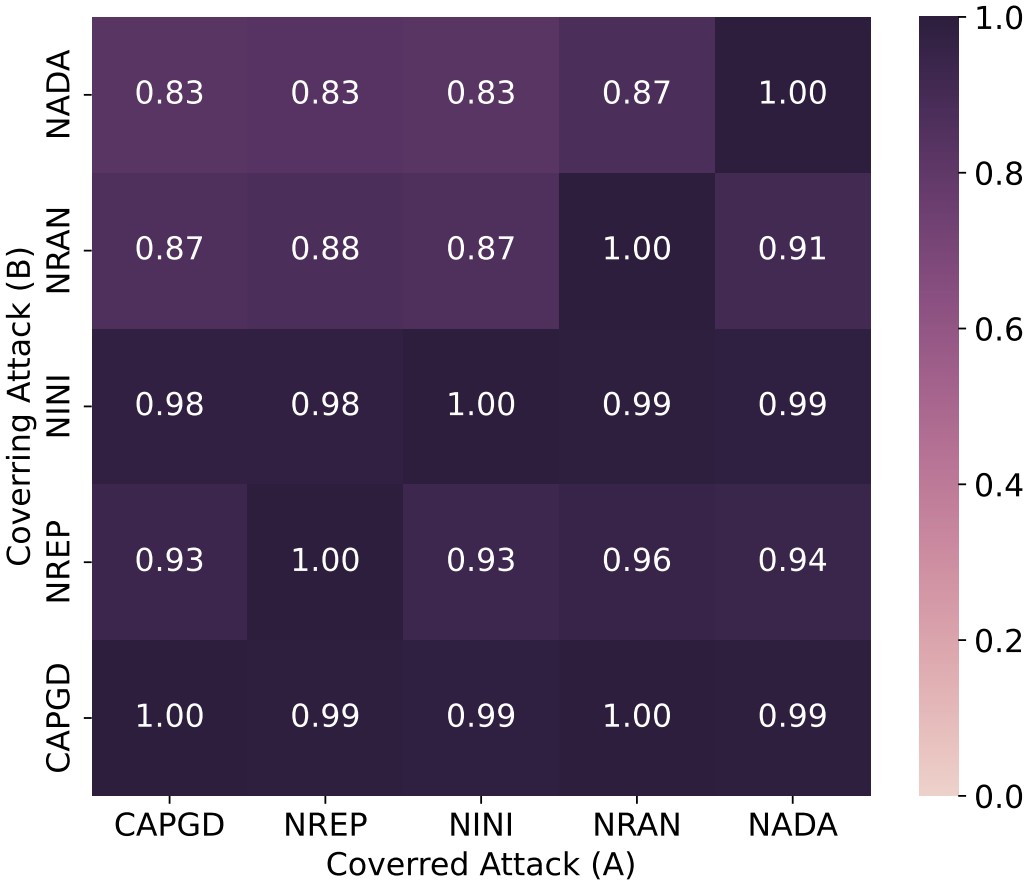


Figure 5.2: Visualization of the utility of CAA's components. For attack A (respectively B) we compute the set of clean examples C_A (respectively C_B)) on which the attack is successful. The percentage represents the proportion of the set $C_A \cup C_B$ is covered by C_B. CAPGD-NADA is CAPGD without an adaptive step, CAPGD-NRAN is CAPGD without the random start, CAPGD-NINI is CAPGD without the clean example initialization and CAPGD NREP is CAPGD without repair at each iteration.

Table 5.2: Ablation study: Robust accuracy for CAPGD and its variant without key components. The Clean column corresponds to the accuracy of the model on the subset of clean samples that we attack. A lower robust accuracy means a more effective attack. The lowest robust accuracy is in bold.

Dataset	Model	Clean	NREP	NINI	NRAN	NADA	CAPGD
URL	TabTr.	93.6	**10.9**$_{\pm 0.1}$	11.8$_{\pm 0.3}$	12.6$_{\pm 0.0}$	34.6$_{\pm 0.4}$	**10.9**$_{\pm 0.1}$
	RLN	94.4	**12.7**$_{\pm 0.2}$	14.9$_{\pm 0.2}$	14.8$_{\pm 0.0}$	30.2$_{\pm 0.5}$	**12.6**$_{\pm 0.2}$
	VIME	92.5	**56.3**$_{\pm 0.1}$	58.1$_{\pm 0.3}$	56.9$_{\pm 0.0}$	65.2$_{\pm 0.1}$	**56.3**$_{\pm 0.1}$
	STG	93.3	**72.6**$_{\pm 0.0}$	73.4$_{\pm 0.2}$	73.0$_{\pm 0.0}$	75.3$_{\pm 0.1}$	**72.6**$_{\pm 0.0}$
	TabNet	93.4	**19.2**$_{\pm 0.7}$	27.6$_{\pm 0.8}$	29.7$_{\pm 0.0}$	34.4$_{\pm 0.3}$	**19.3**$_{\pm 0.6}$
LCLD	TabTr.	69.5	38.3$_{\pm 0.4}$	38.0$_{\pm 0.9}$	38.0$_{\pm 0.0}$	44.4$_{\pm 1.1}$	**27.1**$_{\pm 0.9}$
	RLN	68.3	5.3$_{\pm 0.2}$	1.4$_{\pm 0.3}$	1.6$_{\pm 0.0}$	1.1$_{\pm 0.3}$	**0.2**$_{\pm 0.1}$
	VIME	67.0	17.9$_{\pm 0.6}$	7.1$_{\pm 0.5}$	7.3$_{\pm 0.0}$	3.6$_{\pm 0.4}$	**2.6**$_{\pm 0.2}$
	STG	66.4	59.4$_{\pm 0.1}$	58.0$_{\pm 0.3}$	56.5$_{\pm 0.0}$	59.7$_{\pm 0.2}$	**55.5**$_{\pm 0.2}$
	TabNet	67.4	30.4$_{\pm 0.6}$	33.1$_{\pm 1.4}$	10.8$_{\pm 0.0}$	7.3$_{\pm 0.4}$	**6.3**$_{\pm 0.4}$
CTU	TabTr.	**95.3**	**95.3**$_{\pm 0.0}$	**95.3**$_{\pm 0.0}$	**95.3**$_{\pm 0.0}$	**95.3**$_{\pm 0.0}$	**95.3**$_{\pm 0.0}$
	RLN	**97.8**	**97.8**$_{\pm 0.0}$	**97.8**$_{\pm 0.0}$	**97.8**$_{\pm 0.0}$	**97.8**$_{\pm 0.0}$	**97.8**$_{\pm 0.0}$
	VIME	**95.1**	**95.1**$_{\pm 0.0}$	**95.1**$_{\pm 0.0}$	**95.1**$_{\pm 0.0}$	**95.1**$_{\pm 0.0}$	**95.1**$_{\pm 0.0}$
	STG	**95.3**	**95.3**$_{\pm 0.0}$	**95.3**$_{\pm 0.0}$	**95.3**$_{\pm 0.0}$	**95.3**$_{\pm 0.0}$	**95.3**$_{\pm 0.0}$
	TabNet	**96.1**	**96.1**$_{\pm 0.0}$	**96.1**$_{\pm 0.0}$	**96.1**$_{\pm 0.0}$	**96.1**$_{\pm 0.0}$	**96.1**$_{\pm 0.0}$
WIDS	TabTr.	75.5	**48.2**$_{\pm 0.3}$	54.7$_{\pm 0.9}$	53.3$_{\pm 0.0}$	64.9$_{\pm 0.6}$	**48.0**$_{\pm 0.3}$
	RLN	77.5	**61.8**$_{\pm 0.3}$	66.3$_{\pm 0.4}$	63.7$_{\pm 0.0}$	72.3$_{\pm 0.5}$	**61.8**$_{\pm 0.3}$
	VIME	72.3	**51.4**$_{\pm 0.3}$	55.6$_{\pm 0.9}$	54.1$_{\pm 0.0}$	62.9$_{\pm 0.4}$	**51.4**$_{\pm 0.3}$
	STG	77.6	**65.1**$_{\pm 0.4}$	68.9$_{\pm 0.3}$	67.6$_{\pm 0.0}$	74.0$_{\pm 0.2}$	**65.1**$_{\pm 0.4}$
	TabNet	79.7	**10.1**$_{\pm 0.5}$	20.7$_{\pm 0.9}$	17.5$_{\pm 0.0}$	42.3$_{\pm 0.7}$	**10.2**$_{\pm 0.3}$

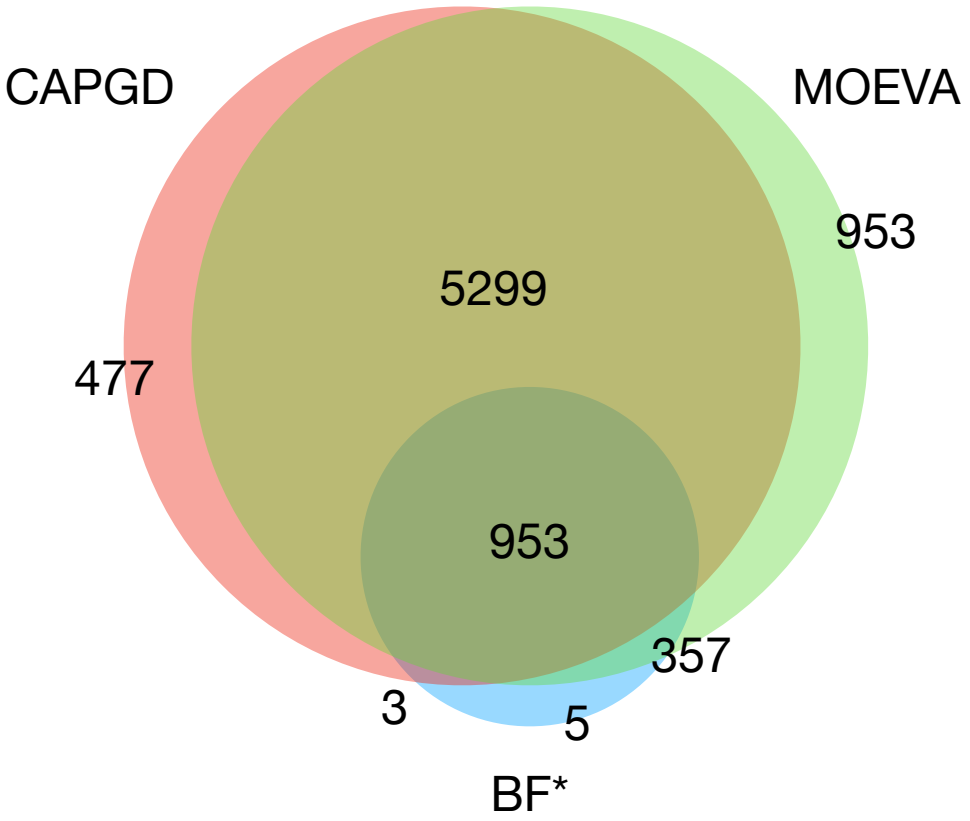


Figure 5.3: Visualization of the complementarity of CAPGD, MOEVA, and BF* with the number of successful adversarial examples.

5.4 CAA: an ensemble of gradient and search attacks

We next propose *Constrained Adaptive Attack* (CAA), an effective and efficient ensemble of gradient- and search-based attacks. The idea underlying CAA is that gradient-based attacks for tabular data are more efficient but less successful than search-based attacks. Thus, CAA integrates the best search-based attacks from each family in a complementary way, such that we maximize the set of adversarial examples that can be generated.

5.4.1 Design of CAA

Following our related work study, we consider the search attacks MOEVA and BF* [KHS+18; KKT23]. As a first step, we compare in Figure 5.3 these two

Table 5.3: Robust accuracy for CAPGD, MOEVA, and CAA. The Clean column corresponds to the accuracy of the model on the subset of clean samples that we attack. A lower robust accuracy means a more effective attack. The lowest robust accuracy is in bold.

		Robust accuracy (↓)				
Dataset	Model	Clean	CAPGD	BF*	MOEVA	CAA
URL	TabTr.	93.6	$10.9_{\pm 0.1}$	$93.2_{\pm 0}$	$18.2_{\pm 0.8}$	$\mathbf{8.9}_{\pm \mathbf{0.2}}$
	RLN	94.4	$12.6_{\pm 0.2}$	$93.8_{\pm 0}$	$23.6_{\pm 0.5}$	$\mathbf{10.8}_{\pm \mathbf{0.2}}$
	VIME	92.5	$56.3_{\pm 0.1}$	$92.2_{\pm 0}$	$56.5_{\pm 0.9}$	$\mathbf{49.5}_{\pm \mathbf{0.5}}$
	STG	93.3	$72.6_{\pm 0.0}$	$93.2_{\pm 0}$	$58.2_{\pm 0.9}$	$\mathbf{58.0}_{\pm \mathbf{0.8}}$
	TabNet	93.4	$19.3_{\pm 0.6}$	$90.9_{\pm 0}$	$17.5_{\pm 0.6}$	$\mathbf{11.0}_{\pm \mathbf{0.5}}$
LCLD	TabTr.	69.5	$27.1_{\pm 0.9}$	$61.1_{\pm 0}$	$10.7_{\pm 0.8}$	$\mathbf{7.9}_{\pm \mathbf{0.6}}$
	RLN	68.3	$0.2_{\pm 0.1}$	$38.9_{\pm 0}$	$0.8_{\pm 0.2}$	$\mathbf{0.0}_{\pm \mathbf{0.0}}$
	VIME	67.0	$2.6_{\pm 0.2}$	$52.6_{\pm 0}$	$24.1_{\pm 1.5}$	$\mathbf{2.4}_{\pm \mathbf{0.1}}$
	STG	66.4	$55.5_{\pm 0.2}$	$\mathbf{53.0}_{\pm \mathbf{0}}$	$55.4_{\pm 0.2}$	$53.6_{\pm 0.1}$
	TabNet	67.4	$6.3_{\pm 0.4}$	$49.0_{\pm 0}$	$0.8_{\pm 0.1}$	$\mathbf{0.4}_{\pm \mathbf{0.1}}$
CTU	TabTr.	95.3	$95.3_{\pm \mathbf{0.0}}$	$95.3_{\pm 0}$	$95.3_{\pm 0.0}$	$95.3_{\pm 0.0}$
	RLN	97.8	$97.8_{\pm 0.0}$	$97.5_{\pm 0}$	$\mathbf{94.0}_{\pm \mathbf{0.2}}$	$\mathbf{94.0}_{\pm \mathbf{0.2}}$
	VIME	95.1	$95.1_{\pm 0.0}$	$95.1_{\pm 0}$	$\mathbf{40.8}_{\pm \mathbf{4.7}}$	$\mathbf{40.8}_{\pm \mathbf{4.7}}$
	STG	95.3	$95.3_{\pm 0.0}$	$95.3_{\pm 0}$	$95.3_{\pm 0.0}$	$95.3_{\pm 0.0}$
	TabNet	96.1	$96.1_{\pm 0.0}$	$13.0_{\pm 0}$	$\mathbf{0.0}_{\pm \mathbf{0.0}}$	$\mathbf{0.0}_{\pm \mathbf{0.0}}$
WIDS	TabTr.	75.5	$48.0_{\pm 0.3}$	$67.7_{\pm 0}$	$59.2_{\pm 0.6}$	$\mathbf{45.9}_{\pm \mathbf{0.3}}$
	RLN	77.5	$61.8_{\pm 0.3}$	$77.0_{\pm 0}$	$67.7_{\pm 0.3}$	$\mathbf{60.9}_{\pm \mathbf{0.2}}$
	VIME	72.3	$51.4_{\pm 0.3}$	$71.2_{\pm 0}$	$59.4_{\pm 0.5}$	$\mathbf{50.3}_{\pm \mathbf{0.2}}$
	STG	77.6	$65.1_{\pm 0.4}$	$77.5_{\pm 0}$	$68.8_{\pm 0.3}$	$\mathbf{63.8}_{\pm \mathbf{0.2}}$
	TabNet	79.7	$10.2_{\pm 0.3}$	$73.1_{\pm 0}$	$13.9_{\pm 0.4}$	$\mathbf{5.3}_{\pm \mathbf{0.4}}$

Table 5.4: Attack duration for CAPGD, MOEVA, and CAA. A lower duration is better. The lowest time between MOEVA and CAA is in bold.

		Duration in seconds (↓)			
Dataset	Model	CAPGD	BF*	MOEVA	CAA
URL	TabTr.	$1_{\pm 0}$	$33_{\pm 0}$	$75_{\pm 2}$	$\mathbf{17}_{\pm 1}$
	RLN	$1_{\pm 0}$	$27_{\pm 0}$	$74_{\pm 3}$	$\mathbf{19}_{\pm 1}$
	VIME	$2_{\pm 1}$	$32_{\pm 0}$	$70_{\pm 1}$	$\mathbf{51}_{\pm 2}$
	STG	$2_{\pm 0}$	$58_{\pm 0}$	$90_{\pm 0}$	$\mathbf{73}_{\pm 3}$
	TabNet	$8_{\pm 0}$	$444_{\pm 0}$	$165_{\pm 4}$	$\mathbf{58}_{\pm 1}$
LCLD	TabTr.	$5_{\pm 1}$	$154_{\pm 0}$	$124_{\pm 5}$	$\mathbf{83}_{\pm 2}$
	RLN	$1_{\pm 0}$	$147_{\pm 0}$	$50_{\pm 1}$	$\mathbf{10}_{\pm 3}$
	VIME	$1_{\pm 0}$	$149_{\pm 0}$	$49_{\pm 2}$	$\mathbf{13}_{\pm 1}$
	STG	$3_{\pm 0}$	$191_{\pm 0}$	$60_{\pm 2}$	$\mathbf{57}_{\pm 2}$
	TabNet	$4_{\pm 0}$	$754_{\pm 0}$	$68_{\pm 2}$	$\mathbf{23}_{\pm 0}$
CTU	TabTr.	$4_{\pm 0}$	$371_{\pm 0}$	$\mathbf{98}_{\pm 4}$	$110_{\pm 5}$
	RLN	$9_{\pm 8}$	$12_{\pm 0}$	$\mathbf{98}_{\pm 3}$	$112_{\pm 4}$
	VIME	$4_{\pm 0}$	$924_{\pm 0}$	$\mathbf{107}_{\pm 3}$	$116_{\pm 2}$
	STG	$5_{\pm 0}$	$548_{\pm 0}$	$\mathbf{105}_{\pm 3}$	$119_{\pm 4}$
	TabNet	$7_{\pm 0}$	$816_{\pm 0}$	$\mathbf{157}_{\pm 7}$	$182_{\pm 4}$
WIDS	TabTr.	$3_{\pm 0}$	$440_{\pm 0}$	$65_{\pm 3}$	$\mathbf{49}_{\pm 1}$
	RLN	$3_{\pm 0}$	$2520_{\pm 0}$	$52_{\pm 1}$	$\mathbf{49}_{\pm 2}$
	VIME	$2_{\pm 0}$	$1406_{\pm 0}$	$48_{\pm 2}$	$\mathbf{41}_{\pm 1}$
	STG	$3_{\pm 0}$	$1888_{\pm 0}$	$64_{\pm 1}$	$\mathbf{59}_{\pm 1}$
	TabNet	$5_{\pm 0}$	$10472_{\pm 0}$	$77_{\pm 4}$	$\mathbf{25}_{\pm 1}$

attacks in terms of the original examples for which they could generate successful adversarial examples. We also include CAPGD in this comparison, since we have shown that this attack subsumes the other gradient-based attacks. Our results reveal that CAPGD and MOEVA together subsume BF* except for 5 examples. Additionally, CAPGD and MOEVA are complementary, with CAPGD generating 477 unique examples and MOEVA 953. Overall, the combination of CAPGD with MOEVA yields the strongest method including only one gradient-based attack and one search-based attack. One could also include BF* for a slight increase in effectiveness, but this would come at the computational cost of running this attack in addition to the other two; our experimental results (presented below) actually reveal that BF* brings a substantial computational cost compared to CAPGD and MOEVA. Hence, we stick to CAPGD and MOEVA only.

The principle of CAA is thus to successively apply CAPGD and MOEVA, in that order. By applying CAPGD first, CAA has the opportunity to generate valid adversarial examples at low computational cost (benefiting from the performance of gradient attacks compared to search attacks). If CAPGD fails on an original example, CAA executes the slower but more effective MOEVA method.

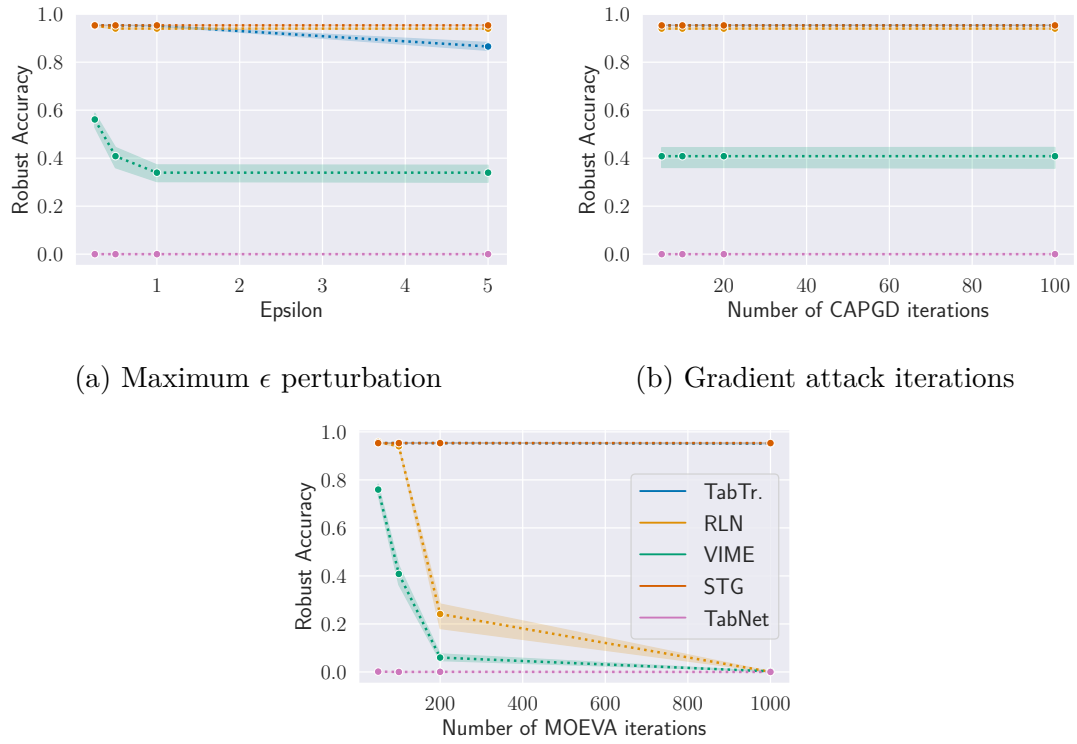
5.4.2 Effectiveness and efficiency of CAA

We evaluated the effectiveness (robust accuracy) and efficiency (computation time) of CAA compared to the other methods. The hyperparameters of all attacks are fixed for all experiments (see Section 10.2.1 of the appendix) and follow the recommendation given in their original paper.

Table 5.3 shows that CAA achieves the best performance in all cases but one: for STG model and LCLD dataset, BF* achieves a robust accuracy 0.6 percentage points lower than CAA – these are the unique original examples from which BF* could generate successful adversarial examples. Overall, CAA leads to a decrease in accuracy of up to 96.1%, 84.3%, and 21.7% compared to CAPGD, BF*, and MOEVA respectively.

The main advantage of CAA is its ability to find "easy" constrained adversarial using the cheaper gradient attack CAPGD, before processing harder examples with expensive search. We compare in Table 5.4 the cost of running each attack, i.e., its execution time. CAA shines particularly in terms of efficiency. CAA reduces execution costs by up to 5 times compared to MOEVA. It is significantly faster than MOEVA for LCLD, URL, and WIDS datasets (except STG), and marginally more costly for CTU.

CAA is up to 418 times faster than BF*. In particular, in the only case where BF* marginally outperforms CAA, BF* requires 3.4 times more computation to generate the adversarial examples.



(a) Maximum ϵ perturbation

(b) Gradient attack iterations

(c) Search attack iterations

Figure 5.4: Impact of CAA budget on the robust accuracy for CTU dataset.

5.4.3 Impact of attack budget

We study the impact of the attacker's budget on the effectiveness of CAA, in terms of (i) maximum perturbation ϵ and (ii) the number of iterations of its components. We focus on the CTU dataset, which models are the only ones to remain robust through our previous experiments. All the datasets are evaluated in Appendix 10.2.2. Figure 5.4 reveals in (a) that the maximum perturbation distance ϵ has little impact on the effectiveness of the attack. Increasing the number of iterations for the gradient attack component (b) does not have an impact on the success rate of CAA. Increasing the budget of the search attack component (c) significantly impacts the robustness of some models. While TabTransformer and STG remain robust, the robust accuracy of RLM and VIME drops below 0.4 when doubling the number of search iterations to 200, and to zero with 1000 search iterations.

5.4.4 Impact of Adversarial training

We evaluate the effectiveness of our attack against models made robust with Madry's adversarial training (AT) [MMS+17], using examples generated by the

Table 5.5: CAA performances (XX+/-YY) against Madry adversarially trained model. XX refers to accuracy. YY is the difference between the accuracy of the adversarially trained model and standard training (cf. Table 5.3), such that '+' means a higher accuracy for the adversarially trained model.

Dataset	Accuracy	TabTr.	RLN	VIME	STG	TabNet
URL	Clean	$93.9_{+0.3}$	$95.2_{+0.8}$	$93.4_{+0.9}$	$94.3_{+1.0}$	$99.5_{+6.1}$
	CAA	$56.7_{+47.8}$	$56.2_{+45.4}$	$69.8_{+20.3}$	$90.0_{+32.0}$	$91.8_{+80.8}$
LCLD	Clean	$73.9_{+4.4}$	$69.5_{+1.2}$	$65.5_{-1.5}$	$15.6_{-50.8}$	$0.0_{-67.4}$
	CAA	$70.3_{+62.4}$	$63.0_{+63.0}$	$10.4_{+8.0}$	$12.1_{-41.5}$	$0.0_{-0.4}$
CTU	Clean	$95.3_{+0.0}$	$97.3_{-0.5}$	$95.1_{+0.0}$	$95.1_{-0.2}$	$0.2_{-95.8}$
	CAA	$95.3_{+0.0}$	$97.1_{+3.0}$	$94.0_{+53.2}$	$95.1_{-0.2}$	$0.2_{+0.2}$
WIDS	Clean	$77.3_{+1.8}$	$78.0_{+0.5}$	$72.1_{-0.2}$	$62.6_{-15.1}$	$98.4_{+18.6}$
	CAA	$65.1_{+19.2}$	$66.6_{+5.7}$	$52.1_{+1.8}$	$45.2_{-18.6}$	$58.4_{+53.1}$

PGD attack. We consider this defense because it was shown to be the only reliable defense against evasion attacks [TCB+20; Car23]. In Table 5.5, we show the clean accuracy and the robust accuracy (against CAA) of the adversarially trained models (big numbers in Table 5.5). We also show the accuracy difference with the models trained with standard training (small numbers).

Adversarial training can degrade clean and robust performance. As a preliminary check, we investigate whether adversarial training degrades the clean performance of the models. This is important to ensure that a non-increase of robust accuracy does not originate from clean performance degradation (instead of being due to CAA's strength). Our evaluation shows that adversarial training significantly degrades the clean performance of the STG and Tabnet architectures. The accuracy of STG models drops to 15.6% and 62.6% for LCLD and WIDS respectively. As for Tabnet models, the clean accuracy drops to 0.0% (LCLD) and 0.2% (CTU). In all other cases, clean accuracy remains stable.

CAA remains effective against adversarial training for some architectures. The effectiveness of CAA against robust models is architecture- and dataset-dependent. The attack remains effective on VIME architecture applied to LCLD and WIDS, with robust accuracy as low as 10.1% and 52.2% respectively, as well as on RLN architecture on the WIDS dataset (66.6% robust accuracy and only +5.7% improvement compared to standard training). However, the robustness against CAA of Tabtransformer architecture is significantly improved on URL and LCLD datasets by respectively 47.8% and 62.4%, and marginally improved on WIDS dataset by 19.2%. Similarly, RLN robustness to CAA improves on URL (+45.4%) and LCLD (+63%).

5.4.5 Impact of constraints complexity

CAPGD and CAA fail to generate adversarial examples on CTU for 2 out of 5 models. CTU dataset has a large number of constraints (360) compared to the other datasets, and some are particularly challenging (involving 90 features). We argue that some of these constraints hinder gradient attacks and are harder to optimize. To confirm our hypothesis and provide additional insights, we split the constraints of CTU based on their type into 4 buckets:

- CG0: 1 constraint involving 90 features in the form of $\sum F_i = \sum F_j$ where both sums represent the total number of sent packets.
- CG1: 1 constraint involving 90 features in the form of $\sum F_i = \sum F_j$ where both sums represent the total number of received packets.
- CG2: 34 constraints in the form of $BYTE/PACKETS \leq 1500$, to model the fact that each packet contains at most 1500 bytes.
- CG3: 324 constraints in the form of $A \leq B$ where A and B are statistical properties (min, max, sum) for each port, and direction.

First, we run an ablation study, where we ignore one bucket of constraint at a time. Next, we study the success rate when we considered each bucket separately. Finally, we reported the impact of the number of constraints to optimize from CG3, the largest bucket.

The number of constraints and their complexity impact the success rate of attack. The results in Table 5.6 and 5.7 show that for gradient attacks, removing one type of constraint is not enough to improve the success rate. Constraints across multiple remaining categories are not satisfied. The individual bucket study confirms that only when considering constraints of type CG2 alone, CAPGD improves its success rate (in VIME and TabNet). When only considering CG3 constraints, reducing the number of constraints improves the success rate (by reducing robust accuracy from 95.3% when considering 100% of CG3 constraints to 84.5% and 43.0% respectively when considering 50% and 10% of the constraints).

5.4.6 Generalization to non-differentiable models

Tree-based models, and in particular random forest and XGBoost models remain among the best architectures for tabular data for some datasets [BLS+21]. We hypothesize that CAA is relevant to any models, including tree-based models with two settings:

1. in transferability, by generating adversarial examples on a surrogate model and evaluating their success rate on the target (tree-based) model,

2. by applying CAA (through the search-based component MOEVA that is model agnostic).

Table 5.6: Robust accuracy with subset of constraints and CAPGD attack. Ω is the complete set of constraints. CGX denotes the constraint group X. For CG2 and CG3, we evaluate with the entire group and on 10%, 25%, 50% selected randomly and averaged on 5 seeds.

		CAPGD				
	Group	RLN	STG	TabNet	TabTr.	VIME
Ablation	Ω	97.8	95.3	96.1	95.3	95.1
	$\Omega \setminus CG0$	97.8	95.3	96.1	95.3	95.1
	$\Omega \setminus CG1$	97.8	95.3	96.1	95.3	95.1
	$\Omega \setminus CG2$	97.8	95.3	96.1	95.3	95.1
	$\Omega \setminus CG3$	97.8	95.3	96.1	95.3	95.1
Components	CG0	97.8	95.3	96.1	95.3	95.1
	CG1	97.8	95.3	96.1	95.3	95.1
	CG2	75.3	95.3	31.4	94.3	0.0
	CG3	97.2	95.3	95.3	95.3	95.1
Percentage CG3	10%	85.2	95.3	43.0	95.1	11.3
	25%	93.4	95.3	59.9	95.3	37.8
	50%	94.9	95.3	84.5	95.3	93.2

Table 5.7: Robust accuracy with subset of constraints and CAA attack. Ω is the complete set of constraints. CGX denotes the constraint group X. For CG2 and CG3, we evaluate with the entire group and on 10%, 25%, 50% selected randomly and averaged on 5 seeds.

		CAA				
	Group	RLN	STG	TabNet	TabTr.	VIME
Ablation	Ω	94.0	95.3	0.0	95.3	40.8
	$\Omega \setminus CG0$	93.9	95.3	0.0	95.3	21.0
	$\Omega \setminus CG1$	94.1	95.3	0.0	95.3	37.1
	$\Omega \setminus CG2$	93.9	95.3	0.1	95.3	40.8
	$\Omega \setminus CG3$	89.3	95.3	0.0	95.3	2.4
Components	CG0	91.6	95.3	0.0	95.2	2.8
	CG1	91.1	95.3	0.0	95.2	1.9
	CG2	72.0	95.3	0.0	94.3	0.0
	CG3	92.7	95.3	0.0	95.3	19.0
Percentage CG3	10%	80.8	95.3	0.0	94.9	0.6
	25%	87.1	95.3	0.0	95.3	2.5
	50%	88.6	95.3	0.0	95.3	8.3

Table 5.8: Tree-based model robust accuracy in direct and transferability scenario (minimum robust accuracy over 5 neural networks).

Dataset	Model	Clean	Direct	Transferability
URL	Random Forest	96.2	52.7	72.4
	XGBoost	97.4	27.3	46.7
LCLD	Random Forest	64.3	22.3	5.3
	XGBoost	68.3	9.1	9.4
CTU	Random Forest	95.6	92.2	95.0
	XGBoost	97.3	76.3	97.1
WIDS	Random Forest	52.2	5.2	14.1
	XGBoost	80.4	38.0	60.7

We train Random Forests (RF) and XGBOOST models to achieve the best performance on our datasets.

CAA is effective against non-differentiable models Table 5.8 shows the robust accuracy of both models against CAA for the four datasets in two settings: (1) Direct attack where CAA (using its search component MOEVA) attacks directly the RF and XGBOOST models, and (2) Transfer attacks, where we craft the examples on our deep learning (DL) models and evaluate them on the RF and XGBOOST models.

Our evaluation shows that (1) DL models of our study achieve comparable clean performance to the shallow models, (2) both RF and XGBOOST models are vulnerable to direct CAA attacks (down to 9.1% of robust accuracy on LCLD XGBoost), and (3) CAA attacks on DNN transfer to RF (down to 5.3% robust accuracy) and XGBoost (down to 9.4% robust accuracy) models.

The results confirm the relevance and significance of our attacks on tabular models, including undifferentiable models.

5.5 Conclusion

In this chapter, we first propose CAPGD, a new parameter-free gradient attack for constrained tabular machine learning. We also design CAA, a new Constrained Adaptive Attack that combines the best gradient-based attack (CAPGD) and the best search-based attack (MOEVA). We evaluate our attacks over four datasets and five architectures and demonstrated that our new attacks outperform all previous attacks in terms of effectiveness and efficiency. We believe that our work is a springboard for further research on the robustness of tabular machine learning and to open multiple research perspectives on constrained tabular ML. We hope that CAA will contribute to a faster development of adversarial defenses and recommend it as part of a standard evaluation pipeline of new tabular machine models. One promising venue is the field of cost-effective query-based attacks, that remained unexplored on constrained tabular data. Combining our domain-constrained protocols with concepts of cost and benefits will be crucial in assessing and bolstering the resilience of machine learning systems and ensuring their reliability in real-world applications. Another promising direction is in the evaluation of robustification methods such as adversarial training in combination with synthetic data. We explore this research direction in the next chapter where we propose a benchmark, evaluating 14 training methods across 5 datasets using CAA as a strong and effective attack.

6

Defending Against Adversarial Attacks in Tabular Deep Learning

After proposing Constrained Adaptive Attack (CAA), a strong constrained adversarial attack, we study robustification mechanisms for deep neural network architectures. We propose training methods based on adversarial training and synthetic data generation.

Contents

6.1 Introduction

This chapter addresses the second challenge of this thesis which is the robustification of machine learning models against adversarial examples. We leverage the strong and effective attack CAA proposed in Chapter 5 to evaluate the clean and robust accuracy of 14 training methods on 5 architectures from different mechanism families and across 5 datasets.

Modern machine learning (ML) models have reached or surpassed human-level performance in numerous tasks, leading to their adoption in critical settings such as finance, security, and healthcare. However, concomitantly to their increasing deployment, researchers have uncovered significant vulnerabilities in generating valid adversarial examples (i.e., constraint-satisfying) where test or deployment data are manipulated to deceive the model. Most analyses of these performance drops have focused on the fields of Computer Vision and Large Language Models where extensive benchmarks for adversarial robustness are available (e.g., [CAS+20] and [WCP+23]).

The need for dedicated defenses for tabular model robustness is enhanced by the unique challenges that tabular machine learning raises compared to computer vision and NLP tasks. One significant challenge is that tabular data exhibit *feature constraints*, which are complex relationships and interactions between features. Satisfying these feature constraints can be a non-convex or even non-differentiable problem, making established evasion attack algorithms relying on gradient descent ineffective in generating valid adversarial examples (i.e., constraint-satisfying) [GCG+20]. Furthermore, attacks designed specifically for tabular data often disregard feature-type constraints [BAL+19] or, at best, consider categorical features without accounting for feature relationships [WHB+20; XHR+23; BHZ+23], and are evaluated on datasets that contain only such features. This limitation restricts their applicability to domains with heterogeneous feature types.

Moreover, tabular ML models often involve specific feature engineering, that is, "secret" and inaccessible to an attacker. For example, in credit scoring applications, the end user can alter a subset of model features, but the other features result from internal processing that adds domain knowledge before reaching the model [GCG+20]. This raises the need for new threat models that take into account these specificities.

Thus, the machine learning research community currently lacks an empirical understanding of the impact of architecture in combination with robustification mechanisms on tabular data model architectures.

To address this gap, we propose 6 new defense mechanisms based on synthetic data generation. We evaluated adversarial robustness using *Constrained Adaptive Attack (CAA)* introduced in Chapter 5, a combination of gradient-based and search-based attacks that have been shown to be the most effective against tabular models.

Our empirical study on defenses based on adversarial training (AT) reveals the following insights:

Test performance is misleading: Given the same tasks, different architectures have similar ID performance but lead to very disparate robust performances. Even more, data augmentations that improve ID performance can hurt robust performance.

Importance of domain constraints: Disregarding domain constraints overestimates robustness and leads to the selection of sub-optimal architectures and defenses when considering the domain constraints.

Data augmentation effectiveness is task-specific. There is no data augmentation that is optimal for both ID and robust performance across all tasks. Some simpler augmentations (like Cutmix) can outperform complex generative approaches.

Contributions. Our contributions can be summarized as follows:

- Six defense mechanisms that combine synthetic data generation with adversarial training to train robust models.
- A large-scale empirical study that reveals that test performance is misleading, the importance of domain constraints, and that augmentation effectiveness is task-specific.
- An industrial application of our defense mechanisms to the use-case of BGL BNP Paribas, a major financial actor in Luxembourg.

6.2 Preliminaries

Adversarial robustness refers to the ability of a model to maintain its accuracy when exposed to adversarial examples, which are inputs that have been intentionally perturbed to cause the model to make a classification mistake.

6.2.1 Problem formulation

Given a machine learning model $H(.)$, and a pair of input $x \in \mathcal{X}$ and label $y \in \mathcal{Y}$ the objective is to ensure that the model's prediction $H(x)$ remains correct even if x is perturbed as an adversarial example $\hat{x} = x + \delta$. The adversarial example can be unconstrained, that is only limited by a distance D threshold ϵ like in computer vision: $D(x, \hat{x}) \leq \epsilon$, or constrained, adding the constraints satisfaction requirements as in Chapter 4 and 5: $x \in \mathcal{X}$.

The goal for the defender is to minimize the empirical risk of adversarial examples.

Tramèr et al. [TCB+20] demonstrate that several recently published defenses, although evaluated on existing adaptive attacks, could still be broken by new adaptive attacks. More recently Carlini [Car23] showed the simplicity of creating such

attacks by letting a large language model propose the attack. Hence, adversarial training-based methods are so far the only reliable defense against evasion attacks.

6.2.2 Adversarial training

We consider a classifier H with parameters θ trained with a loss function l. Madry et al. [MMS+17] formulate a saddle point problem to find model parameters that minimize the adversarial risk:

$$\arg\min_{\theta} \mathbb{E}_{(x,y)\sim\mathcal{D}}\Big[\max_{\delta\in\mathcal{S}} l(\theta, x+\delta, y)\Big] \tag{6.1}$$

where (x, y) are training data samples from $\mathcal{D}$ distribution, $\delta \in \mathcal{S}$ is the perturbation in the set of allowed maximum perturbations, l is the loss of the model, and θ the model's learnable parameters.

To solve the inner optimization problem, Madry et al. [MMS+17] use Projected Gradient Descent to generate adversarial examples.

PGD adds iteratively a perturbation δ that follows the sign of the gradient ∇ with respect to the current adversary x_t of the input x. That is, at iteration $t+1$ it produces the input

$$x^{t+1} = \Pi_{x+\delta}(x^t + \alpha sgn(\nabla_{x^t} l(\theta_h, x^t, y))) \tag{6.2}$$

where θ_h the parameters of our predictor h, Π is a clip function ensuring that $x+\delta$ remains bounded in a sphere around x of a size ϵ using a norm p, and $\nabla_x l$ is the gradient of the loss function tailored to our task. For instance, we can use cross-entropy loss for classification tasks.

6.2.3 Robust overfitting

Rice et al. [RWK20] demonstrated that adversarial training suffers from *robust overfitting*. Robust overfitting is a phenomenon that causes the accuracy on the train set to quickly deteriorate while it continues to increase on the train set. Rice et al. [RWK20] propose to use early stopping as the main method against robust overfitting and show that models trained with early stopping are more robust than those trained with other regularization techniques such as data augmentation. On the contrary Rebuffi et al. [RGC+21] show that combining data augmentation with regularization methods can significantly improve robustness.

6.3 Defense against adversarials in tabular data

We present our defense mechanisms against adversarial examples in computer vision.

6.3.1 Adversarial training

We start with adversarial training. Madry et al. [MMS+17] showed the importance of using a strong attack during adversarial training.

We use adversarial training with PGD as in [MMS+17]. We compute the adversarial example for K gradient steps of size α

$$x^{(k+1)} = P_{\mathcal{S}}(x^{(k)} + \alpha \nabla \mathcal{L}(x^{(k)}, y, h))) \tag{6.3}$$

where $x^{(0)}$ is chosen at random within $\mathcal{S}$ distance of the clean example x_0, $P_{\mathcal{A}}(a)$ projects a point a back onto a set $\mathcal{A}$ and ∇ is the gradient projected on the L_2 unit sphere.

We use PGD with the adaptation of Kurakin et al. [KGB16] that trains with both clean and adversarial examples at each batch.

The loss function at each batch is therefore:

$$\hat{\mathcal{L}_B} = \frac{1}{(m-k)+\lambda k}\left(\lambda \sum_{i=1}^{k} \mathcal{L}(h(PGD(x_i, y_i)), y_i) + \sum_{i=k+1}^{m} \mathcal{L}(h(x_i), y_i)\right) \tag{6.4}$$

where m is the batch size, k is the number of adversarial examples and λ the weights of adversarial examples. We used $k = m/2$ and $\lambda = 1$. As depicted in the loss function, the synthetic data generator produces new examples at each batch. Hence, for a model trained for e epochs over a training set of size $|X_{train}|$ we generate $\frac{e \times |X_{train}|}{2}$ synthetic examples.

6.3.2 Data augmentation and adversarial training

Inspired by a recent study in computer vision, that demonstrates the effectiveness of data augmentation methods in combination with adversarial training, we propose to use data augmentation for tabular data to robustify models.

Data augmentation methods

We use data augmentation to artificially expand the size and diversity of a training dataset.

We distinguish between two types of data augmentation methods, heuristic-based and data-driven augmentations.

Heuristic-based data generation methods modify existing examples based on one or more existing samples following simple rules. These methods include, CutMix[YHO+19] which combines two images by cutting a rectangular region from one image and pasting it onto another, Cutout [DT17] replaces a part of the image with an empty patch, and MixUp [ZCD+17] that creates convex combinations of pairs of images. We propose a simple adaptation of CutMix for tabular data where

we randomly combine two clean examples to create a new pair of examples. By definition, Cutout and MixUp are not directly adaptable to tabular data. For Cutout, that would mean resetting part of the feature values to zero. For MixUp, convex combinations of different feature types in a single example are not as straightforward as for image. Hence, these synthetic generation methods are not directly adaptable to tabular data.

Data-driven generation methods train a generative model by learning the distribution of the training data. Based on the learned distribution, these techniques can produce synthetic data samples. We reuse recent advances in the field of Deep Generative Models (DGM) to build our defense.

We use the same five generative models as [SDC+24]:

- **WGAN** [ACB17] is a GAN model trained with Wasserstein loss within a standard generator-discriminator GAN framework. In our implementation, WGAN utilizes a MinMax transformer for continuous features and one-hot encoding for categorical features. It is not specifically designed for tabular data.
- **TableGAN** [PMG+18] is one of the pioneering GAN-based methods for generating tabular data. Besides the conventional generator and discriminator setup in GANs, the authors introduced a classifier trained to understand the relationship between labels and other features. This classifier ensures a higher number of semantically correct generated records. TableGAN applies a MinMax transformer to the features.
- **CTGAN** [XSC+19b] employs a conditional generator and a training-by-sampling strategy within a generator-discriminator GAN framework to model tabular data. The conditional generator produces synthetic rows conditioned on one of the discrete columns. The training-by-sampling method ensures that data are sampled according to the log frequency of each category, aiding in better modeling of imbalanced categorical columns. CTGAN uses one-hot encoding for discrete features and a mode-based normalization for continuous features. A variational Gaussian mixture model [CHS18] is used to estimate the number of modes and fit a Gaussian mixture. For each continuous value, a mode is sampled based on probability densities, and its mean and standard deviation are used for normalization.
- **TVAE** [XSC+19b] was introduced as a variant of the standard Variational AutoEncoder to handle tabular data. It employs the same data transformations as CTGAN and trains the encoder-decoder architecture using evidence lower-bound (ELBO) loss.
- **GOGGLE** [LQB+23] is a graph-based method for learning the relational structure of data as well as functional relationships (dependencies between features). The relational structure is learned by constructing a graph where

nodes represent variables and edges indicate dependencies between them. Functional dependencies are learned through a message-passing neural network (MPNN). The generative model generates each variable considering its surrounding neighborhood.

Adversarial training with data augmentation

Let GEN be a generic synthetic data generation function that takes as input a batch i of inputs (X_i, Y_i).

Following the same principle as with adversarial training, we train with a combination of k_1 clean examples, k_2 adversarial examples, k_3 synthetic examples, and k_4 adversarial synthetic data with $\lambda_1, \lambda_2, \lambda_3, \lambda_4$ weights respectively

The loss function at each batch is

$$\begin{aligned}\mathcal{L}_B = &\frac{1}{\sum_{j=1}^{4} \lambda_j k_j}\Big(\\ &\lambda_1 \sum_{i=1}^{k_1} \mathcal{L}(h(x_i, y_i), y_i) + \\ &\lambda_2 \sum_{i=k_1+1}^{k_1+k_2} \mathcal{L}(h(PGD(x_i, y_i)), y_i) + \\ &\lambda_3 \sum_{i=k_1+k_2+1}^{k_1+k_2+k_3} \mathcal{L}(h(GEN(x_i, y_i)), y_i) + \\ &\lambda_4 \sum_{i=k_1+k_2+k_3+1}^{m} \mathcal{L}(h(PGD(GEN(x_i, y_i), y_i)), y_i)\Big)\end{aligned} \tag{6.5}$$

We use $\lambda_i = 1$ and $k_i = m/4$ for all $i \in [1, ..., 4]$.

6.4 Empirical settings

We detail in this section the evaluation settings of our empirical study.

6.4.1 Tasks

We curated datasets meeting the following criteria: (1) **open source:** the datasets must be publicly available with a clear definition of the features and preprocessing, (2) **from real-world applications:** datasets that do not contain simulated data, (3) **binary classification:** datasets that support a meaningful binary classification task, and (4) **with feature relationships**: datasets that contain feature relationships and constraints, or they can be inferred directly from the definitions of features.

After an extensive review of tabular datasets, only the following five datasets match our requirements.

Table 6.1: Properties of the use cases of our benchmark.

Dataset	Domain	Output to flip	Total size	# Features	# Ctrs	Inbalance
CTU	Botnet detection	Malicious connections	198 128	756	360	99.3/0.7
LCLD	Credit scoring	Reject loan request	1 220 092	28	9	80/20
Malware	Malware detection	Malicious software	17 584	24 222	7	45.5/54.5
URL	Phishing	Malicious URL	11 430	63	14	50/50
WIDS	ICU survival	Expected survival	91 713	186	31	91.4/8.6

The **CTU** [CO22] includes legitimate and botnet traffic from CTU University. Its challenge lies in the extensive number of linear domain constraints, totaling 360. **LCLD** [Geo18] is a credit-scoring containing accepted and rejected credit requests. It has 28 features and 9 *non-linear* constraints. The most challenging dataset of our study is the **Malware** dataset prepared by [DGS+22]. The very large number of features (24222), most of which are involved in each constraint, make this dataset challenging to attack. **URL** [HY21] is a dataset comprising both legitimate and phishing URLs. Featuring only 14 linear domain constraints and 63 features, it represents the simplest case in our study. The **WiDS** [LRG+20] includes medical data on the survival of patients admitted to the ICU, with only 31 linear domain constraints.

Our datasets include varying complexity in terms of number of features and constraints and diverse class imbalance intensity. We summarize the datasets and their relevant properties in Table 6.1 and provide more details in Appendix 10.3.1.

6.4.2 Architectures

We consider five state-of-the-art deep tabular architectures from the survey by [BLS+21]: **TabTransformer** [HKC+20] and **TabNet** [AP21], are based on transformer architectures. **RLN** [SS18] uses a regularization coefficient to minimize a counterfactual loss, **STG** [YLN+20] improves feature selection using stochastic gates, while **VIME** [YZJ+20] depends on self-supervised learning. We provide in Appendix 10.3.1 the details of the architectures and the training hyperparameters. These architectures are on par with XGBoost, the top shallow machine-learning model for our applications.

6.4.3 Metrics

The models are fine-tuned to maximize cross-validation AUC. This metric is threshold-independent and is not affected by the class unbalance of our dataset.

We only attack clean examples that are not already misclassified by the model and from the critical class, that is respectively for each aforementioned dataset the class of phishing URLs, rejected loans, malwares, botnets, and not surviving patients. Because we consider a single class, the only relevant metric is robust accuracy on constrained examples, which corresponds to the recall. Unsuccessful

Table 6.2: Clean and robust performances across all architectures in the form XX/YY. XX is the accuracy with standard training, and YY is the accuracy with adversarial training.

Dataset	Accuracy	TabTr.	RLN	VIME	STG	TabNet
CTU	ID	95.3/95.3	97.8/97.3	95.1/95.1	95.3/95.1	96.0/0.2
	Robust	95.3/95.3	94.1/97.1	40.8/94.0	95.3/95.1	0.0/0.2
LCLD	ID	69.5/73.9	68.3/69.5	67.0/65.5	66.4/15.6	67.4/0.0
	Robust	7.9/70.3	0.0/63.0	2.4/10.4	53.6/12.1	0.4/0.0
MALWARE	ID	95.0/95.0	95.0/96.0	95.0/92.0	93.0/93.0	99.0/99.0
	Robust	94.0/95.0	94.0/96.0	95.0/92.0	93.0/93.0	97.0/99.0
URL	ID	93.6/93.9	94.4/95.2	92.5/93.4	93.3/94.3	93.4/99.5
	Robust	8.9/56.7	10.8/56.2	49.5/69.8	58.0/90.0	11.0/91.8
WIDS	ID	75.5/77.3	77.5/78.0	72.3/72.1	77.7/62.6	79.8/98.4
	Robust	45.9/65.1	60.9/66.6	50.3/52.1	50.3/45.2	5.3/58.4

adversarial examples count as correctly classified when measuring robust accuracy.

We only consider examples that respect domain constraints to compute robust accuracy. If an attack generates invalid examples, they are defacto considered unsuccessful and are reverted to their original example (correctly classified).

We report in the Appendix 10.13 all the remaining performance metrics, including the recall, the precision, and the Matthew Correlation Coefficient (MCC).

6.5 Empirical evaluation

We provide multiple figures to visualize the main insights. We only report scenarios where data augmentation and adversarial training do not lead to performance collapse. We report in Appendix 10.3.2 all the results and investigate the collapsed scenarios.

6.5.1 Adversarial Training

We report the ID and robust accuracies of our architectures prior to data increase in Table 6.2.

Adversarial training alone is not enough to robustify models. AT improves adversarial accuracy for all the cases, but AT alone is not sufficient to completely robustify the models on LCLD, URL, and WIDS datasets. For CTU, adversarial training the maximum drop of accuracy with adversarial training is 1.1% for VIME, when adversarial training does not break clean performance (clean accuracy of 0% for TabNet). All malware classification models are completely robust with and without adversarial training; hence, we will restrict the study of

Table 6.3: CAA performances against Madry Adversarially Trained (AT) model. Adv. Tr. + Augmentation correspond to the best robust accuracy of AT in combination with a Data augmentation among Cutmix, TVAE, WGAN, TableGAN, CT-GAN and GOGGLE.

Dataset	Training	TabTr.	RLN	VIME	STG	TabNet
URL	Adversarial Training	56.7	56.2	69.8	90.0	91.8
	Adv. Tr. + Augmentation	66.0	66.1	73.7	85.6	89.9
LCLD	Adversarial Training	70.3	63.0	10.4	12.1	0.0
	Adv. Tr. + Augmentation	78.5	64.3	76.8	81.2	100.0
CTU	Adversarial Training	95.3	97.1	94.0	95.1	0.2
	Adv. Tr. + Augmentation	98.3	97.5	100.0	98.3	100.0
WIDS	Adversarial Training	65.1	66.6	52.1	45.2	58.4
	Adv. Tr. + Augmentation	68.1	100.0	100.0	73.8	100.0

improved defenses with augmentation in the following sections to the remaining datasets.

6.5.2 Evaluation of adversarial training in combination with data augmentation

Adversarial training with data augmentation outperforms adversarial training alone.

Table 6.3 reports the robust accuracy of adversarial training alone and the best robust accuracy of adversarial training in combination with data augmentation. We observe that combining adversarial training with data augmentation systematically outperforms adversarial training alone except for URL with STG and TabNet. Note that in these two cases, the robust accuracy is greater than 85.6%. For instance, for LCLD, the VIME architecture's robust accuracy augments from 10.4% to 76.8%. Overall, all architectures can benefit from at least one data augmentation technique with adversarial training; however, standard training with data augmentation can outperform adversarial training without data augmentation (for e.g., on URL dataset using GOGGLE or CTGAN augmentations).

No single data augmentation method consistently outperforms the others.

In Figure 6.1, we show on the Y axis, the robustness against constrained adversarial examples. In blue, we report adversarial training methods. We observe that the most effective augmentation method (represented with symbols) varies depending on the architecture and models. For instance, in LCLD case, TVAE is most effective for RLN architecture but CTGAN is better for STG TabTransformer

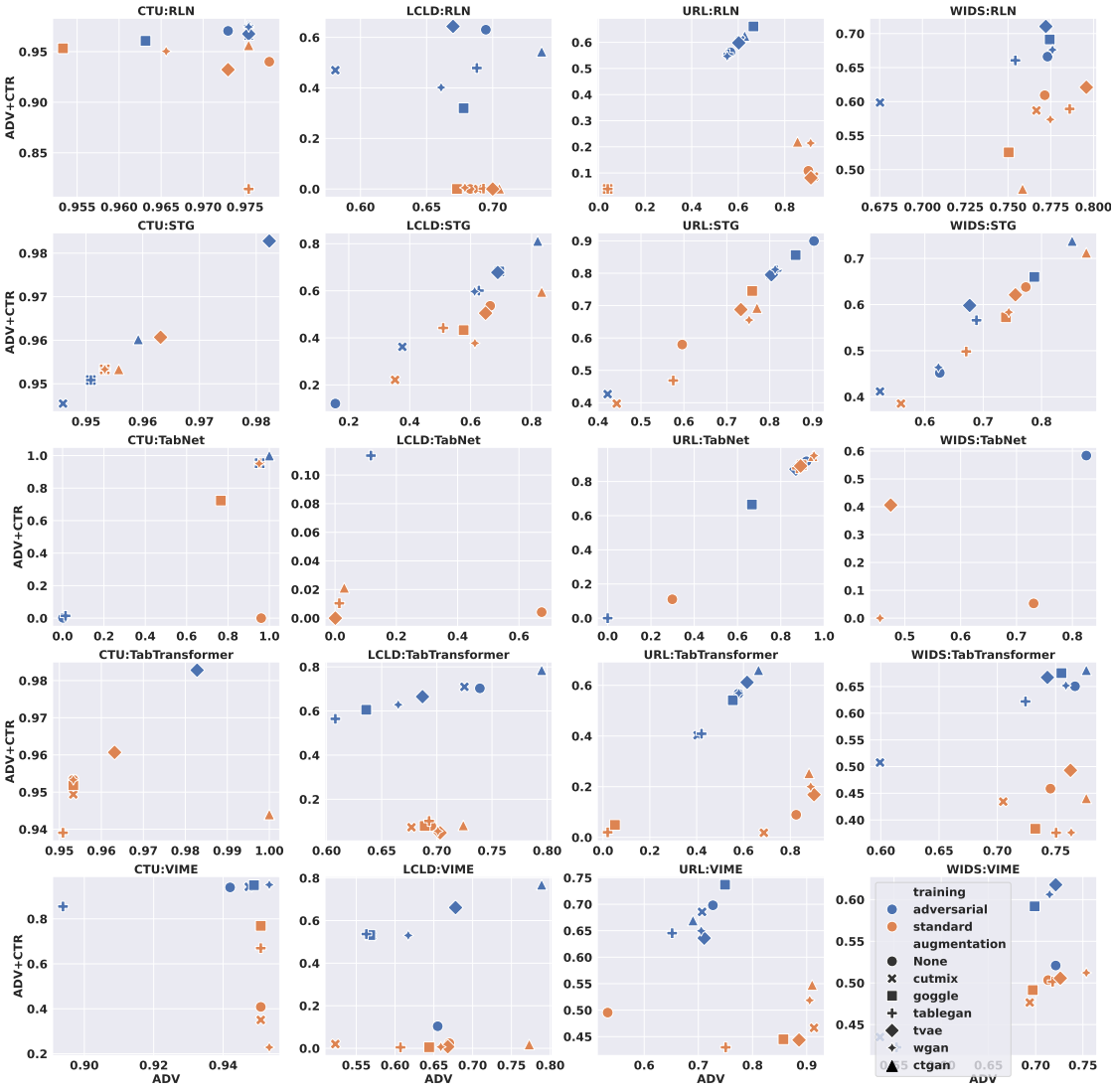
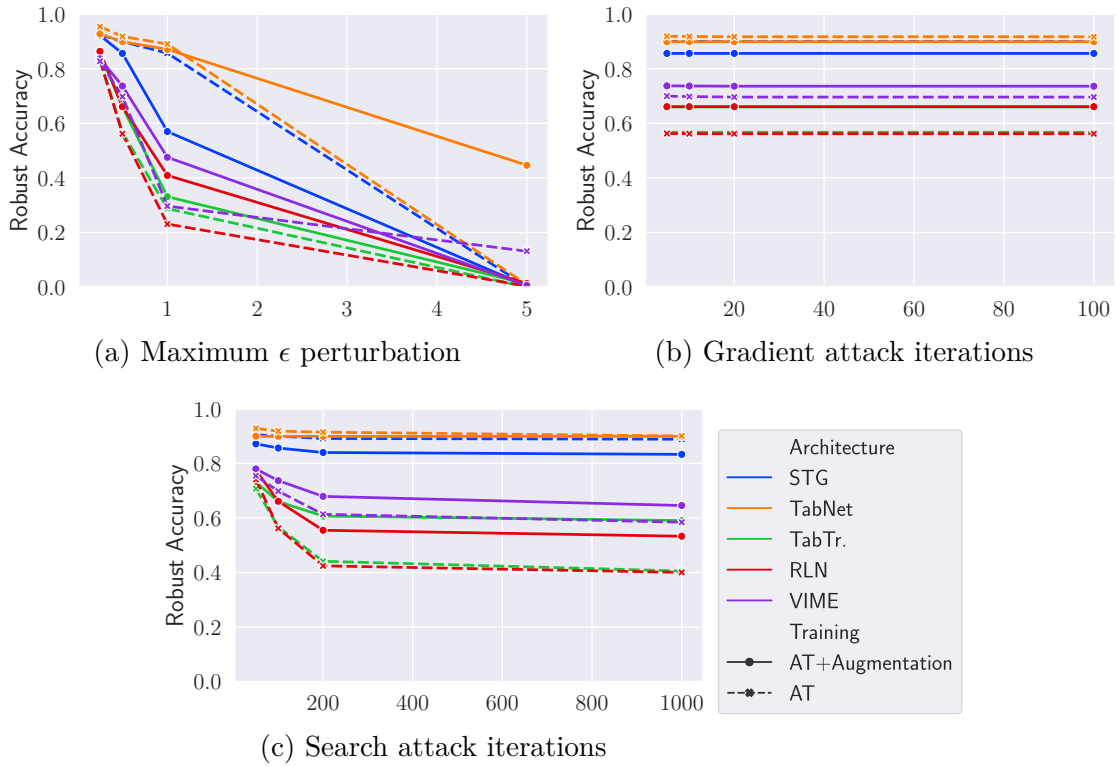


Figure 6.1: Robust performance while considering domain constraints (ADV+CTR: Y-axis) and without (ADV: X-axis) on all our use cases confirms the relevance of studying constrained-aware attacks.



(a) Maximum ϵ perturbation

(b) Gradient attack iterations

(c) Search attack iterations

Figure 6.2: Impact of attack budget on the robust accuracy for URL dataset.

and VIME. Similarly, For URL-RLN, the most effective augmentation method is GOGGLE while it is TVAE for LCLD.

AT+Augmentations models remain robust even under stronger attacks.

We evaluated each robustified model against variants of the CAA attack, varying the L_2 distance of the perturbation ϵ from 0.5 to $\{0.25, 1, 5\}$, the gradient iterations from 10 to $\{5, 20, 100\}$, and the search iterations from 100 to $\{50, 200, 1000\}$. We report per architecture for each dataset the most robust model with AT and augmentation, and the robust model with AT only. We present in Fig. 6.2 the results for the URL dataset and refer to Appendix 10.3.2 for the other use cases. Our results show that the best defenses with AT+Augmentations (continuous lines) remain robust against increased gradient and search iteration budgets and remain more robust than AT alone (dashed lines) for VIME, RLN, and Tabtransformer architectures. Against an increase in perturbation size ϵ, AT+Augmentations is more robust than AT alone for TabNet, TabTransformer, VIME, and RLN architectures. In particular, for $\epsilon = 5$, the robust accuracy of TabNet architectures remains above 40% with AT+Augmentations while the robust accuracy with AT

alone drops to 0%.

With data augmentation and AT, ID and robust performances are correlated. Although there is no trend of relationship between ID performance and robust performance in standard training, our study shows that robustness and ID performance are correlated after adversarial training. This trend is depicted in Figure6.3. For example, the Pearson correlation between ID and robust performance increases from 0.15 to 0.76 for LCLD. All correlation values are in Appendix 10.3.2.

6.5.3 Using only data augmentation

No data augmentation consistently outperforms the baselines with AT. Among the 20 scenarios in Fig. 6.1, the original models achieve better constrained robustness than augmented models with adversarial training only for 4 scenarios: TabNet architecture on URL, LCLD and WIDS, and STG architecture on URL datasets. No data-augmentation technique consistently outperforms the others across all architectures. Cutmix, the simplest data augmentation, is often the best (in 7/20 scenarios).

Adversarial training with data augmentations outperforms data augmentation alone.

Figure 6.1 shows that in 19 cases over 20, adversarial training with data augmentation outperforms data augmentation alone. Only in the case of URL-TabNet, WGAN alone (95.2%) outperforms the best adversarial training method (91.8%).

With data augmentation alone, ID and robust performances are not aligned. In Figure 6.3 we study the impact of data augmentation on ID and robust performance, both in standard and adversarial training. With standard training, ID performance is misleading in CTU and URL datasets. Although all models exhibit similar ID performance, some of the augmentations lead to robust models, while others decrease it. CTGAN data augmentation is the best data augmentation for ID performance in all use cases, both with standard and adversarial training.

6.5.4 Evaluation with unconstrained attacks

In Figure 6.1 we study the robustness of each architecture with different defense mechanisms. We report both the robustness against unconstrained attacks (attacks unaware of domain knowledge) and attacks optimized to preserve the feature relationships and constraints.

Evaluation with unconstrained attacks is misleading. Under standard training (orange scatters in Fig. 6.1), there is no relation between robustness to unconstrained attacks and the robustness when domain constraints are enforced. There is, however, a linear relationship under adversarial training with data augmentation only for STG, Tabstransformer, and VIME architectures. These results show that nonconstrained attacks are not sufficient to reliably assess the robustness

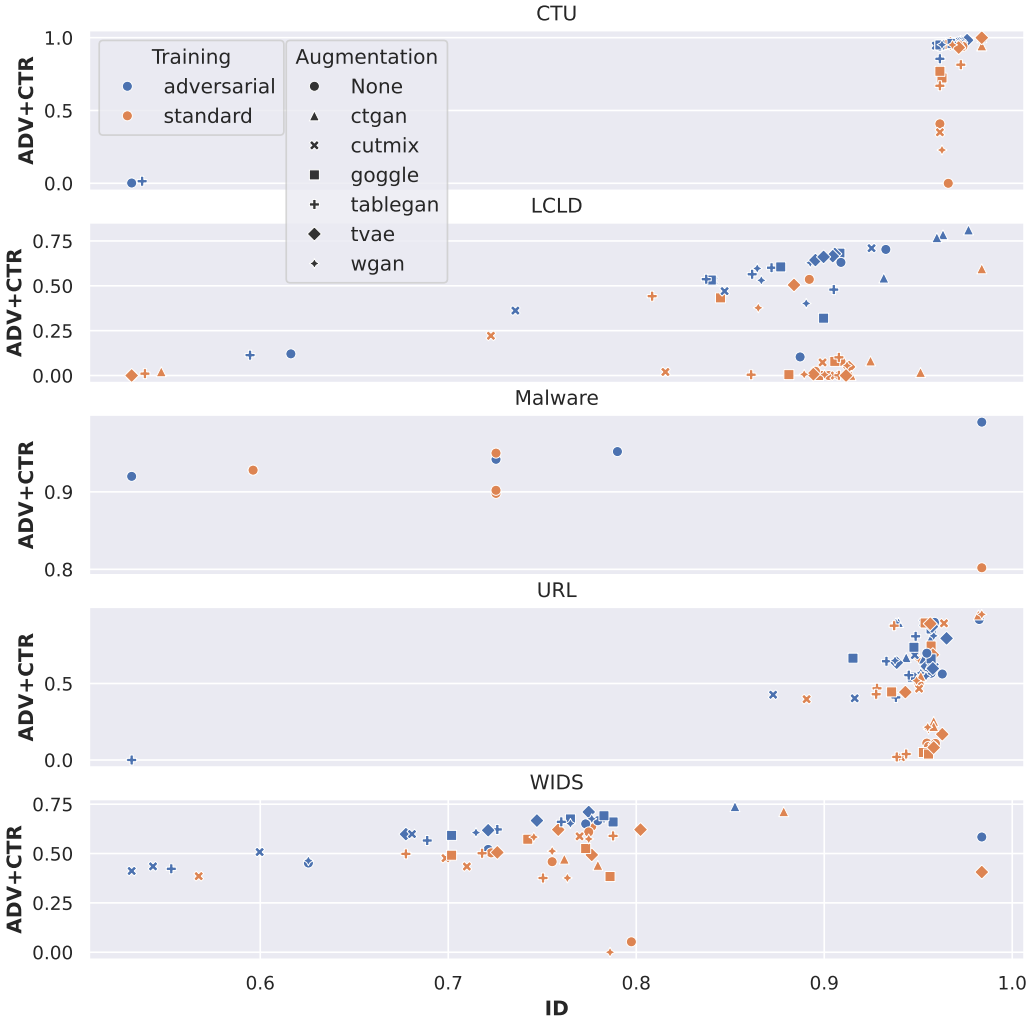


Figure 6.3: Summary of our main experiments; Y-axis: Robust Accuracy, X-axis ID accuracy

of deep tabular models. Detailed correlation values are in the Appendix 10.3.2.

6.6 Discussion: application to BGL dataset and models

Our industrial partner BGL BNP Paribas Luxembourg, a major actor in the financial industry in Luxembourg, has leveraged the capabilities of machine learning to automate services. Conceptually BGL develops ML systems to classify financial data into binary classes. It remains essential to measure its robustness to adversarial examples.

We verify the most important claims of Section 6.3:

1. CAA is an effective method to generate constrained adversarial examples.
2. Adversarial training alone is not enough to robustify models.
3. Adversarial training with data augmentation outperforms adversarial training alone.
4. Adversarial training with data augmentations outperforms data augmentation alone.

6.6.1 Empirical settings

Dataset

We apply the benchmark to the data available to BGL BNP Paribas. The objective of the ML model is to classify this data into binary classes. For simplicity, we refer to these classes as "accepted" and "rejected". The dataset contains 1,093,587 labeled inputs.

Model architecture

We consider the same architectures and hyperparameters search as for the defenses evaluation in Section 6.3. **TabTransformer**[HKC+20] and **TabNet**[AP21] are both built on transformer architectures. **RLN**[SS18] employs a regularization coefficient to reduce counterfactual loss, **STG**[YLN+20] enhances feature selection through stochastic gates, and **VIME** [YZJ+20] leverages self-supervised learning. The architectural details and training hyperparameters can be found in Appendix 10.3.1.

Defenses

We evaluate defense mechanisms based on adversarial training and data augmentation following the methods we propose in Section 6.3. We use the data augmentation methods, except for GOGGLE which was not evaluated in our partner settings due to resource limitations on our partner premises.

Table 6.4: Clean and robust performances across all architectures in the form XX/YY. XX is the accuracy of the clean examples before attack, and YY is the accuracy on their adversarial counterparts. AT: Adversarial Training, DA: Data Augmentation.

Training	RLN	STG	TabNet	TabTr.	VIME
Standard	74.3 / 43.3	73.4 / 73.0	78.3 / 26.2	23.6 / 19.2	73.9 / 08.1
AT	75.8 / 75.5	77.5 / 77.4	00.8 / 00.8	13.6 / 10.5	72.7 / 01.2
DA	77.9 / 72.0	85.2 / 71.6	60.2 / 60.2	62.0 / 09.6	77.0 / 02.5
AT + DA	77.4 / 74.8	39.2 / 39.2	00.3 / 00.3	72.5 / 70.2	74.3 / 31.9

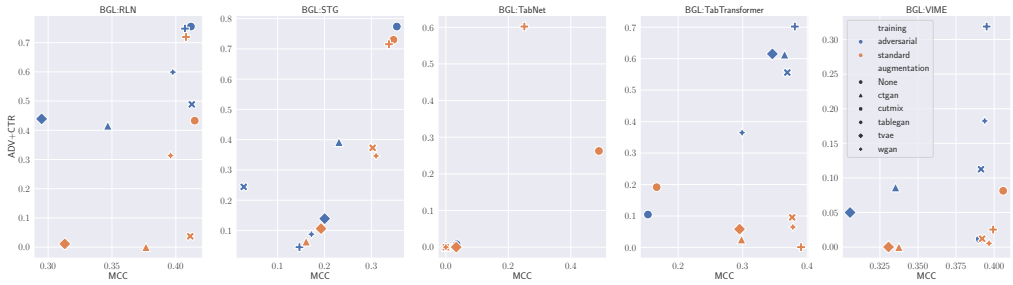


Figure 6.4: Summary of our main experiments; Y-axis: Robust Accuracy, X-axis IID MCC

Metrics

The models are fine-tuned to maximize the AUC using cross-validation. We only attack examples that belong to the critical class of rejected transactions to evaluate the robustness of the model. We use robust accuracy to evaluate the robustness of the models. That is the accuracy of examples after the attack regardless if they were correctly classified prior to perurbation.

To measure the clean performance of the models, we use Matthew's Correlation Coefficient which is not sensitive to class imbalance. We observed the limitation of accuracy in practice. STG standard training had an accuracy of 66.2% with a precision of 52.0% and a recall of 74.6% while STG WGAN and adversarial training had a higher accuracy of 66.3% with a precision of 68.5% and a recall of 11.5%.

6.6.2 Results

Table 6.4 shows the clean accuracy of examples and the robust accuracy of their respective adversarial examples. For Data Augmentation (DA) methods, we report the accuracy of the method that achieves the highest robust accuracy.

Claim 1: CAA is an effective method to generate constrained adver-

sarial examples.

The first line of Table 6.4 shows that CAA successfully generates adversarial examples in 4/5 cases. For VIME, the accuracy drops from 73.9% to 8.1%. For the most robust architecture. For STG, the model remains robust as the success rate only drops from 73.4% to 73.0%. We observe that STG remains the most robust architecture.

Claim 2: Adversarial training alone is not enough to robustify models.

Table 6.4 reveals that our second claim does not hold for all models. Adversarial training completely robustifies RLN as the accuracy only drops by 0.3%. STG is robust in clean training (0.4% drop) and remains robust with adversarial training (0.1%). Adversarial training does not robustify VIME and actually worsens the robust accuracy.

Figure 6.4 shows that adversarial training breaks the clean performance of the model for TabNet (MCC of 0.03) and TabTransformer (0.15).

Claim 3: Adversarial training with data augmentation outperforms adversarial training alone.

Our third claim does not hold for all models. The effectiveness of adversarial training with data augmentation depends on the architecture. For RLN, the robust accuracy of adversarial training remains similar with (74.8%) and without (75.5%) data augmentation. For, STG using data augmentation and adversarial training hurts the robust accuracy from 77.4% to 39.2% and the MCC from 0.35 to 0.23 for CTGAN Madry, which robust accuracy is reported in Table 6.4. For VIME, adversarial training in combination with data augmentation (31.9%) clearly outperforms adversarial training alone (1.2%).

Claim 4: Adversarial training with data augmentations outperforms data augmentation alone.

This claim holds true when none of the methods degrades clean performance. For STG, adversarial training with data augmentation decreases STG MCC from 0.35 to 0.23. For RLN, data augmentation with adversarial training increases the robust accuracy, from 72.0% to 74.8%. Similary VIME robust accuracy reaches 31.9% from 2.5%.

Overall, CAA is an effective method to generate constrained adversarial examples for our industrial application. However, the main claims of our defense study are not consistently verified across all model architectures. The degradation of the clean performance of the models when using adversarial training and data augmentation contributes to the degradation of robust accuracy.

6.7 Conclusion

In this chapter, we proposed a method to integrate synthetic examples in the Madry adversarial training in tabular deep learning. We proposed six novel defenses

based on six data augmentations proposed in or adapted from the literature.

We conducted a large-scale empirical study showing that adversarial training alone is not enough to robustify tabular deep learning models, as CAA (our attack from Chapter 5) successfully generates adversarial examples against adversarially trained models. Our key finding is that adversarial training with data augmentation outperforms adversarial training alone, resulting in robust models, even against large attack budgets.

Additionally, we apply our defense mechanisms to a real use case of BGL BNP Paribas. We confirmed that CAA is an effective method to generate constrained adversarial on a real use case. Plus, we increase the robustness of our partner's model using data augmentation and adversarial training, effectively increasing the robust accuracy from 43.3% to 75.5% while maintaining the clean performance (from an MCC of 0.418 to 0.415).

We believe our work serves as a foundation for further research in developing defenses against adversarial attacks on tabular models. To support this effort, we introduce TabularBench, the first benchmark designed to evaluate the robustness of deep learning models on tabular data (see Chapter 7).

7

TabularBench: Benchmarking Adversarial Robustness for Tabular Deep Learning in Real-world Use-cases

Building on the attacks and defenses presented in Chapter 4 to 6, we propose TabularBench, the first benchmark specifically designed for evaluating adversarial attacks and defenses within a constrained feature space. With this benchmark, we aim to support research in developing robustification methods against constrained adversarial examples.

Contents

Table 7.1: Existing related benchmarks and their differences with ours

Benchmark	**Domain**	**Metric**	**Realistic evaluation**
Tabsurvey [BLS+21]	Tabular	ID performance	No
Tableshift [GPS23]	Tabular	OOD performance	No
ARES [DFY+20]	CV	Adversarial performance	No
Robustbench [CAS+20]	CV	Adversarial performance	Yes
DecodingTrust [WCP+23]	LLM	Trust (incl adversarial)	Yes
OURS	Tabular	Adversarial performance	Yes

7.1 Introduction

In this chapter, we leverage the strong and effective attack CAA proposed in Chapter 5 and the novel defenses introduced in Chapter 6 to create an online benchmark on the clean and robust accuracy of 5 architectures from different mechanisms families and 14 training methods across 5 datasets.

Despite the widespread use of tabular data and the maturity of Deep Learning (DL) models for this field, the impact of evasion attacks on tabular data has not been thoroughly investigated. Although there are existing benchmarks for *in-distribution* (ID) tabular classification [BLS+21], and distribution shifts [GPS23], there is no available benchmark of adversarial robustness for deep tabular models, in particular in critical real-world settings. We summarize in Table 7.1 these related benchmarks.

The need for dedicated benchmarks for tabular model robustness is enhanced by the unique challenges that tabular machine learning raises compared to computer vision and NLP tasks. The most important challenge comes from *feature constraints*, which are relationships and interactions between features that are (1) hard to satisfy for an attacker but (2) can give a false sense of robustness to the defender. Attacks designed specifically for tabular data often overlook feature-type constraints [BAL+19] or, at best, consider categorical features without accounting for feature relationships [WHB+20; XHR+23; BHZ+23]. Furthermore, the different approaches also use different threat models and attacker capabilities assumptions. This limitation restricts the evaluation and comparison of attacks and defenses proposed in the literature.

Thus, the machine learning research community currently lacks a reliable and high-quality benchmark to enable the investigations of attacks and robustification mechanisms on tabular data.

Such a benchmark for tabular adversarial attacks should feature deployable attacks and defenses that reflect as accurately as possible the robustness of models

within a reasonable computational budget. A reliable benchmark should also consider recent advances in tabular deep learning architectures and data augmentation techniques, and tackle realistic attack scenarios and real-world use cases considering their domain constraints and realistic capabilities of an attacker.

To address this gap, we propose TabularBench, the first comprehensive benchmark of the robustness of tabular deep learning classification models. We use *Constrained Adaptive Attack (CAA)* introduced in Chapter 5, a combination of gradient-based and search-based attacks that have recently been shown to be the most effective against tabular models.

Our contributions to this benchmark can be summarized as follows:

- **Leaderboard** (`https://serval-uni-lu.github.io/tabularbench`): a website with a leaderboard based on *more than 200* evaluations to track the progress and the current state of the art in adversarial robustness of tabular deep learning models for each critical setting. The goal is to clearly identify the most successful ideas in tabular architectures and robust training mechanisms to accelerate progress in the field.
- **Dataset Zoo** : a collection of real and synthetic datasets generated with and without domain-constraint satisfaction, over five critical tabular machine learning use cases.
- **Model Zoo** : a collection of the most robust models that are easy to use for any downstream application. We pre-trained these models in particular on our five downstream tasks and we expect that this collection will promote the creation of more effective adversarial attacks by simplifying the evaluation process across a broad set of *over 200* models.

7.2 TabularBench: Adversarial Robustness Benchmark for Tabular Data

In Appendix 10.3.1 we report the detailed evaluation settings such as metrics, attack parameters, and hardware. We focus below on the datasets, classifiers, and synthetic data generators.

7.2.1 Tasks

We curated datasets meeting the following criteria: (1) **open source:** the datasets must be publicly available with a clear definition of the features and preprocessing, (2) **from real-world applications:** datasets that do not contain simulated data, (3) **binary classification:** datasets that support a meaningful binary classification task, and (4) **with feature relationships**: datasets that contain feature relationships and constraints, or they can be inferred directly from the definitions of features.

After an extensive review of tabular datasets, only the following five datasets

Table 7.2: Properties of the use cases of our benchmark.

Dataset	Domain	Output to flip	Total size	# Features	# Ctrs	Inbalance
CTU	Botnet detection	Malicious connections	198 128	756	360	99.3/0.7
LCLD	Credit scoring	Reject loan request	1 220 092	28	9	80/20
Malware	Malware detection	Malicious software	17 584	24 222	7	45.5/54.5
URL	Phishing	Malicious URL	11 430	63	14	50/50
WIDS	ICU survival	Expected survival	91 713	186	31	91.4/8.6

match our requirements.

The **CTU** [CO22] includes legitimate and botnet traffic from CTU University. Its challenge lies in the extensive number of linear domain constraints, totaling 360. **LCLD** [Geo18] is a credit-scoring containing accepted and rejected credit requests. It has 28 features and 9 *non-linear* constraints. The most challenging dataset of our benchmark is the **Malware** dataset prepared by [DGS+22]. The very large number of features (24222), most of which are involved in each constraint, make this dataset challenging to attack. **URL** [HY21] is a dataset comprising both legitimate and phishing URLs. Featuring only 14 linear domain constraints and 63 features, it represents the simplest case in our benchmark. The **WiDS** [LRG+20] includes medical data on the survival of patients admitted to the ICU, with only 31 linear domain constraints.

Our datasets include varying complexity in terms of number of features and constraints and diverse class imbalance intensity. We summarize the datasets and their relevant properties in Table 7.2 and provide more details in Appendix 10.3.1 .

7.2.2 Architectures

We consider five state-of-the-art deep tabular architectures from the survey by [BLS+21]: **TabTransformer** [HKC+20] and **TabNet** [AP21], are based on transformer architectures. **RLN** [SS18] uses a regularization coefficient to minimize a counterfactual loss, **STG** [YLN+20] improves feature selection using stochastic gates, while **VIME** [YZJ+20] depends on self-supervised learning. We provide in Appendix 10.3.1 the details of the architectures and the training hyperparameters. These architectures are on par with XGBoost, the top shallow machine-learning model for our applications.

7.2.3 Data Augmentation

Our benchmark considers synthetic data augmentation using five state-of-the-art tabular data generators. These generators were pre-trained to learn the distribution of the training data. Then, we augmented each of our datasets 100-fold (for example, for URL dataset, we generated 1.143.000 synthetic examples). Appendix 10.3.1 details the generator architectures and the training hyperparameters.

WGAN [ACB17] is a typical generator-discriminator GAN model using Wasser-

stein loss. We follow the implementation of [SDC+24] and apply a MinMax transformation for continuous features and one-hot encoding for categorical to adapt this architecture for tabular data.

TableGAN [PMG+18] is an improvement over standard GAN generators for tabular data. It adds a classifier (trained to learn the labels and feature relationships) to the generator-discriminator setup to improve semantic accuracy. TableGAN uses MinMax transformation for features.

CTGAN [XSC+19a] uses a conditional generator and training-by-sampling strategy in a generator-discriminator GAN architecture to model tabular data.

TVAE [XSC+19a] is an adaptation of the Variational AutoEncoder architecture for tabular data. It uses the same data transformations as CTGAN and training with ELBO loss.

GOGGLE [LQB+23] is a graph-based model that learns relational and functional dependencies in data using graphs and a message passing DNN, generating variables based on their neighborhood.

Cutmix [YHO+19] In computer vision, patches are cut and pasted among training images where the labels are also mixed proportionally. We adapted the approach to tabular ML and for each pair of rows of the same class, we randomly mix half of the features to generate a new sample.

For training, each batch of real examples is augmented with a same-size random synthetic batch (without replacement). However, the evaluation only runs on real examples. In AT, we generate adversarials from half of the real examples randomly selected and half of the synthetic examples.

7.2.4 TabularBench API

To encourage the wide adoption of TabularBench as the go-to place for Tabular Machine Learning evaluation, we designed its API to be modular, extensible, and standardized. We split its architecture into three independent components. More details of each component are provided in Appendix 10.4.1.

A dataset Zoo For each dataset in this study, we have collected, cleaned, and pre-processed the existing raw dataset. The processed datasets are loaded with a *Dataset factory*, and then the user gets their associated meta-data and pre-defined constraints. The datasets are automatically downloaded when not found.

```
ds = dataset_factory.get_dataset("lcld_v2_iid")
metadata = ds.get_metadata()
```

Constraints handling One of the features of our benchmark is the support of feature constraints, in the dataset definition and in the attacks.

Boundaries, mutability, and type constraints are defined in a CSV file, and the corresponding pandas DataFrame can be accessed via:

```
metadata = ds.get_metadata(only_x=True)
```

We implemented a novel *Constraint Parser* where the user can write the relations in a natural human-readable format to describe the relationships between features. Relation constraints are defined using Python, offering a high-level language that also provides linting and type-checking to minimize errors.

For example the following constraint:

$$\omega_1 \equiv f_0 = f_1 + f_2 \tag{7.1}$$

is expressed as:

```
from tabularbench.constraints.relation_constraint import Feature
constraint1 = Feature(0) == Feature(1) + Feature(2)
```

Features can also be accessed by names, increasing readability:

```
from tabularbench.constraints.relation_constraint import Feature
constraint2 = Feature(3) <= Feature(4)
constraint3 = Feature("open_acc") <= Feature("total_acc")
```

Here, constraints 2 and 3 are equivalent. We support the following operators:

- Base operators: Pre-built numeric operations (+,-,*,/, âĂę).
- Safe operators: SafeDivision and Log allow a fallback value if the denominator is 0.
- Constraints operators: Pre-built operator that returns a BaseRelationConstraint: OrConstraint, AndConstraint, LessConstraint, LessEqualConstraint
- Tolerance-aware constraint operators: EqualConstraint allows a tolerance value in assessing the equality constraint. == can also be used for no-tolerance equalities.

Constraints of existing datasets can be retrieved with:

```
constraints = ds.get_constraints()
relation_constraints = constraints.relation_constraints
```

We provide two executor backends that take a set of examples and a constraint as input and return their corresponding penalty function values. NumpyBackend uses a Numpy array representation and Pytorch backend uses a PyTorch tensor representation, hence preserving the gradient if required. The set of supported constraints and their translation to a penalty function can easily be extended. Our benchmark implementation uses object-oriented programming and the visitor design pattern for the executor backends.

A model Zoo Our API supports five architectures, and for each, six data augmentation techniques (as well as no data augmentation) and two training schemes (standard training and adversarial training). Hence, 70 pre-trained models for each of our five datasets are accessible. Below, we fine-tune with CAA AT and CTGAN augmentation a pre-trained Tabtransformer with Cutmix augmentation:

```
scaler = TabScaler(num_scaler="min_max", one_hot_encode=True)
scaler.fit(x, metadata["type"])
model = TabTransformer("regression", metadata, scaler=scaler,pretrained="
                                        LCLD_TabTr_Cutmix")
train_dataloader = CTGANDataLoader(dataset=ds, split="train", scaler=scaler,
                                        attack="caa")
model.fit(train_dataloader)
```

A standarized benchmark To generate our leaderboard, we offer a one-line command that loads a pre-trained model from the zoo, and reports the clean and robust accuracy of the model following our benchmark's setting (taking into consideration constraint satisfaction and L2 minimization):

```
clean_acc, robust_acc = benchmark(dataset='LCLD', model="TabTr_Cutmix",
                                        distance='L2', constraints=True)
```

7.3 Limitations

While our benchmark is the first to tackle adversarial robustness in tabular deep learning models, it does not cover all the directions of the field and focuses on domain constraints and defense mechanisms. Some of the orthogonal work is not addressed:

Generalization to other distances: We restricted our study to the L_2 distance to measure imperceptibility. Imperceptibility varies by domain, and several methods have been proposed to measure it [BAL+19; KKT22; DGS+22]. These methods have not been evaluated against human judgment or compared with one another, so there is no clear motivation to use one or another. In our research, we chose to use the well-established L_2 norm (following [DGS+22]). Our algorithms and benchmarks support other distances and definitions of imperceptibility. We provide in Appendix 10.3.2 an introduction to how our benchmark generalizes to other distances.

Generalization to non-binary classification: We restricted our study to binary tabular classification as it is the only case where we identified public datasets with domain constraints. The attacks used in our benchmark natively support multi-class classification. Our live leaderboard welcomes new datasets and will be updated if relevant datasets are designed by the community.

Generalization to other types of defenses: We only considered defenses based on data augmentation with adversarial training. Adversarial training based defenses are recognized as the only reliable defenses against evasion attack [TCB+20; Car23]. All other defenses are proven ineffective when the attacker is aware of them and performs adaptive attacks.

7.4 Conclusion

In this work, we introduce TabularBench, the first benchmark of adversarial robustness of tabular deep learning models against constrained evasion attacks. We leverage Constrained Adaptive Attack (CAA), the best constrained tabular attack, to benchmark state-of-the-art architectures and defenses.

We provide a Python API to access the datasets, along with implementations of multiple tabular deep learning architectures, and provide all our pretrained robust models directly through the API.

This benchmark was used to conduct the empirical study of Chapter 6 that constitutes the first large-scale study of tabular data model robustness against evasion attacks. Our study covers five real-world use cases, five architectures, and six data augmentation mechanisms totaling more than 200 models. Our study identifies the best augmentation mechanisms for IID performance (CTGAN) and robust performance (Cutmix), and provides actionable insights on the selection of architectures and robustification mechanisms.

We are confident that our benchmark will accelerate the research of adversarial defenses for tabular ML and welcome all contributions to improve and extend our benchmark with new realistic use cases (multiclass), models, and defenses.

This chapter concludes our research on robustness against adversarial examples. In Chapter 8 we propose a protocol to evaluate the robustness of ML models and retraining methods to distribution shift under industrial constraints.

8

On the Impact of Industrial Delays when Mitigating Distribution Drifts: an Empirical Study on Real-world Financial Systems

Research has developed drift detectors as a means to decide when to trigger model retraining to mitigate degradation of performance over time. Previous studies do not consider the industrial constraints facing ML systems in production. In this chapter, our objective is to uncover the capabilities of retraining strategies to mitigate the effect of drifts in the presence of labeling and deployment delays.

Contents

8.1 Introduction

This chapter addresses the third and last challenge of this thesis, which is the realistic evaluation of performance drift mitigation techniques for ML systems in production.

Industry players exploit Machine Learning (ML) technologies to leverage an increasingly large amount of data and reduce operational costs, develop new disruptive products, and deliver personalized services to their customers. Therefore, ML is a significant contributor to the digitalization of the industry.

Our industrial partner, the Data Science Lab of BGL BNP Paribas Luxembourg (henceforth referred to as BGL BNP Paribas) is an important actor in the financial industry in Luxembourg. The company has many operational benefits from automated services built using ML technologies, including scalability at low costs and the ability to process large amounts of data efficiently and flexibly.

However, the wide dissemination of ML technologies within industrial software systems is hindered by their high maintenance costs [ABB+19]. Our partner has observed that the effectiveness (e.g. prediction accuracy) of their ML systems declines over time due to changes in data distribution [WHC+16].

The usual solution to mitigate the effect of drifts on ML systems is to retrain the ML model periodically (periodic retraining) or continuously (online learning). In our partner's case, online learning is prohibited by stringent security policies, which prevent feeding models with live data without manual checks. Thus, our partner relies on periodic retraining. The critical questions they face are *how often* and *how* they should retrain their model.

Alternatively, research has developed *drift detectors* as a better means to decide when to trigger model retraining [LLD+19]. A drift detector is a statistics-based method that takes as input a stream of samples. After each sample, it returns whether the observed distribution has shifted or not compared to previous samples. Using drift detectors to trigger retraining at the most appropriate times can reduce the cost of periodic retraining and increase its effectiveness (in improving model performance in spite of drifts).

Although the use of periodic retraining and drift detectors has been intensively investigated in the literature [LLD+19; PCvD+22; BG07b], previous studies do not consider the industrial constraints facing ML systems in production. In particular, two types of delay inhibit the retraining process. First, *labeling delay* implies a temporal distance between the time at which new data reaches the system and the time at which its ground truth label is retrieved. Second, *deployment delay* is inherent to the strict quality assurance and manual validation processes that financial institutions (and other critical institutions) impose on their software systems. This implies that there is a significant time gap between when a software system is fully engineered (or updated) and when it is running in production. In

our experiments, we set the labeling delay to 10 days and the deployment delay to 28 days.

In this chapter, our objective is to uncover the capabilities of retraining strategies to mitigate the effect of drifts *in presence* of labeling and deployment delays. Therefore, we conduct an empirical study involving one real-world financial system of our partner, two publicly available datasets, and covering 16 retraining scheduling methods (based on periodic retraining or drift detectors). We measure the capability of these strategies to retrain models efficiently (minimizing the number of retraining) and effectively (maximizing performance over time). We specifically investigate the impact that the aforementioned delay has on existing retraining practices.

Our contributions can be summarized as follows:

1. We formulate the problem of retraining against distribution drifts in the presence of *labeling and deployment delays*.

2. We propose a novel step-by-step protocol for ML practitioners to diagnose distribution drifts with deployment delays and identify the best retraining strategies that fit their case.

3. We report on an empirical study of the effectiveness and efficiency of retraining strategies. We notably shed light on the impact of window size of retraining, the importance of drift detector tuning, and how the delay affects the Pareto-optimal retraining strategies.

Through our study, we highlight the importance that labeling and deployment delays have on the predictive maintenance of machine learning-based systems, and the scale at which these delays impact the solutions to combat distribution drifts in the real world. By providing a proper definition and evaluation protocol of this industry-relevant problem overlooked by the literature, we hope to inspire future research on designing effective and efficient solutions.

8.2 Problem

Without loss of generality, we consider a classification problem defined on a n dimensional feature space $\mathcal{X} \subseteq \mathbb{R}^n$ and a binary label space $\mathcal{Y} = \{0, 1\}$. We assume that the samples come as a time series S where each sample $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$ is indexed by a discrete time parameter t_i, such that t_i represents the time at which the input x_i reached the system. The *labeling delay* of x_i is the time between t_i and the moment x_i receives its true label y_i. For simplicity, we assume that this delay is constant across the inputs, that is, y_i is received at time $t_i + \delta_l$.

Let $h_{t_j} : \mathcal{X} \to \mathcal{Y}$ be the classification model trained at time t_j. Then h_{t_j} can

only be trained on inputs $\{x_i\}$ such that $t_i + \delta_l \leq t_j$.[1] Furthermore, the *deployment delay* of h_{t_j} is the time needed to deploy it in production. As before, we assume that this delay δ_d is constant. Thus, any model h_{t_j} can only make predictions on inputs that arrive in the system after it is deployed, that is, on inputs $\{x_k\}$ such that $t_j + \delta_d \leq t_k$.

We define a *retraining schedule* as a totally ordered sequence of times $sched = \{t_1 \dots t_n\}$ that determines when a model should be re-trained. For example, periodic retraining yields such a sequence in which all elements are exactly separated by a constant period p, that is, $t_{j+1} = t_j + p$. By contrast, drift detectors decide the schedule on the fly based on their statistical analysis of the data and the model. Any retraining schedule determines a sequence of models $H = \{h_{t_1} \dots h_{t_n}\}$ with h_{t_j} defined as above. Then, any input example x_i observed at time t_i is predicted by the latest available model, that is, the predicted label for x_i is given by $\hat{y}_i = h_{t_*}(x_i)$ where $t_* = \max\{t_k \in sched \text{ s.t. } t_k + \delta_{prod} \leq t_i\}$.

A retraining schedule can be evaluated based on its *effectiveness* and *efficiency*. Effectiveness is defined as the correctness of predictions made by the sequence of models H, calculated by an arbitrary scoring function $score(\mathcal{Y}, \hat{Y})$, with $\hat{Y} = \{\hat{y}_i\}$. Efficiency measures the overall cost of model retraining. For simplicity, in our study, we assume this cost to be constant across models (all other things being equal) and compute it as the number $n = |H|$ of retraining. This assumption makes sense in the context of our partner, where the cost of deploying models in production largely surpasses the computational cost to retrain the model and the data labeling cost (since labels are obtained through a later automated financial analysis).

8.3 Methodology

We propose a novel experimental protocol to thoroughly evaluate retraining scheduling techniques under realistic industrial constraints (labeling and deployment delays). These constraints have been overlooked by previous studies. We do so, moreover, while carefully and empirically considering alternative design decisions that affect model performance (incl. hyperparameter tuning and training window size).

Our protocol starts from an initial model m_0 trained on an initial training set $S_{train} =]d_0, d_{N_{train}}]$, which contains the first N_{train} example of the time series S. The protocol evaluates this model on the remaining samples of the time series $S_{test} = [d_{N_{train}}, d_{|S|}[$ using an arbitrary score function, which takes as input the prediction of the model and the true label. The objective of the subject scheduling techniques is to increase the score function of a set of models on this test set.

[1] Note that we define here which data are *available* to train the model. The actual choice of which data to use for retraining is a different topic on its own that we partly address in our first series of experiments.

8.3.1 Model hyperparameter tuning

The very first part of our protocol investigates whether we can improve the baseline model via hyperparameter tuning and whether repeating this tuning process at each retraining can improve the model's effectiveness.

We select the best hyperparameter tuning strategy across three strategies. "No tuning" means that we reuse the hyperparameters provided by our industrial partner. "Initial tuning" means that, before any evaluation, we train the baseline model from scratch on the training set S_{train} while applying hyperparameters tuning on the baseline model. Afterward, the hyperparameters remain the same throughout the evaluation process. "Re-tuning" means that we tune the hyperparameters each time the model is retrained (based on the data available for re-training at this time).

To tune the hyperparameters, we use K-fold validation with time series splits and Bayesian search with the objective of maximizing the average score function over the folds. Compared to random or stratified K-fold splits, time series splits have the advantage of evaluating the model generalization to future samples instead of identically distributed samples. This validation process is especially more suitable for data drift scenarios as this better measures the model robustness to drifts and, therefore, leads to better tuning.

We identify the **best tuning strategy** and reuse it for the rest of our protocol.

8.3.2 Training window size

We next need to use an appropriate window size, i.e. how many of the most recent data model retraining should use. Training with the maximum amount of data does not always produce the best model, according to the dilemma between model plasticity (learning new information) and stability (retaining previous knowledge) [LH03; GBB14].

We thus compare the effectiveness achieved by different window sizes that range from a fraction of the data up to all the available data. We select the best window size with periodic retraining schedules with different periods ranging from a period as short as the label delay up to a period corresponding to a single retraining across the entire time series. We consider scenarios without and with delays. An evaluation without delays measures maximal potential performance improvements of alternative window sizes, whereas an evaluation with delays measures the improvement in a realistic context. Thanks to this protocol, we select the **best window size** and use it to evaluate retraining schedules.

8.3.3 Drift detector evaluation

We propose a protocol to evaluate a drift detector and its parameters in a realistic scenario. To evaluate a drift detector and its parameters, we start from an

initial model m_0 trained on the training set S_{train} with the best tuning strategy and window size (as explained before).

Once a new sample $d_i = x_i, t_i, y_i$ arrives in the system, we use the initial model m_0 to predict a label $\hat{y}_i$ for x_i. We then feed our drift detectors with the feature x_i, the time t_i, the label y_i, and/or the prediction $\hat{y}_i$, depending on what the detector needs. If the detector does not detect a drift, we do nothing and wait for the next samples to arrive and be processed.

If the detector detects a drift, we retrain a model with the data $]d_{i'-window\_size}, d'_i]$, such that d'_i such that d'_i is d_i rounded up to the closest multiple of k. We round up d_i to the closest multiple of k to limit the computational cost when evaluating a large amount of retraining strategy. This enables us to reuse models when the detector being evaluated triggers model retraining at the same time as another detector (or periodic retraining) did before. This also prevents worst-case scenarios where a detector is too sensitive and attempts to retrain too frequently.

The newly trained model will become available for prediction after a delay $\delta = \delta_l + \delta_d$ with regard to t'_i. Between the detection of drift in t_i and the model being available in time $t'_i + \delta$, we continue to predict the samples with m_0.

Whether we continue to detect drift or not depends on the type of drift detector. If the drift detector is data-based, we continue to detect drift. Indeed, these detectors do not depend on the model and can continue to trigger model retraining during this time window. On the other hand, error-based and predictive-based detectors observe distributions that are dependent on the model such as the error rate or the uncertainty. Therefore, after detecting a drift, they need to wait for the new model and the new distribution to be available.

After the first drift has occurred, for the remainder of the sample $d_j, j > i$ we use the latest available model to make the prediction and follow the same process as in m_0. A model m_i is available at time t_{m_i} if and only if it has been trained with data older than $t_m - \delta_l - \delta_d$.

Thanks to this protocol, we can evaluate the **drift detectors** effectiveness (according to the score function) and efficiency (number of retraining).

8.3.4 Comparing drift detectors and periodic retraining

To evaluate the periodic retraining strategy, we use the same protocol as for the drift detectors and instantiate a drift detector that is equivalent to periodic retraining. The detector counts the number of samples it receives and does not return drift until it has seen the number of samples corresponding to the desired period. In this case, it triggers a drift and resets its counter. Note that this detector behaves like a data drift detector and can detect drift before the latest available model is used.

For periodic retraining schedules, we assume that the more often we retrain, the more effective our set of models will be at testing. Hence, we evaluate the periodic

schedule protocol with different periods to have different points of comparison on the efficiency (number of models) and effectiveness (ML metric) Pareto front.

For each detector, we tune its parameters using Bayesian search with the objective of minimizing the number of models used and maximizing the ML effectiveness (using the same score function as for the model's hyperparameters tuning) on the test set. To avoid information leakage, we split the S_{train} set into K-fold of (s_{train}, s_{val}) using the time series split. For each fold, we evaluate the effectiveness and efficiency of the drift detector parameters using the protocol of Section 8.3.3. For each detector individually, we select the evaluated parameters that are on the Pareto front of efficiency and effectiveness.

We evaluate the effectiveness and efficiency of the kept parameters of all drift detectors on the complete dataset. We build the Pareto front of all the detectors' efficiency and effectiveness and compare them with the effectiveness and efficiencies we obtain with periodic retraining schedules.

Thanks to this protocol, we identify the **best drift detectors'** and its best parameters.

8.4 Experiments

We describe below the protocol of our empirical study, then assess the impact of each step of our protocol.

8.4.1 Dataset, model and metrics

We apply our empirical study to the transaction system of BGL BNP Paribas. The objective of the ML model is to classify a transaction as accepted or refused based on the recent transaction of a particular client. The dataset contains 1,093,587 labeled inputs from transactions that occurred over a 5.6-year period. The timestamps associated with inputs are precise for a single day.

The classifier previously developed by our partner is a random forest. To comply with the regulation, our partner must have the capacity to interpret the automated decision made by the model; and tree-based models are interpretable by design and, therefore, we use a random forest architecture like our partner. Due to the sensitivity of the system, we did not work with the real model in production but have created a baseline model with the support of our partner's instructions. Therefore, we built a random forest classifier with 100 estimators up to 8-level deep.

We trained the baseline model with 400,000 samples, following our partner's recommendations. The minimum period for periodic retraining corresponds to the average label delay, which is 5,293. For simplicity, we round it down to 5,000 in our experiments.

We use Matthew's correlation coefficient (MCC) to score the prediction of our

models which is well suited for unbalanced datasets. It is important to note that, in our partner's case, even small differences in MCC correspond to a significant business impact. For example, during our experiments we noticed that a difference in 0.01 MCC corresponds to 3,000 transactions on average.

Figure 8.1 (blue curve) shows the performance of the baseline model over time, without retraining. This reveals that the model performance is not stable over time and tends to degrade after a certain point due to distribution drifts. On the first 20,000 samples, our model has an MCC of 0.5595. Later, the MCC score ranges between 0.5169 and 0.6099, which is a significant difference business-wise and alerts our partner. We observe a peak performance on the batch $[740, 760[$. After this peak, performance tends to degrade until an MCC of 0.5443 for the last batch of our evaluation. The orange curve shows the performance of the best model we could find in our study; this shows the potential benefits that appropriate retraining can produce.

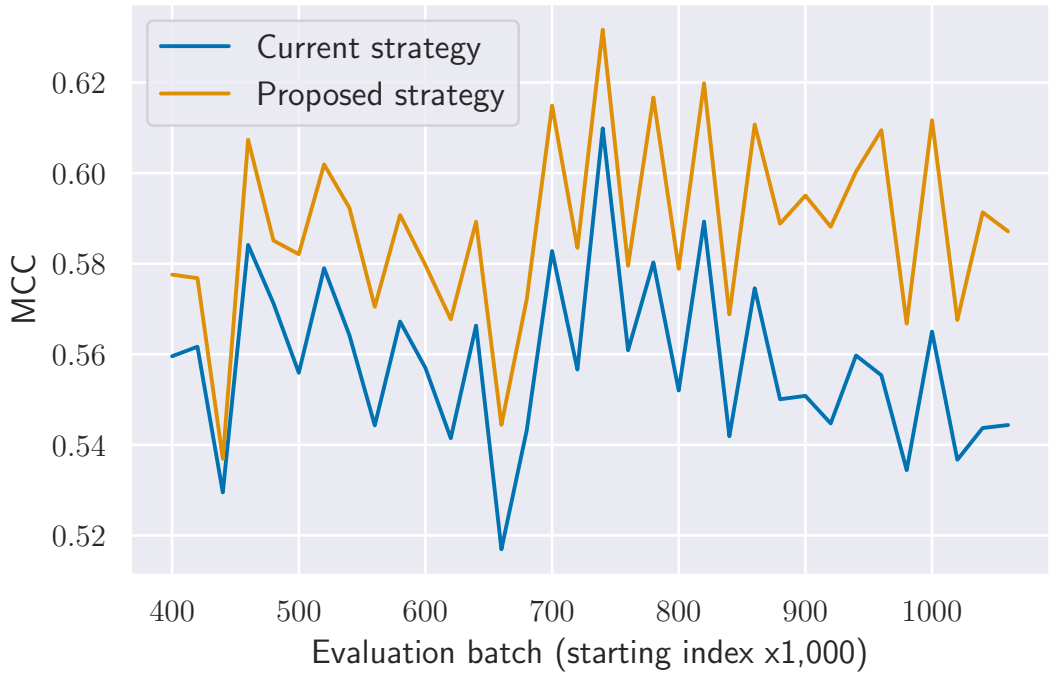


Figure 8.1: Evolution of ML effectiveness (Matthews correlation coefficient) with time (in batches of 20,000 inputs) in a scenario without retraining.

8.4.2 Results

Benefits of hyperparameter tuning

Table 8.1: ML effectiveness (MCC) of different training strategies for different testing windows. For not retrained models, the data used are the same as the initial model. For retrained model, we use the data that directly precede the test window.

Retrained	Model Hyperparameters	MCC on [T, T + 20k[at T = 400k	440k	660k	980k
No	No tuning	0.5596	0.5295	0.5169	0.5344
	Initial tuning	0.5776	0.5416	0.5433	0.5503
Yes	No tuning	-	0.5331	0.5290	0.5552
	Initial tuning	-	0.5454	0.5440	0.5718
	Re-tuning	-	0.5464	0.5448	0.5689

Table 8.2: Impact of Retraining period on ML effectiveness.

Model hyperparameters	Retraining period 20,000	50,000	100,000	200,000	400,000
No tuning	0.5678	0.5656	0.5648	0.5627	0.5616
Initial tuning	**0.5887**	**0.5877**	**0.5864**	**0.5855**	**0.5842**
Re-tuning	**0.5882**	**0.5867**	**0.5862**	**0.5848**	**0.5837**

We first investigate the benefits of re-tuning the model hyper-parameters over time. We consider three different scenarios: 1) the baseline model of our partner is used throughout ("no tuning"); 2) we tune the model based on a time split[2] ("initial tuning"); and 3) the model is re-tuned each time it is re-trained ("re-tuning"). Since we want to assess the potential of model tuning to get better models over time, we ignore labeling and deployment delays at this stage.

Table 8.1 shows the best-case scenario when tuning occurs right after the model is deployed ($T = 400k$) and right before the three most important performance drifts (T = 440k, 660k and 980k). We provide the MCC of all these models in the case where they are re-trained and in which they are not. We observe, at $T = 400k$, that our initial tuning already improves over the baseline model ("no tuning") and this improvement remains throughout the batches, regardless of whether we retrain the models or not. Moreover, even without retraining, our initially tuned model

[2]By contrast, the baseline model of our partner is tuned based on non-time-sensitive k-fold validation.

outperforms the baseline model retrained right before the different drifts. This demonstrates that proper model tuning has an even greater impact than retraining with recent data. On the contrary, re-tuning the model each time it is retrained offers little to no benefit compared to the initial tuning.

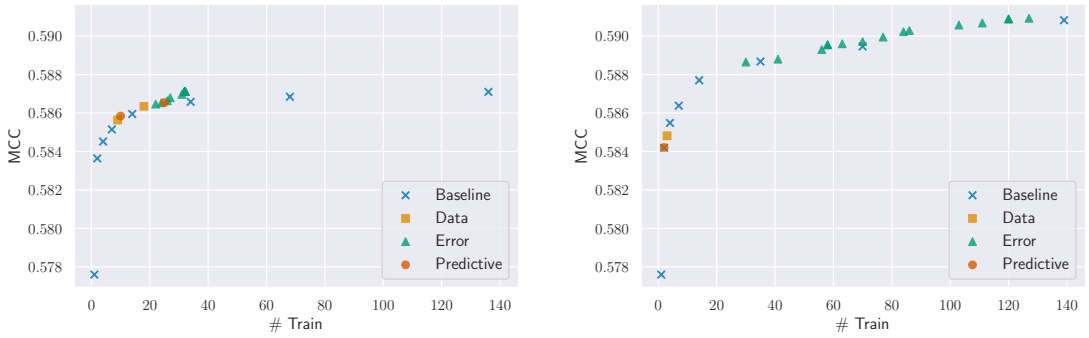
We repeat our comparison between no tuning, initial tuning, and re-tuning but this time associated with periodic retraining (using different periods). Our results – refer to Table 8.2 – confirm that initial model tuning yields significant improvements but that re-tuning at each retraining is not necessary. These conclusions are observed in all the retraining periods considered. For our study, this means that we may proceed with a careful initial tuning and skip re-tuning the model each time we retrain it.

Impact of the training window on periodic retraining.

Table 8.3: Impact of window size on ML effectiveness.

		Window Size				
Delays	Period	50,000	100,000	200,000	400,000	All
Yes	5,000	0.5753	0.5842	**0.5898**	**0.5908**	0.5896
	10,000	0.5744	0.5826	0.5882	**0.5895**	**0.5884**
	20,000	0.5728	0.5822	**0.5878**	**0.5887**	0.5877
	50,000	0.5733	0.5815	0.5868	**0.5877**	**0.5870**
	100,000	0.5702	0.5821	**0.5861**	**0.5864**	0.5849
	200,000	0.5747	0.5817	**0.5856**	**0.5855**	0.5833
	400,000	0.5749	0.5793	**0.5827**	**0.5842**	0.5810
No	5,000	0.5709	0.5804	**0.5861**	**0.5871**	0.5854
	10,000	0.5720	0.5808	**0.5859**	**0.5868**	0.5855
	20,000	0.5715	0.5809	**0.5859**	**0.5866**	0.5854
	50,000	0.5728	0.5799	0.5849	**0.5859**	**0.5852**
	100,000	0.5698	0.5814	**0.5849**	**0.5851**	0.5840
	200,000	0.5740	0.5804	**0.5844**	**0.5845**	0.5823
	400,000	0.5748	0.5789	**0.5821**	**0.5836**	0.5806

We evaluate the impact of the retraining window on the ML effectiveness when using the simple periodic retraining strategy. As before, we study this impact in the ideal scenario without delay to measure the maximum potential gains and in a realistic scenario with our partner delays. Table 8.3 reveals that using the 400,000 most recent samples to retrain is always among the 2 best solutions (in bold) without delay and the best solution with delays, independently of the retraining period. With delays, we observe that the difference in ML effectiveness between



(a) With delays $\delta_l = 10$ days, $\delta_d = 4$ weeks

(b) Without delays

Figure 8.2: Pareto front of drift detectors, no retraining and periodic retraining schedules.

400,000 samples and 200,000 samples is small (up to 0.0015 of MCC) and that both settings are acceptable. However, as the size of the window decreases further, the effectiveness is also such that for a period of 5,000 the difference between 400,000 and 50,000 is 0.0162. We also observe that using all the available data is only the third best solution – for all periods except 50,000 – and is therefore not the optimal solution. Consequently, we empirically set the retraining window used in our experiments at 400,000 examples. Beyond our study, these experiments show that the training window size can have a drastic impact on model effectiveness and therefore deserves careful consideration.

Conclusion: Tuning the hyperparameters of the initial model has a positive impact that even surpasses that of retraining. However, re-tuning at each retraining does not bring additional improvement. Training with all available data is counterproductive; empirically, we determined that the ideal window size is 400,000 samples.

Comparison of scheduling methods

We investigate the effectiveness and efficiency of different retraining schedules. In particular, we compare retraining schedules based on drift detectors with periodic retraining. Here we focus on the realistic scenario with delays of BGL BNP Paribas.

Figure 8.2a and Table 8.4 compare the efficiency (number of retraining) and effectiveness (MCC) of the different methods. In Figure 8.2a, each data point is a particular method setting, ie a scheduling method (periodic retraining or drift detector) with given parameter values. A method setting dominates another if it is higher and/or more to the left. In Table 8.4, for each method the right number

Table 8.4: For each schedule, we report the number of parameter settings (A) / (B). (A) are the settings on the efficiency/effectiveness Pareto front across all methods. (B) are the settings on the Pareto front local to the method.

Type	Schedule	No	Delay $\delta_l = 10$ days $\delta_d = 4$ weeks
Baseline	No detection	1 /1	1 /1
	Periodic	4 / 7	4 / 7
Data-based detector	Statistical test	0 / 25	0 / 25
	Divergence	1 / 4	2 / 6
	PCA-CD	0 / 3	0 / 4
Error-based detector	ADWIN (CE)	3 / 9	1 / 4
	ADWIN (PE)	1 / 9	1 / 3
	DDM	0 / 4	0 / 3
	EDDM	0 / 3	0 / 4
	HDDM-A	5 / 6	10 / 12
	HDDM-W	2 / 2	17 / 17
	KSWIN (CE)	1 / 8	0 / 7
	KSWIN (PE)	1 / 3	1 / 3
	Page-Hinkley (CE)	2 / 8	1 / 3
	Page-Hinkley (PE)	0 / 3	0 / 2
Predictive-based detector	Uncertainty	1 / 14	1 / 11
	Aries ADWIN	0 / 4	1 / 4

shows the number of parameter settings that are Pareto-optimal *within* this method (i.e., the best settings of this particular method); the left number shows the number of these settings that are Pareto-optimal *across* all methods. In this section, we consider only the second column (with delay).

We first notice that all types of scheduling methods (periodic and the three types of detectors) are on the Pareto front. By definition, the no-retraining strategy is also on the Pareto front, since it is inherently the most efficient. We filtered out drift detector schedules that were equivalent to "no retraining" or "always retraining".

Periodic retraining and error-based detectors together offer a flexible compromise between effectiveness and efficiency. Figure 8.2a shows that for a retraining budget greater than 22, error-based detectors are at least as effective as periodic retraining with less retraining. For example, the HDDM-W detector (rightmost triangle)

can achieve better performance with only 32 retraining than the most effective periodic retraining strategy that uses 136 retraining. Similarly, Page-Hinkley (CE) (third triangle from the left) obtains slightly better performance using 26 retraining than periodic retraining every 20,000 samples and using 34 retraining. Until seven retrainings, periodic retraining remains the most effective retraining strategy. Between 7 and 22 retraining, there is no clear advantage of effectiveness and efficiency of using drift detectors over periodic retraining schedules.

The second column of Table 8.4 shows that within a type of drift detector, the capacity of each detector to land on the effectiveness/efficiency Pareto front varies. The statistical test and PCA-CD detector fail to generate a single detector setting on the Pareto front, whereas the divergence detector does. For error-based detectors, DDM, EDDM, KSWIN (CE), and Page-Hinkley (PE) also fail to reach the Pareto front in any setting. Both predictive-based detectors reach the Pareto front.

Conclusion: Periodic retraining and error-based detectors together offer a flexible compromise between effectiveness and efficiency. Periodic retraining is most effective for lower numbers of retraining (up to 22). For a higher number of retrains (22+), error-based detectors are the only retraining schedule on the Pareto front; with only 32 retraining, they achieve better performance than any schedule including periodic retraining with 136 training.

8.4.3 Generalization study

Without delays

To emphasize the importance of considering the delay in the evaluation of retraining schedules, we compare the drift detector and periodic retraining schedules with and without delay. We find that not considering delay overestimates the effectiveness/efficiency trade-off of retraining schedules. Indeed, comparing Figure 8.2a (with delays) with Figure 8.2b (without delays), we observe that the Pareto front lies systematically higher when there is no delay. For example, the most effective strategy without delays has an MCC of 0.5909 for Page-Hinkley (CE), while with delays, the most effective strategy has an MCC of 0.5871 for HDDM-W.

The relative ranking between the methods also changes, indicating that the optimal drift detection method without delay does not remain optimal if delays occur. Table 8.4 compares the number of method settings (for each method) that are on the Pareto front, in the cases without delays (left column) and with delays (right column). We see that the scheduling method settings on the Pareto front are different in the two cases. For instance, KSWIN (CE) had one setting on the Pareto front without delay, but this setting disappears from the front when there is delay. Hence, not only are the expected effectiveness and efficiency overestimated but also which method settings are Pareto-optimal.

Conclusion: Not considering delay overestimates the effectiveness/efficiency trade-off of retraining schedules. The relative ranking between the methods also changes, indicating that the optimal drift detection method without delay does not remain optimal if delays occur.

Impact of delays

We study the impact of varying delays on the effectiveness and efficiency of retraining scheduling methods. More precisely, we simulate the scenario where the deployment delay is increased or decreased after the methods are tuned and running. We tune the detectors based on the previously used delays δ_l and δ_d and then evaluate them in the cases where δ_d is halved or doubled.

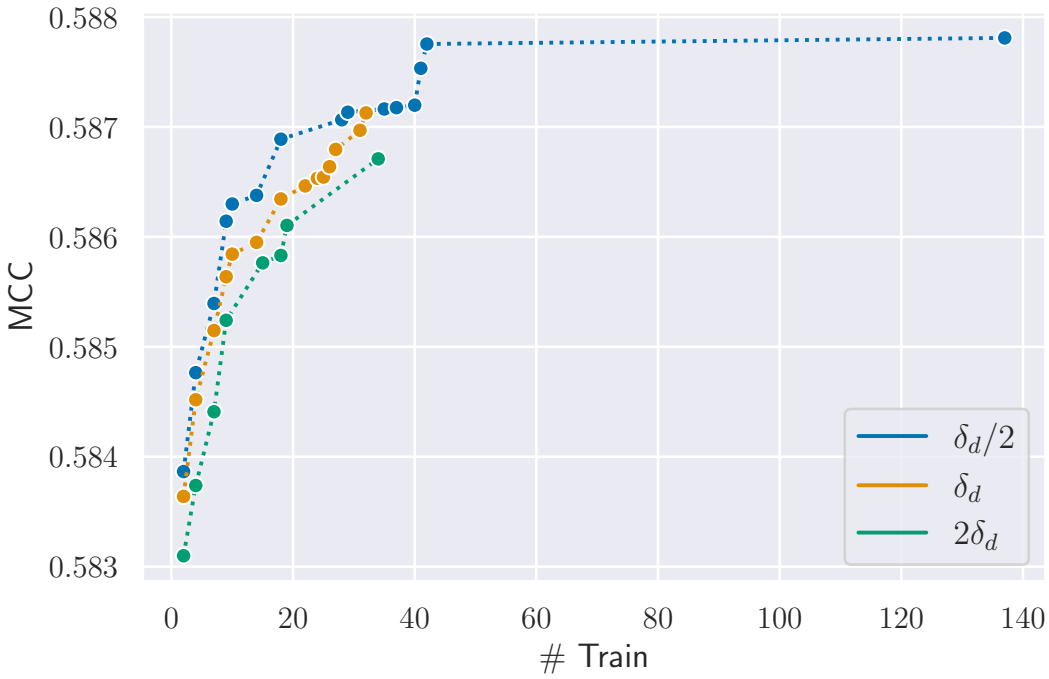


Figure 8.3: Pareto front of retraining schedule with same parameters and different deployment delays.

Figure 8.3 compares the Pareto fronts in the cases where δ_d is halves, unchanged, and doubled. We observe that the $\delta_d/2$ front dominates the δ_d front, which itself dominates the $2\delta_d$ one. This indicates that a reduction in deployment delay (compared to the delay considered when tuning the drift detectors) yields improvement, whereas an augmented delay incurs a loss in the effectiveness/efficiency trade-off.

Table 8.5: Parameters on the Pareto front with delay δ_d that remains on the Pareto front for delays $\delta_d/2$ and $2\delta_d$.

Type	Detector	Delay $\delta_d/2$	Delay δ_d	Delay $2\delta_d$
Baseline	No detection	1	1	1
	Periodic	4	4	3
Data-based detector	Statistical test	0	0	0
	Divergence	2	2	2
	PCA-CD	0	0	0
Error-based detector	ADWIN (CE)	0	1	0
	ADWIN (PE)	0	1	0
	DDM	0	0	0
	EDDM	0	0	0
	HDDM-A	0	10	10
	HDDM-W	1	17	16
	KSWIN (CE)	0	0	0
	KSWIN (PE)	0	1	0
	Page-Hinkley (CE)	0	1	0
	Page-Hinkley (PE)	0	0	0
Predictive-based detector	Uncertainty	1	1	0
	Aries ADWIN	0	1	0

Table 8.6: Summary of the generalization study. ✓means the claim holds, ✗ means the claim does not hold, and ✓* means the claim holds in some cases.

	Claim	BGL	LCLD	Electricity
Model	Tuning hyperparameters has a positive impact on performance	✓	✓	✓
	Tuning hyperparameters outperforms model retraining	✓	✓*	✗
	Re-tuning does not bring additional improvements	✓	✓*	✗
	Training with all available data is counterproductive	✓	✓	✓
Detector selection	Periodic retraining + error based offer flexible compromise	✓	✗	✓*
	Periodic retraining is effective for low retraining budgets	✓	✗	✗
	Error-based detectors are best for high retraining budgets	✓	✓	✓
	In lower budgets, error-based detectors outperforms periodic	✓	✓	✓
Impact of delay	Not considering delays overestimates the trade-off	✓	✓	✓
	Enabling delay disrupt the ranking of drift detectors	✓	✓	✓
	Change in delay has an inverse effect on efficiency/effectiveness	✓	✓	✓
	Change in delay disrupts the ranking of drift detectors	✓	✓	✓

Table 8.5 shows the number of schedules (method settings) that are on the Pareto front for the original delay δ_d (central column) and how many of these exact schedules remain on the front for delays $\delta/2$ (left column) and 2δ (right column). A retraining schedule generalizes if it remains on the Pareto front in spite of the delay change. We observe that only three methods generalize to the halved and doubled delays: periodic retraining, HDDM-W, and divergence. For HDDM-W, the parameters that generalize when augmenting the delay are different than the ones when reducing the delay. Apart from periodic retraining, the divergence-based detector is the only one that generalizes to both augmentation and reduction of the deployment delay. The uncertainty detector only generalizes to a reduction of the delays. All other methods do not generalize.

Conclusion: Changes in deployment delay have an opposite effect on the effectiveness/efficiency Pareto front. Such changes also disrupt the relative ranking of the retraining scheduling methods, such that only periodic retraining and divergence-based detection remain Pareto-optimal across the different delay values.

Generalization to public datasets and models

We evaluated our protocol on two publicly available datasets, a credit scoring dataset – the Lending Club Loan Data (LCLD)[Kag19] and the widely used Electricity [Har99]. The description of the datasets and the complete generalization study can be found in Appendix 10.5.1.

We summarize the generalization of our claims in Table 8.6. Our generalization study confirms that driving the retraining with drift detectors yields better efficiency/effectiveness trade-off than periodic retraining and our protocol leads to the optimal drift detectors and retraining strategies for unseen datasets. We provide the implementation of our protocol and its application on our two public datasets to facilitate and encourage research in the direction of distribution drift mitigation in a realistic context.

8.5 Conclusion

In this chapter, we studied the problem of maintaining the performance of ML models with drift detector triggered retrain in the presence of delays. We considered an industrial use case where the label arrives ten days after the prediction and the model goes through a four-week validation phase before deployment. To this end, we evaluated 15 drift detectors in three different scenarios: the ideal scenario without delay, a realistic scenario with delays, and finally a scenario with varying delays. We show that drift detector evaluation without delays tends to overestimate its capability to mitigate the model performance drift. We also show that although

periodic retraining is effective in the absence of delays, drift detectors are more effective in a realistic scenario. Finally, we showed that error-based detectors remain effective even though they are a reactive approach that needs the latest model to predict drift, hence being more impacted by the delay. Through this work, we aim to encourage future research to develop advanced drift mitigation techniques, including, but not limited to, drift detectors that consider realistic scenarios. Such an approach is essential for ensuring the practical applicability of these techniques in industrial contexts.

9

Conclusion

This chapter proposes the overall conclusion of this dissertation and suggests potential future research direction.

Contents

9.1 Summary of contributions

The main objective of this thesis was to enhance the robustness of critical industrial systems to adversarial examples and distribution drift. We presented methods, tools, and empirical evaluations that contribute to properly evaluating the robustness of critical systems.

The first contribution shed light on the importance of considering domain constraints when evaluating the robustness of critical industrial systems. Our analysis demonstrated that not considering these constraints leads to adversarial attacks generating unfeasible examples. Hence, using such evaluations introduces a bias in the robustness evaluation. We lay out a framework to express domain constraints as a penalty function and develop adversarial attacks that respect domain constraints. We instantiated our framework with two attacks: CPGD, a gradient attack, and MOEVA, a strong search-based attack using a genetic algorithm. Such attacks allow the correct evaluation of critical industrial systems in the constrained feature space. We showed that such attacks can fool shallow models in the financial and cybersecurity domains. With the second contribution, our objective was to better understand the robustness of deep learning models to constrained adversarial attacks. Inspired by mechanisms developed in computer vision, we improved our gradient attack and proposed CAPGD by integrating step size selection, multiple initialization, and gradient step momentum. Additionally, we added a generic repair operator for equality constraints. We demonstrated the benefits and complementarity of these mechanisms. We demonstrated that CAPGD subsumes all other gradient attacks in generating constrained adversarial examples. Then, we showed the complementarity of CAPGD and MOEVA and that their combination subsumes all other searched-based methods. Therefore, we proposed CAA, the sequential application of CAPGD and MOEVA. We showed this attack to outperform all other attacks (by up to 96.1% for CAPGD and 21.9% for MOEVA) while being up to five times faster than MOEVA. We aim for CAA to become the standard for adversarial testing of critical systems under domain constraints.

Our second contribution tackles the problem of enhancing the robustness of critical systems to constrained adversarial examples. We showed that adversarial training alone, although effective in augmenting the robustness, is not enough to fully robustify models. Learning from the computer vision literature, we built a collection of defenses based on the combination of synthetic data augmentation with adversarial training. We show these methods to outperform adversarial training. We validated our main claims on a real-world use case with BGL BNP Paribas. We showed that CAA is effective in evaluating the robustness of domain-constrained systems, that adversarial training alone is not sufficient but that its combination with synthetic data generation can produce robust models, depending

on the architecture. We released TabularBench, a benchmark to systematically track progress in adversarial attacks and defenses in the constrained feature space. Notably, we publicly release 5 datasets with their associated domain constraints, 5 model architecture implementations, 14 training methods, and a model zoo of over 200 pre-trained-models. An important technical contribution of this benchmark is the release of a Python-based API to define constraints, a parser to translate these constraints into an Abstract Syntax Tree (AST), and a visitor to translate this AST into a penalty function that can be integrated into constrained attacks.

For our third and last contribution, we studied the problem of maintaining the performance of ML models over time, under real-world constraints. We demonstrated the importance of considering real-world delays, introduced by the complex lifecycle of ML models when deployed in critical contexts. Indeed, in such a context, validation often includes manual testing to verify the correct behavior of the systems, which creates a delay. Additionally, the data collection process, in particular acquiring labels for recent examples is another source of delays. We demonstrated that overlooking such delays when designing a maintenance strategy based on retraining, can lead to the selection of a suboptimal strategy. For example, on BGL BNP Paribas's system, we showed that although simple periodic retraining is effective without delays, a strategy based on drift detection in the error rate of models should be favored in the presence of delays.

9.2 Perspectives

In the following, we discuss potential future research that follows the contributions and ideas presented in this dissertation:

- **Threat models beyond L_p norms.** In future work, it would be valuable to expand the study of adversarial examples by exploring alternative approaches to measuring imperceptibility, beyond the L_p norm used in our research. Imperceptibility is domain-specific, and various methods have been proposed to quantify it [BAL+19; KKT22; DGS+22]. However, these methods have yet to be thoroughly evaluated against human judgment or compared to one another, leaving the choice of metric unclear. Alternatively, approaches that emphasize cost and utility rather than semantic similarity, have recently been proposed [BDF+24; KKT22]. For instance, [BDF+24] considers the cost of modifying each feature as a metric to minimize through adversarial search. In addition to this cost, [KKT22] considers the utility (financial benefits) of modifying a feature and aims at finding an optimal trade-off between cost and utility. These methods enable a more accurate quantification of the risk posed by adversarial examples. We can anticipate that the likelihood of a costly but low-utility adversarial example reaching the system will be lower than that of a cheaper but highly useful example. Future research could

involve systematically assessing these methods, in combination with complex domain constraints, to determine their capability to generate increasingly realistic examples. Our current algorithms and benchmarks support multiple distance metrics and definitions of imperceptibility, and an overview of how our benchmark can be generalized to other distances is provided in Appendix 10.3.2. Further investigation into these aspects could significantly enhance the robustness and applicability of adversarial example studies across different domains.

- **Towards more realistic attacker capabilities: attacking the system as a whole.** In this dissertation, we focused on the robustness of individual models against constrained adversarial examples. While considering the domain constraints is a first step towards considering the integration of the model in a broader system, defended by a validity gate, it is not sufficient to evaluate the robustness of the system as a whole. For instance, we assumed that the attacker had full knowledge of the domain constraints, which may not be the case in practice. To address this issue, we could limit the capabilities of the attacker to query the system, returning rejected if the constraints are not satisfied or if the ML model predicts the example as rejected and accepted otherwise. From this perspective, the attacker would have to query the system to gain information about the constraints and the model, making the attack more realistic but also more challenging to execute. Another assumption we made regards the availability of the models. We assume that the attacker has access to the model, which may not hold in practice. To address this issue, we could limit the attacker's capabilities to query the system, either by the number of queries or by the types of queries allowed (e.g. constraints queries, ML model queries). Later, evaluating the robustness of the model, given a query budget for the attacker in scenarios where they conduct a query attack or attempt to build a surrogate model, would be a valuable research direction. These research directions would increase the realism and relevance of the evaluation of the robustness of critical systems to adversarial examples.
- **Exploring the interaction between adversarial examples and distribution drift.** In this dissertation, we independently studied the robustness of critical ML models against adversarial examples and distribution drift. For future work, it would be valuable to explore the interaction between these two phenomena. For example, it would be interesting to investigate whether realistic adversarial examples could assist in detecting distribution drift. A first step could involve examining whether it is easier to generate realistic adversarial examples from in-distribution samples or out-of-distribution samples. Alternatively, other metrics, such as aggressiveness [DGS+22], could be studied to differentiate between in-distribution and out-of-distribution

samples. The second step would involve distinguishing (based on adversarial metrics) between shifted examples, with respect to the input distribution, that causes misclassification and those for which the model successfully generalizes. If a correlation exists, it could lead to the development of a metric that estimates the likelihood of an example causing misclassification, thus enabling the creation of a drift detector that does not require true labels.

10

Appendices

This chapter contains the appendices of each chapter of the main text.

Contents

10.1 A Unified Framework for Adversarial Attacks in Constraint Feature Space

10.1.1 Extension to multi-class classification tasks

To simplify, we limited the definition of the problem and its solutions to binary classifiers. We propose an extension of the problem to multi-class classification tasks and show how MOEVA can be extended to handle these use cases. We consider a n-dimensional feature space $\mathcal{X}$ over the feature set $F = \{f_1, f_2, ..f_n\}$. For simplicity, we assume $\mathcal{X}$ to be normalized such that $\mathcal{X} \subseteq [0,1]^n$. Let $\mathcal{Y} = \{1, 2, ..., K\}$ be the set of labels of a K-class classification task. Let a function $H : \mathcal{X} \to \mathcal{Y}$ be a K-class classifier and $h_k : \mathcal{X} \to [0,1]$ be a single output predictor that predicts a continuous probability score of x to belong to class k. We can induce H from h with $H(x) = \arg\max_k(\{h_k(x)|k \in \mathcal{Y}\})$. For multi-class classification tasks, one must consider the targeted and untargeted scenarios.

Given an original example x classified as c and a $\mathcal{X}_\Omega$ the subset of $\mathcal{X}$ that satisfies the set Ω of domain constraints the attack objective of

1. a *targeted attack* towards class $\hat{y} \neq y$ is to generate an adversarial example $x + \delta$ such that $H(x + \delta) = \hat{y}$, $\delta < \epsilon$

2. an *untargeted attack* is to generate an adversarial example $x + \delta$ such that $H(x + \delta) \neq y$, $\delta < \epsilon$

for a maximal perturbation threshold ϵ under a given $p - norm$, and $x + \delta \in \mathcal{X}_\Omega$.

These 2 types of attack can be solved by updating the objective functions of MOEVA. One can update the objective function$g_1(x)$ as follows:

1. *targeted attack* towards class $\hat{y} \neq y$: $g_1(x) \equiv 1 - h_{\hat{y}}(x)$ (as we aim to maximize the probability $h_{\hat{y}}(x)$, we use $1 - h_{\hat{y}}(x)$ to obtain a minimization towards 0 problem)

2. *untargeted attack*: $g_1(x) \equiv h_y(x)$

while the objective functions g_2 (perturbation) and g_3 (constraints penalty) remain unchanged.

10.1.2 Experimental settings

Dataset and models

We evaluate CPGD and MOEVA on four datasets of different sizes, number of features, and types (and number) of constraints: Botnet attacks detection, credit scoring, malware detection, and URL phishing detection.

[CO19b], [AGM+20] and [HY21] showed that Random Forest is the most effective model architecture to correctly classify respectively, botnet attacks, malwares, and phishing attacks on our datasets. [GCG+20] demonstrated that Random Forest for a credit scoring task is among the most effective techniques. As they point out, this model architecture is particularly interesting for its performance while maintaining interpretability, a common requirement in the financial domain.

Credit scoring - LCLD: We engineer a dataset from the publicly available Lending Club Loan Data (*https://www.kaggle.com/wordsforthewise/lending-club*). This dataset contains 151 features, and each example represents a loan that was accepted by the Lending Club. However, among these accepted loans, some are not repaid and charged off instead. Our goal is to predict, at the request time, whether the borrower will be repaid or charged off. This dataset has been studied by multiple practitioners on Kaggle. However, the original version of the dataset contains only raw data and to the extent of our knowledge, there is no featured engineered version commonly used. In particular, one shall be careful when reusing feature-engineered versions, as most of the versions proposed present data leakage in the training set that makes the prediction trivial. Therefore, we propose our own feature engineering. The original dataset contains 151 features. We remove the example for which the feature "loan status" is different from "Fully paid" or "Charged Off" as these represent the only final status of a loan: for other values, the outcome is still uncertain. For our binary classifier, a 'Fully paid" loan is represented as 0 and a "Charged Off" as 1. We start by removing all features that are not set for more than 30% of the examples in the training set. We also remove all features that are not available at loan request time, as this would introduce bias. We impute the features that are redundant (e.g. grade and sub-grade) or too granular (e.g. address) to be useful for classification. Finally, we use one-hot encoding for categorical features. We obtain 47 input features and one target feature. We split the dataset using random sampling stratified on the target class and obtain a training set of 915K examples and a testing set of 305K. They are both unbalanced, with only 20% of charged-off loans (class 1). We trained a neural network to classify accepted and rejected loans. It has 3 fully connected hidden layers with 64, 32, and 16 neurons. Our model achieved an AUROC score of 0.7236 Our random forest model with 125 estimators reaches an AUROC score of 0.72

For each feature of this dataset, we define boundary constraints as the extremum value observed in the training set. We consider the 19 features that are under the control of the Lending Club as immutable. We identify 9 relationship constraints (3 linear, and 6 non-linear ones).

Botnet attacks - CTU-13: This is a feature-engineered version of CTU-13 proposed by [CO19b]. It includes a mix of legit and botnet traffic flows from the CTU University campus. Chernikova et al. aggregated the raw network data related

to packets, duration, and bytes for each port from a list of commonly used ports. The dataset is made of 143K training examples and 55K testing examples, with 0.74% examples labeled in the botnet class (traffic that a botnet generates). Data have 756 features, including 432 mutable features. We trained a neural network to classify legitimate and botnet traffic. It has 3 fully connected hidden layers with 64, 64, and 32 neurons. Our model achieved an AUROC score of 0.9967. We identified two types of constraints that determine what feasible traffic data is. The first type concerns the number of connections and requires that an attacker cannot decrease it. The second type is inherent constraints in network communications (e.g. maximum packet size for TCP/UDP ports is 1500 bytes). In total, we identified 360 constraints.

In addition, we build and train a random forest classifier to detect botnet attacks. Our model reaches an AUROC score of 0.9925

Malware detection - AIMED: Malwares are a major threat to IT systems security. With the recent improvement of machine learning techniques, practitioners and researchers have developed ML-based detection systems to discriminate malicious software from benign software [UAB19]. Such systems are vulnerable to adversarial attacks as shown by [CSD19] with the AIMED attack: they successfully evade the classifier without reducing the malicious effect of the software. We use the dataset of benign and malicious portable executable provided in [AGM+20]. In the same paper, the authors showed that including packed and unpacked benign executables with malicious ones is less biased towards detecting the packing as a sign of maliciousness. Therefore, we select 4396 packed benign, 4396 unpacked benign, and 8792 malicious executables. As in [AGM+20], we extract a set of static features: PE headers, PE sections, DLL imports, API imports, Rich Header, File generic. In total, we obtain a dataset of 17 584 samples and 24 222 features. We use 85% of the dataset for training and validation and the remaining 15% for testing and adversarial generation. The trained random forest classifier reaches a test AUROC of 0.9957.

From the 24 222 features, we identify 88 immutable features based on the PE format description from Microsoft. We also extract feature relation constraints from the original PE file examples we collected and those generated by AIMED. For example, the sum of binary features set to 1 that describe API imports should be less than the value of features api_nb, which represents the total number of imports on the PE file.

URL Phishing - ISCX-URL2016: Phishing attacks are usually used to conduct cyber fraud or identity theft. This kind of attack takes the form of a URL that reassembles a legitimate URL (e.g. user's favorite e-commerce platform) but redirects to a fraudulent website that asks the user for their personal or banking data. [HY21] extracted features from legitimate and fraudulent URLs as well as

external service-based features to build a classifier that can differentiate fraudulent URLs from legitimate ones. The feature extracted from the URL includes the number of special substrings such as “www", “&", “,", “$", "and", the length of the URL, the port, the appearance of a brand in the domain, in a subdomain or in the path, and the inclusion of “http" or “https". External service-based features include the Google index, the page rank, and the presence of the domain in the DNS records. The complete list of features is present in the replication package. [HY21] provide a dataset of 5715 legit and 5715 malicious URLs. We use 75% of the dataset for training and validation and the remaining 25% for testing and adversarial generation. The random forest model obtains an AUROC score of 0.9676.

We extract a set of 14 relation constraints between the URL features. Among them, 7 are linear constraints (e.g. length of the hostname is less or equal to the length of the URL) and 7 are Boolean constraints of the type $if a > 0$ then $b > 0$ (e.g. if the number of http > 0 then the number slash “/" > 0).

Experimental setup and parameters

We propose to study the effectiveness of MOEVA attack against 4 models per dataset: the first one is trained with clean samples only, the second one with adversarial examples generated on the training set using MOEVA in addition to the clean samples, the third by augmenting the features and constraints of the clean samples and the fourth by combining adversarial retraining and constraints augmentation. CPGD requires a gradient-based model which is not the case of random forests therefore we do not evaluate this attack on the random forest models.

For the LCLD and CTU-13 datasets, we reuse the same maximum perturbation threshold as for the neural network. That is 0.05 in defense and 0.2 in attack for LCLD, and 1.0 in defense, and 4.0 in attack for CTU-13. For Malware and URL, we use the same threshold as LCLD. As for the original study, the budget of the attack is set to 1000 generations for CTU-13, as our preliminary study showed that the attack was not effective with a limited budget on CTU-13. We use 100 generations for the other datasets and obtain a similar success rate as for LCLD. We discuss the choice of these parameters in the next section.

Implementation and hardware

We implement MOEVA as a framework using Python 3.8.8. We use Pymoo’s implementation of genetic algorithms and operators. Our models are trained using Tensorflow 2.5. Moreover, MOEVA is compatible with any classifier that can return the prediction probabilities for a given input x, no matter the framework used to train it. For the CPGD approach, we extend the implementation of PGD proposed by Trusted AI in the Adversarial Robustness Toolbox [NST+18]. We also extend

the Papernot attack implementation from the safe toolbox to support random forests. We run our experiments with neural networks on 2 Xeon E5-2680v4 @ 2.4GHz for a total of 28 cores with 128 GB of RAM. The ones with random forests are run on 2 AMD Epyc ROME 7H12 @ 2.6 GHz for a total of 128 cores with 256GB of RAM.

10.1.3 Hyper-parameters evaluation

Our approaches present a large number of hyper-parameters. We hypothesize that two of them have a direct impact on the success rate. First, as commonly admit, the maximum allowed perturbation ϵ. Given a large enough perturbation ϵ, an attack shall always be able to find an adversarial, as even a legit input from the target class becomes an "adversarial". Second, we hypothesize that the number of iterations given to the CPGD attack impacts the success rate. Concerning MOEVA, and as commonly admitted for genetic algorithms, the number of generations shall have an impact on the success rate of the attack. Preliminary experiments gave us an approximation of what would be a good value for these parameters. We chose an ϵ value of 0.2 and 4, respectively, for the LCLD and CTU-13 projects. Note that the maximum perturbation for normalized features between $[0, 1]$ and L_2 distance is $\sqrt{n}$ where n is the number of features, which is 6.86 and 27.48 for LCLD and CTU-13 respectively. With MOEVA attack, we use 100 and 1000 generation for LCLD and CTU-13 respectively. We started by using the same budget for both datasets. We found on a trial run on CTU-13 that, with only 100 generation, the objective function g_1 (misclassification probability) was slowly decreasing over the generation, but was not able to reach the threshold. Therefore, we attempted to augment the budget to 1000. We propose to study the impact of ϵ and the number of generations/iterations on the success rate $C\&M$. We study the variation of the success rate of our four attacks for $\epsilon \in [0.025, 0.5, 0.1, 0.2, 0.4]$ (LCLD) and $\epsilon \in [0.5, 1, 2, 4, 8]$ (CTU-13) with the aforementioned number of generations/iterations.

Figure 10.1 shows the success rate of our four attacks for different ϵ budgets on the LCLD use case. We see that CPGD reaches a plateau for $\epsilon = 0.1$, while MOEVA does not show any significant improvement after $\epsilon = 0.2$. We want to have a high success rate on both attacks for our main experiments to properly assess the impact of our defense methods. Therefore, we chose $\epsilon = 0.2$. For the CTU-13 use case, we obtained similar results for all ϵ and for both CPGD and MOEVA.

We also study the impact of the number of generations/iterations on the success rate, with ϵ fixed to the aforementioned values.

Table 10.1 shows that MOEVA requires different budgets depending on the use case to reach a similar success rate. Additionally, we observe a high success rate from 50 generations and from 500 generations for LCLD and CTU-13 respectively. To properly assess the effectiveness of the defense, and not be limited by the

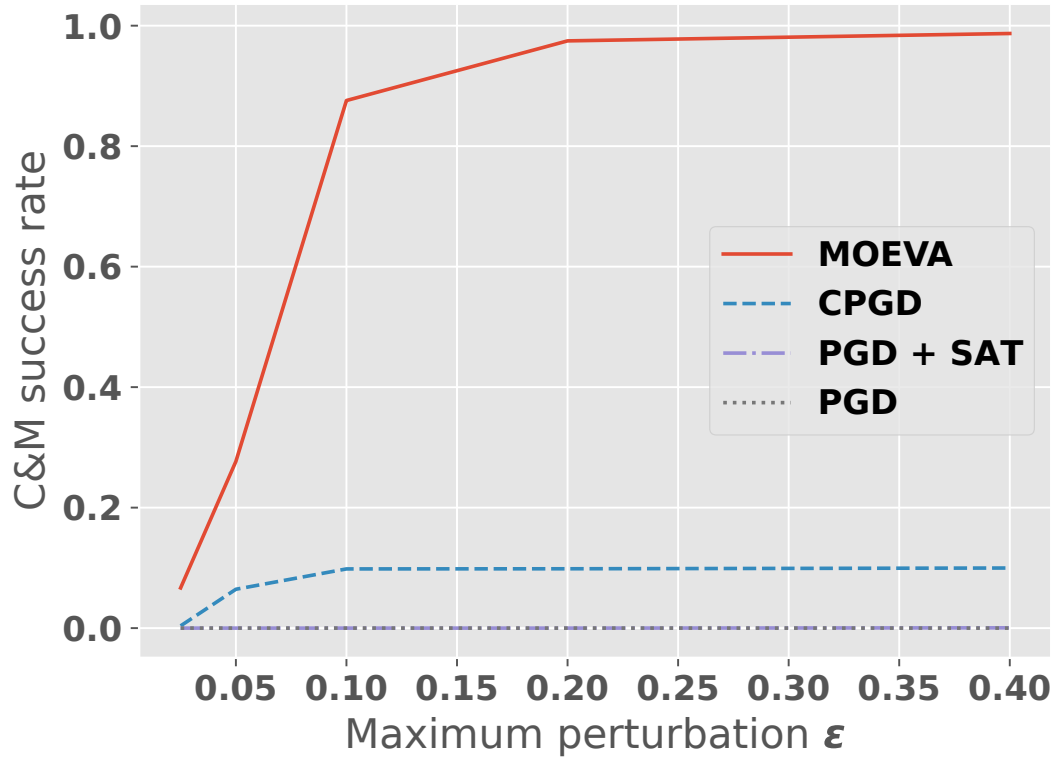


Figure 10.1: Success rate M&C of different attacks over ϵ budget on LCLD. The curves for PGD and PGD+SAT overlap on the value 0.

Table 10.1: Success rate (C&M) in % of MOEVA, for different numbers of generations.

	Number of Generations	C&M
LCLD	50	95.45
	100	97.48
	200	98.45
CTU-13	100	12.92
	500	99.48
	1000	100.00

Table 10.2: Success rate (C&M) in % of CPGD, for different numbers of iterations.

	Number of Generations	C&M
LCLD	500	8.88
	1000	9.85
	2000	10.75
CTU-13	500	0.00
	1000	0.00
	2000	0.00

number of generations, we chose to keep 100 and 1000 values as our default budget throughout the paper.

Table 10.2 shows that the effect of the number of iterations given to CPGD does not influence significantly the success rate of the attack. Moreover, we see that no matter the iteration budget, CPGD is incapable of generating a single adversarial example against the CTU-13 model. The same experiment (omitted from the table) reveals that the classic PGD and PGD coupled with a SAT solver fail to generate a single constrained adversarial example against both models no matter the number of iterations.

10.1.4 Combining constraints engineering and adversarial retraining to defend against search-based attacks.

Our previous results imply that both adversarial retraining using MOEVA and constraint augmentation improve the robustness of CML models. We argue that the two mechanisms are complementary and can be combined for improved robustness.

To evaluate the effectiveness of combining constraints engineering with adversarial retraining to defend our model, we compare the robustness of 4 defense scenarios against MOEVA attack: the first one is no defense and is equivalent to Section 7.1. The second approach uses adversarial retraining, the third approach uses constraint augmentation, and the last is adversarial retraining on a constraint-augmented model.

For a fair comparison, both the constraint-augmented and the original model are adversarially retrained with the same amount of inputs, even if the success rate of MOEVA on the constrain-augmented model is significantly lower, and thus generates fewer adversarial examples to train with. Using this protocol, we use 94K examples generated by MOEVA to retrain. That is about 3 times more than in Section 7.2. Therefore, we expect the success rate of the attack to be different from the success rate in Section 7.2 even though we use the same algorithm to generate the adversarial examples.

Table 10.3: Success rate (%) of MOEVA against different defense strategies according to 3 objectives, C for constraints satisfaction, M for misclassification, and C& M for both constraints satisfaction and misclassification for the same generated example.

Defense	C	M	C&M
None	100.00	99.90	97.48
Augment	100.00	93.33	80.43
MOEVA	100.00	89.00	82.00
MOEVA + Augment	100.00	89.23	77.23

Table 10.4: Success rate MOEVA after adversarial retraining and constraint augmentation.

Defense	LCLD	CTU-13	Malware	URL
None	41.51	5.41	39.30	31.89
Augment	19.73	6.63	28.5	20.99
MOEVA	3.90	4.67	37.69	22.14
MOEVA + Augment	00.77	4.67	28.98	15.94

Table 10.3 presents the results for the credit-scoring task. We show in Section 7.2, that constraint augmentation alone is sufficient to protect the model against botnet detection adversarials, a combination of the two defenses is therefore superfluous.

Starting from a constrained success rate of 97.48% on a defenseless model, the adversarial retraining lowers it to 82% while attacks against a constraint augmented model yield an 80.43% success rate. Combining adversarial retraining on top of a constraint augmentation defense leads to a success rate of 77.23%, improved from using only one or the other technique.

Conclusion: Combining constraint augmentation and adversarial retraining reduces the success rate of constrained adversarial attacks MOEVA by about 20% compared to unprotected models.

10.1.5 Evaluation of Random Forest Classifiers

Table 10.4 shows the success rate of MOEVA against 3 models (clean, adversarially retrained and constraints augmented). First, we observe that MOEVA successfully generates adversarial examples for all 4 datasets on the clean random forest, although we noticed that the success rate is significantly lower for the CTU-13 dataset.

10.2 Constrained Adaptive Attack: Effective Adversarial Attack Against Deep Neural Networks for Tabular Data

10.2.1 Experimental protocol

CAPGD Algorithm

Algorithm 2 CAPGD

Input: $\mathcal{L}$, h, S, Ω, $x^{(0)}$, y, η, N_{iter}, $W = \{w_0, \ldots, w_n\}$
Output: $x_{\max}$, $\mathcal{L}_{\max}$
$x^{(1)} \leftarrow P_{\mathcal{S}}\left(x^{(0)} + \eta \nabla \mathcal{L}'(x^{(0)})\right)$
$\mathcal{L}_{\max} \leftarrow \max\{\mathcal{L}'(x^{(0)}), \mathcal{L}'(x^{(1)})\}$
if $\mathcal{L}_{\max} \equiv \mathcal{L}'(x^{(0)})$ **then**
    $x_{\max} \leftarrow x^{(0)}$
else
    $x_{\max} \leftarrow x^{(1)}$
end if
for $k = 1$ to $N_{\text{iter}}-1$ **do**
    $z^{(k+1)} \leftarrow P_{\mathcal{S}}\left(x^{(k)} + \eta \nabla \mathcal{L}'(x^{(k)})\right)$
    $x^{(k+1)} \leftarrow P_{\mathcal{S}}\left(x^{(k)} + \alpha(z^{(k+1)} - x^{(k)}) + (1-\alpha)(x^{(k)} - x^{(k-1)})\right)$
    $x^{(k+1)} \leftarrow R_{\Omega}(x^{(k+1)})$
    if $\mathcal{L}'(x^{(k+1)}) > \mathcal{L}_{\max}$ **then**
        $x_{\max} \leftarrow x^{(k+1)}$
        $\mathcal{L}_{\max} \leftarrow \mathcal{L}'(x^{(k+1)})$
    end if
    if $k \in W$ **then**
        if Condition 1 **or** Condition 2 **then**
            $\eta \leftarrow \eta/2$
        end if
    end if
end for

CAA Algorithm

Algorithm 3 summarizes the process of CAA. The algorithm takes as input the clean examples X, their associated labels Y, the model H (including its weights, loss function and probability function), Ω the set of domain constraints, and ϵ the maximum perturbation. We start by creating the mask of examples that are already

Algorithm 3 CAA

Input: $X, Y, H, \Omega, \epsilon$
Output: X'
$adv\_mask \leftarrow is\_adv(X, X, Y, H, \Omega, \epsilon)$
$X' \leftarrow X[adv\_mask]$
$X_c \leftarrow X[\neg adv\_mask]$
$Y_c \leftarrow Y[\neg adv\_mask]$
for *Attack* **in** $\{CAPGD, MOEVA\}$ **do**
    $X_i \leftarrow Attack(X_c, Y_c, H, \Omega, \epsilon)$
    $adv\_mask \leftarrow is\_adv(X_i, X_c, Y_c, H, \Omega, \epsilon)$
    $X' \leftarrow X' \cup X_i[adv\_mask]$
    $X_c \leftarrow X_c[\neg adv\_mask]$
    $Y_c \leftarrow Y_c[\neg adv\_mask]$
end for
$X' \leftarrow X' \cup X_c$
procedure ISADV$(X_i, X_c, Y_c, H, \Omega, \epsilon)$
    $adv\_mask \leftarrow \{\}$
    for $k = 1$ **to** $|X|$ **do**
        $adv \leftarrow (X_i[k] \models \Omega) \wedge (H(X_i[k]) \neq Y_c[k]) \wedge (L_p(X_i[k], X_c[k]) \leq \epsilon)$
        $adv\_mask \leftarrow adv\_mask \cup adv$
    end for
    return adv_mask
end procedure

Table 10.5: The datasets evaluated in the empirical study, with the class imbalance of each dataset.

Dataset	Properties			
	Task	Size	# Features	Balance (%)
LCLD [Geo18]	Credit Scoring	1 220 092	28	80/20
URL [HY21]	Phishing URL detection	11 430	63	50/50
CTU-13 [CO22]	Botnet Detection	198 128	756	99.3/0.7
WIDS [LRG$^+$20]	ICU patient survival	91 713	186	91.4/8.6

adversarial (i.e. misclassified by the model) (l.3). We then split the clean examples that are already adversarial X' (l.4), and the candidates X_C on which we will execute the attacks. For each of our attacks (l.6), we generate a set of potentially adversarial examples from the candidate clean examples (l.7). Once again, we compute the mask of examples that are adversarial according to the subprocedure *is_adv* described below. According to the mask, we add the successful attack to the output X' (l.8) and remove the associated clean examples from the candidate set X_C. Hence, for a given example, the next attack is only executed if no attack has been successful, reducing the overall cost of CAA. At the end of CAA, we had the remaining candidates, for which we have not found adversarial examples to the output set of potentially adversarial examples X', to ease the calculation of robust accuracy (e.g. in transferable settings).

The subprocedure *is_adv* goes through all the examples $X_i[k] \in X_i$ and adds $True$ to the mask, if all of the following conditions hold:

- $X_i[k]$ respects the domain constraints,
- $X_i[k]$ classification by $H$is different from its true label $Y_c[k]$,
- $X_i[k]$ pertubation w.r.t to $X_c[k]$ is lower or equal to ϵ.

Datasets

Our dataset design followed the same protocol as in Chapter 4. We present in Table 10.5 the attributes of our datasets and the test performance achieved by each of the architectures.

Credit scoring - LCLD (licence: CC0: Public Domain) We engineer a dataset from the publicly available Lending Club Loan Data[1]. This dataset contains 151 features, and each example represents a loan that was accepted by the Lending Club. However, among these accepted loans, some are not repaid and charged off instead. Our goal is to predict, at request time, whether the borrower will be repaid or charged off. This dataset has been studied by multiple practitioners on Kaggle.

[1]https://www.kaggle.com/wordsforthewise/lending-club

However, the original version of the dataset contains only raw data and to the extent of our knowledge, there is no featured engineered version commonly used. In particular, one shall be careful when reusing feature-engineered versions, as most of the versions proposed present data leakage in the training set that makes the prediction trivial. Therefore, we propose our own feature engineering. The original dataset contains 151 features. We remove the example for which the feature "loan status" is different from "Fully paid" or "Charged Off" as these represent the only final status of a loan: for other values, the outcome is still uncertain. For our binary classifier, a 'Fully paid" loan is represented as 0 and a "Charged Off" as 1. We start by removing all features that are not set for more than 30% of the examples in the training set. We also remove all features that are not available at loan request time, as this would introduce bias. We impute the features that are redundant (e.g. grade and sub-grade) or too granular (e.g. address) to be useful for classification. Finally, we use one-hot encoding for categorical features. We obtain 47 input features and one target feature. We split the dataset using random sampling stratified on the target class and obtained a training set of 915K examples and a testing set of 305K. They are both unbalanced, with only 20% of charged-off loans (class 1). We trained a neural network to classify accepted and rejected loans. It has 3 fully connected hidden layers with 64, 32, and 16 neurons.

For each feature of this dataset, we define boundary constraints as the extremum value observed in the training set. We consider the 19 features that are under the control of the Lending Club as immutable. We identify 9 relationship constraints (3 linear, and 6 non-linear ones):

1. $\text{installment} = \text{loan\_amount} \times \frac{\text{rate} \times (1+\text{rate})^{\text{term}}}{(1+\text{rate})^{\text{term}}-1}$
2. $\text{open\_acc} \leq \text{total\_acc}$
3. $\text{pub\_rec\_bankruptcies} \leq \text{pub\_rec}$
4. $(\text{term} = 36) \vee (\text{term} = 60)$
5. $\text{ratio\_loan\_amnt\_annual\_inc} = \frac{\text{loan\_amnt}}{\text{annual\_inc}}$
6. $\text{ratio\_open\_acc\_total\_acc} = \frac{\text{open\_acc}}{\text{total\_acc}}$
7. $\text{ratio\_pub\_rec\_month\_since\_earliest\_cr\_line} = \frac{\text{pub\_rec}}{\text{month\_since\_earliest\_cr\_line}}$
8. $\text{ratio\_pub\_rec\_bankruptcies\_month\_since\_earliest\_cr\_line} = \frac{\text{pub\_rec\_bankruptcies}}{\text{month\_since\_earliest\_cr\_line}}$
9. $\text{ratio\_pub\_rec\_bankruptcies\_pub\_rec} = \frac{\text{pub\_rec\_bankruptcies}}{\text{pub\_rec}}, \text{if}, \text{pub\_rec} \neq 0, \text{else}, -1$

URL Phishing - ISCX-URL2016 (license CC BY 4.0) Phishing attacks are usually used to conduct cyber fraud or identity theft. This kind of attack takes the form of a URL that reassembles a legitimate URL (e.g. user's favorite e-commerce platform) but redirects to a fraudulent website that asks the user for their personal or banking data. [HY21] extracted features from legitimate and fraudulent URLs as well as external service-based features to build a classifier that can differentiate fraudulent URLs from legitimate ones. The feature extracted from the URL includes the number of special substrings such as "www", "&", ",", "$", "and", the length of the URL, the port, the appearance of a brand in the domain, in a subdomain or in the path, and the inclusion of "http" or "https". External service-based features include the Google index, the page rank, and the presence of the domain in the DNS records. [HY21] provide a dataset of 5715 legit and 5715 malicious URLs. We use 75% of the dataset for training and validation and the remaining 25% for testing and adversarial generation.

We extract a set of 14 relation constraints between the URL features. Among them, 7 are linear constraints (e.g. length of the hostname is less or equal to the length of the URL) and 7 are Boolean constraints of the type $if\, a > 0$ then $b > 0$ (e.g. if the number of http > 0 then the number slash "/" > 0).

Botnet attacks - CTU-13 (license CC BY NC SA 4.0) This is a feature-engineered version of CTU-13 proposed by [CO19b]. It includes a mix of legit and botnet traffic flows from the CTU University campus. Chernikova et al. aggregated the raw network data related to packets, duration, and bytes for each port from a list of commonly used ports. The dataset is made of 143K training examples and 55K testing examples, with 0.74% examples labeled in the botnet class (traffic that a botnet generates). Data have 756 features, including 432 mutable features. We identified two types of constraints that determine what feasible traffic data is. The first type concerns the number of connections and requires that an attacker cannot decrease it. The second type is inherent constraints in network communications (e.g. maximum packet size for TCP/UDP ports is 1500 bytes). In total, we identified 360 constraints.

WiDS (license: PhysioNet Restricted Health Data License 1.5.0 [2]) [LRG+20] dataset contains medical data on the survival of patients admitted to the ICU. The goal is to predict whether the patient will survive or die based on biological features (e.g. for triage). This very unbalanced dataset has 30 linear relation constraints.

Model architectures

Table 10.6 summarizes the family, model architecture, and hyperparameters tuned during the training of our models.

[2] https://physionet.org/content/widsdatathon2020/view-license/1.0.0/

Table 10.6: The three model architectures of our study.

Family	Model	Hyperparameters
Transformer	TabTransformer	*hidden_dim*, *n_layers*, *learning_rate*, *norm*, θ
Transformer	TabNet	*n_d*, *n_steps*, γ, *cat_emb_dim*, *n_independent*, *n_shared*, *momentum*, *mask_type*
Regularization	RLN	*hidden_dim*, *depth*, *heads*, *weight_decay*, *learning_rate*, *dropout*
Regularization	STG	*hidden_dims*, *learning_rate*, *lam*
Encoding	VIME	p_m, α, K, β

TabTransformer is a transformer-based model [HKC+20]. It uses self-attention to map the categorical features to an interpretable contextual embedding, and the paper claims this embedding improves the robustness of models to noisy inputs.

TabNet is another transformer-based model [AP21]. It uses multiple subnetworks that are used in sequence. At each decision step, it uses sequential attention to choose which features to reason. TabNet aggregates the outputs of each step to obtain the decision.

RLN or Regularization Learning Networks [SS18] uses an efficient hyperparameter tuning scheme in order to minimize a counterfactual loss. The authors train a regularization coefficient to weights in the neural network in order to lower the sensitivity and produce very sparse networks.

STG or Stochastic Gates [YLN+20] uses stochastic gates for feature selection in neural network estimation problems. The method is based on probabilistic relaxation of the l_0 norm of features or the count of the number of selected features.

VIME or Value Imputation for Mask Estimation [YZJ+20] uses self and then semi-supervised learning through deep encoders and predictors.

Attacks parameters

For existing attacks, we reuse the hyperparameters proposed in their respective papers. For LowProFool, we use a small step size of $\eta = 0.001$, $\lambda = 8.5$ rade off factor between fooling the classifier and generating imperceptible adversarial example, and run the attack for $N_{iter} = 20,000$ iterations. All other gradient attacks run for $N_{iter} = 10$ iterations. The schedule of decreasing steps of CPGD uses $M = 7$. In CAPGD, we fix the checkpoints as $w_j = \lceil p_j N_{\text{iter}} \rceil \leq N_{\text{iter}}$, with $p_j \in [0, 1]$ defined as $p_0 = 0$, $p_1 = 0.22$ and

$$p_{j+1} = p_j + \max\{p_j - p_{j-1} - 0.03, 0.06\}. \quad (10.1)$$

The influence of the previous update on the current update is set to $\alpha = 0.75$, and $\rho = 0.75$ for the halving of the step. MOEVA runs for $n_{gen} = 100$ iterations generating $n_{off} = 100$ offspring per iteration. Among the offspring, $n_{pop} = 200$ survive and are used for mating in the next iteration. With BF*, we discretize numerical features in $n_{bin} = 150$ bins. We run the attack for a maximum of $N_{iter} = 100$ iterations. CAA applies CAPGD and MOEVA with the same parameters.

Hardware

We run our experiments on an HPC cluster node with 32 cores and 64GB of RAM dedicated to our task. Each node consists of 2 AMD Epyc ROME 7H12 @ 2.6 GHz for a total of 128 cores with 256 GB of RAM.

Reproduction package and availability

The source code, datasets and pre-trained models to reproduce the experiments of this paper are available at `https://github.com/serval-uni-lu/tabularbench`.

10.2.2 Additional results

Budget of attacker

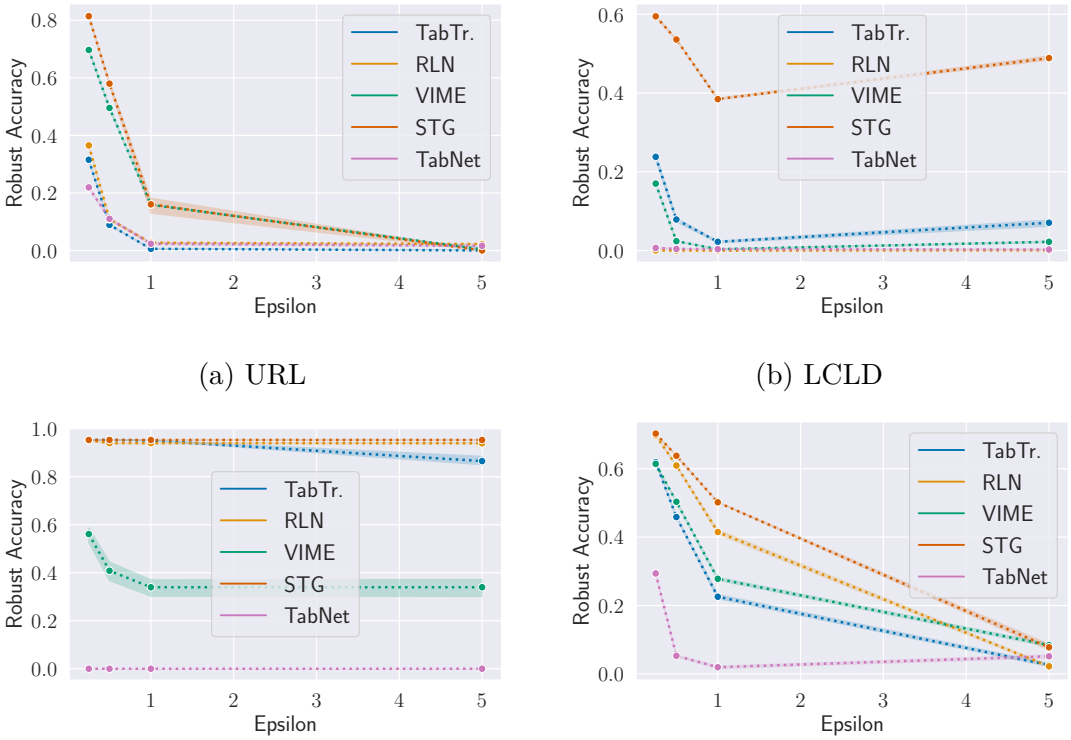
In this section, we study the impact of CAA's budget on its effectiveness. We consider 3 budgets: the maximum perturbation ϵ allowed, the number of iterations in the gradient attack CAPGD (without changing MOEVA's budget), and the number of iterations in the search attack MOEVA (without changing CAPGD's budget).

For each budget, we provide figures and detailed numerical results in tables, corresponding to the same experiment.

Maximum perturbation ϵ Figure 10.2 (numerical results in Table 10.7) reveals that increasing the maximum perturbation ϵ for CAA reduces the robust accuracy of the model in 16/20 cases.

Number of CAPGD iterations Figure 10.3 (numerical results in Table 10.8) reveals that increasing the number of iterations for the gradient attack component have a limited impact on the success rate of CAA. The maximum drop of accuracy is 3.5% points between 10 and 100 iterations for WIDS/TabNet.

Number of MOEVA iterations Figure 10.4 (numerical results in Table 10.9) reveals that increasing the number of iterations for the search attack component only reduces the robust accuracy in 4/20 cases (URL/VIME, URL/STG, CTU/VIME, and CTU/RLN).



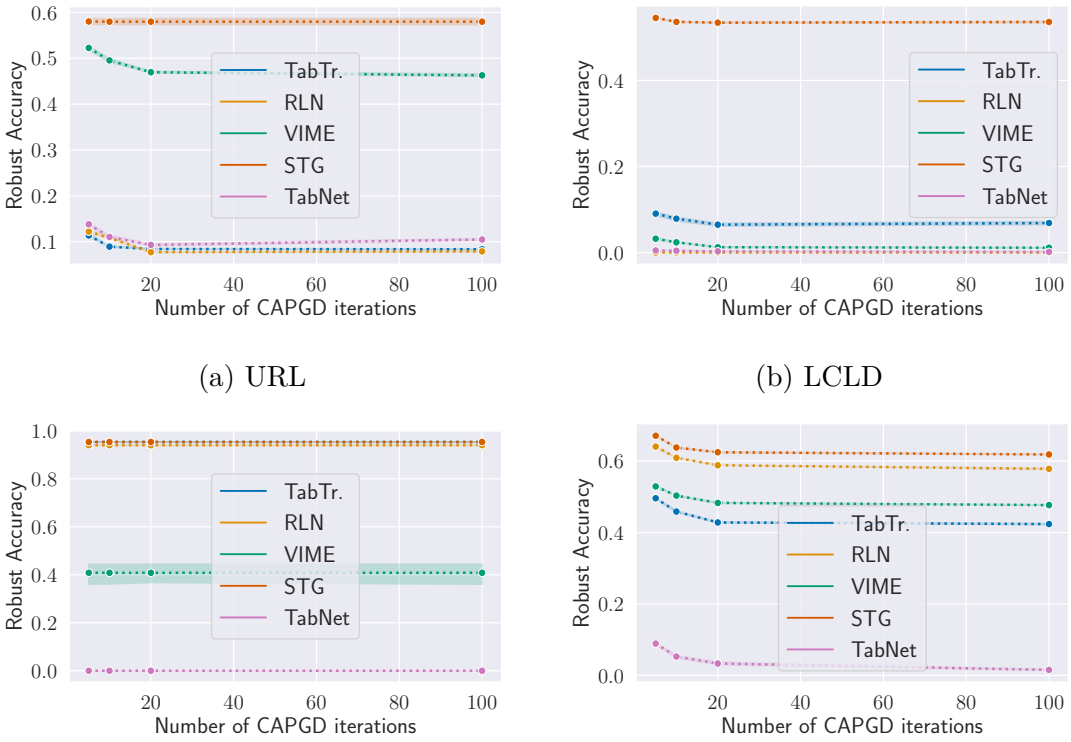
(a) URL (b) LCLD

(c) CTU (d) WIDS

Figure 10.2: Robust accuracy with CAA with varying maximum perturbation ϵ budget.

Table 10.7: Robust accuracy with CAA with varying maximum perturbation ϵ budget. The lowest robust accuracy is in bold.

		Maximum perturbation ϵ			
Dataset	Model	0.25	0.5	1.0	5.0
URL	TabTr.	$31.5_{\pm 0.1}$	$8.9_{\pm 0.2}$	$0.6_{\pm 0.1}$	$\mathbf{0.0}_{\pm \mathbf{0.0}}$
	RLN	$36.5_{\pm 0.4}$	$10.8_{\pm 0.2}$	$2.7_{\pm 0.2}$	$\mathbf{2.3}_{\pm \mathbf{0.0}}$
	VIME	$69.7_{\pm 0.2}$	$49.5_{\pm 0.5}$	$15.9_{\pm 0.1}$	$\mathbf{0.4}_{\pm \mathbf{0.3}}$
	STG	$81.4_{\pm 0.2}$	$58.0_{\pm 0.8}$	$16.1_{\pm 3.1}$	$\mathbf{0.0}_{\pm \mathbf{0.0}}$
	TabNet	$21.9_{\pm 0.7}$	$11.0_{\pm 0.5}$	$\mathbf{2.3}_{\pm \mathbf{0.4}}$	$\mathbf{1.6}_{\pm \mathbf{0.3}}$
LCLD	TabTr.	$23.8_{\pm 0.7}$	$7.9_{\pm 0.6}$	$\mathbf{2.2}_{\pm \mathbf{0.3}}$	$7.1_{\pm 0.9}$
	RLN	$\mathbf{0.0}_{\pm \mathbf{0.0}}$	$\mathbf{0.0}_{\pm \mathbf{0.0}}$	$\mathbf{0.0}_{\pm \mathbf{0.0}}$	$0.1_{\pm 0.0}$
	VIME	$17.0_{\pm 0.2}$	$2.4_{\pm 0.1}$	$\mathbf{0.3}_{\pm \mathbf{0.1}}$	$2.2_{\pm 0.2}$
	STG	$59.5_{\pm 0.3}$	$53.6_{\pm 0.1}$	$\mathbf{38.5}_{\pm \mathbf{0.2}}$	$48.9_{\pm 0.6}$
	TabNet	$0.7_{\pm 0.2}$	$\mathbf{0.4}_{\pm \mathbf{0.1}}$	$\mathbf{0.4}_{\pm \mathbf{0.1}}$	$\mathbf{0.3}_{\pm \mathbf{0.1}}$
CTU	TabTr.	$95.3_{\pm 0.0}$	$95.3_{\pm 0.0}$	$95.1_{\pm 0.2}$	$\mathbf{86.5}_{\pm \mathbf{1.9}}$
	RLN	$95.2_{\pm 0.3}$	$\mathbf{94.0}_{\pm \mathbf{0.2}}$	$\mathbf{94.0}_{\pm \mathbf{0.2}}$	$\mathbf{94.0}_{\pm \mathbf{0.2}}$
	VIME	$56.1_{\pm 3.4}$	$\mathbf{40.8}_{\pm \mathbf{4.7}}$	$\mathbf{34.0}_{\pm \mathbf{4.0}}$	$\mathbf{34.0}_{\pm \mathbf{4.0}}$
	STG	$\mathbf{95.3}_{\pm \mathbf{0.0}}$	$\mathbf{95.3}_{\pm \mathbf{0.0}}$	$\mathbf{95.3}_{\pm \mathbf{0.0}}$	$\mathbf{95.3}_{\pm \mathbf{0.0}}$
	TabNet	$\mathbf{0.0}_{\pm \mathbf{0.0}}$	$\mathbf{0.0}_{\pm \mathbf{0.0}}$	$\mathbf{0.0}_{\pm \mathbf{0.0}}$	$\mathbf{0.0}_{\pm \mathbf{0.0}}$
WIDS	TabTr.	$61.9_{\pm 0.3}$	$45.9_{\pm 0.3}$	$22.6_{\pm 0.7}$	$\mathbf{2.6}_{\pm \mathbf{0.5}}$
	RLN	$69.8_{\pm 0.2}$	$60.9_{\pm 0.2}$	$41.5_{\pm 0.8}$	$\mathbf{2.2}_{\pm \mathbf{0.4}}$
	VIME	$61.4_{\pm 0.1}$	$50.3_{\pm 0.2}$	$27.8_{\pm 0.5}$	$\mathbf{8.4}_{\pm \mathbf{0.5}}$
	STG	$70.3_{\pm 0.1}$	$63.8_{\pm 0.2}$	$50.2_{\pm 0.1}$	$\mathbf{7.8}_{\pm \mathbf{1.0}}$
	TabNet	$29.4_{\pm 0.2}$	$5.3_{\pm 0.4}$	$\mathbf{1.9}_{\pm \mathbf{0.4}}$	$5.2_{\pm 0.5}$



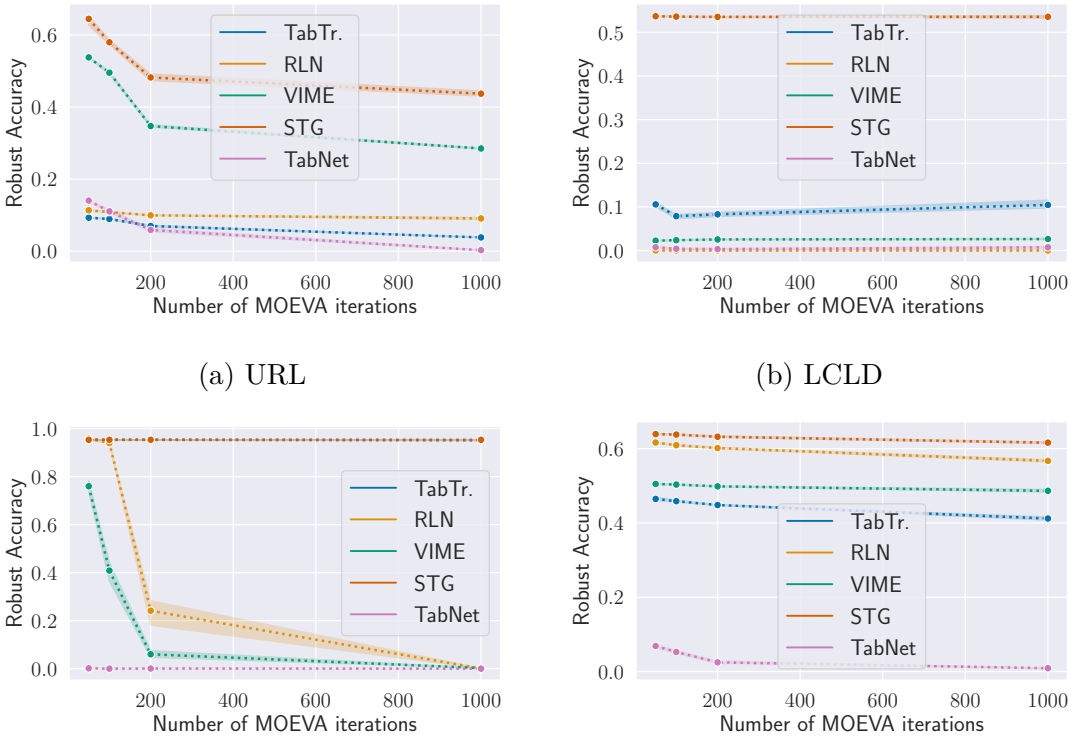
(a) URL (b) LCLD

(c) CTU (d) WIDS

Figure 10.3: Robust accuracy with CAA with varying gradient attack iterations in CAPGD.

Table 10.8: Robust accuracy with CAA with varying gradient attack iterations in CAPGD. The lowest robust accuracy is in bold.

Dataset	Model	# iterations CAPGD 5	10	20	100
URL	TabTr.	11.4±0.5	8.9±0.2	**8.4**±0.1	**8.3**±0.1
	RLN	12.2±0.4	10.8±0.2	**7.7**±0.1	**7.9**±0.2
	VIME	52.3±0.6	49.5±0.5	47.0±0.2	**46.3**±0.3
	STG	**58.0**±0.8	**58.0**±0.8	**58.0**±0.8	**58.0**±0.8
	TabNet	13.8±0.4	11.0±0.5	**9.3**±0.3	10.5±0.3
LCLD	TabTr.	9.0±0.5	7.9±0.6	**6.5**±0.4	**6.9**±0.5
	RLN	**0.0**±0.0	**0.0**±0.0	**0.0**±0.0	**0.0**±0.0
	VIME	3.2±0.3	2.4±0.1	**1.2**±0.1	**1.1**±0.0
	STG	54.5±0.1	**53.6**±0.1	**53.4**±0.2	**53.6**±0.2
	TabNet	0.5±0.2	0.4±0.1	**0.3**±0.1	**0.1**±0.1
CTU	TabTr.	**95.3**±0.0	**95.3**±0.0	**95.3**±0.0	**95.3**±0.0
	RLN	**94.0**±0.2	**94.0**±0.2	**94.0**±0.2	**94.0**±0.2
	VIME	**40.8**±4.7	**40.8**±4.7	**40.8**±4.7	**40.8**±4.7
	STG	**95.3**±0.0	**95.3**±0.0	**95.3**±0.0	**95.3**±0.0
	TabNet	**0.0**±0.0	**0.0**±0.0	**0.0**±0.0	**0.0**±0.0
WIDS	TabTr.	49.6±0.2	45.9±0.3	**42.8**±0.3	**42.4**±0.2
	RLN	64.1±0.2	60.9±0.2	58.8±0.0	**57.8**±0.2
	VIME	52.9±0.3	50.3±0.2	48.3±0.2	**47.7**±0.1
	STG	67.1±0.1	63.8±0.2	62.5±0.2	**61.8**±0.1
	TabNet	8.9±0.4	5.3±0.4	3.3±0.5	**1.6**±0.2



(a) URL

(b) LCLD

(c) CTU

(d) WIDS

Figure 10.4: Robust accuracy with CAA with varying search attack iterations in MOEVA.

Table 10.9: Robust accuracy with CAA with varying search attack iterations in MOEVA. The lowest robust accuracy is in bold.

Dataset	Model	# iterations MOEVA 50	100	200	1000
URL	TabTr.	9.3±0.0	8.9±0.2	7.0±0.2	**3.8**±0.1
	RLN	11.3±0.2	10.8±0.2	9.9±0.1	**9.1**±0.4
	VIME	53.8±0.2	49.5±0.5	34.7±0.4	**28.5**±0.2
	STG	64.5±1.7	58.0±0.8	48.2±1.0	**43.7**±0.8
	TabNet	14.0±0.2	11.0±0.5	5.9±0.6	**0.3**±0.1
LCLD	TabTr.	10.6±0.6	**7.9**±0.6	**8.3**±0.5	10.5±1.1
	RLN	**0.0**±0.0	**0.0**±0.0	**0.0**±0.0	**0.0**±0.0
	VIME	**2.3**±0.1	**2.4**±0.1	**2.5**±0.1	2.6±0.1
	STG	**53.7**±0.1	**53.6**±0.1	**53.5**±0.2	**53.6**±0.4
	TabNet	0.8±0.2	**0.4**±0.1	**0.3**±0.1	**0.7**±0.3
CTU	TabTr.	**95.3**±0.0	**95.3**±0.0	**95.3**±0.0	**95.1**±0.2
	RLN	95.8±0.4	94.0±0.2	24.1±6.1	**0.0**±0.1
	VIME	76.0±2.7	40.8±4.7	6.0±1.5	**0.2**±0.0
	STG	**95.3**±0.0	**95.3**±0.0	**95.3**±0.0	**95.3**±0.0
	TabNet	**0.1**±0.1	**0.0**±0.0	**0.0**±0.1	**0.0**±0.0
WIDS	TabTr.	46.5±0.5	45.9±0.3	44.8±0.2	**41.2**±0.5
	RLN	61.7±0.2	60.9±0.2	60.2±0.3	**56.7**±0.4
	VIME	50.5±0.2	50.3±0.2	49.9±0.2	**48.7**±0.4
	STG	63.9±0.2	63.8±0.2	63.2±0.3	**61.6**±0.3
	TabNet	6.9±0.4	5.3±0.4	2.5±0.4	**0.9**±0.1

Table 10.10: The datasets evaluated in the empirical study, with the class imbalance of each dataset.

Dataset	Properties			
	Task	Size	# Features	Balance (%)
LCLD [Geo18]	Credit Scoring	1 220 092	28	80/20
URL [HY21]	Phishing URL detection	11 430	63	50/50
CTU-13 [CO22]	Botnet Detection	198 128	756	99.3/0.7
WIDS [LRG+20]	ICU patient survival	91 713	186	91.4/8.6
Malware [AGM+20]	Malware detection	17 584	24 222	45.5/54.5

10.3 Defending Against Adversarial Attacks in Tabular Deep Learning

10.3.1 Experimental protocol

Datasets

Our dataset design followed the same protocol as in Chapter 4. We present in Table 10.10 the attributes of our datasets and the test performance achieved by each of the architectures.

Credit scoring - LCLD (licence: CC0: Public Domain) We engineer a dataset from the publicly available Lending Club Loan Data[3]. This dataset contains 151 features, and each example represents a loan that was accepted by the Lending Club. However, among these accepted loans, some are not repaid and charged off instead. Our goal is to predict, at request time, whether the borrower will be repaid or charged off. This dataset has been studied by multiple practitioners on Kaggle. However, the original version of the dataset contains only raw data and to the extent of our knowledge, there is no featured engineered version commonly used. In particular, one shall be careful when reusing feature-engineered versions, as most of the versions proposed present data leakage in the training set that makes the prediction trivial. Therefore, we propose our own feature engineering. The original dataset contains 151 features. We remove the example for which the feature "loan status" is different from "Fully paid" or "Charged Off" as these represent the only final status of a loan: for other values, the outcome is still uncertain. For our binary classifier, a 'Fully paid" loan is represented as 0 and a "Charged Off" as 1. We start by removing all features that are not set for more than 30% of the examples in the training set. We also remove all features that are not available at

[3]https://www.kaggle.com/wordsforthewise/lending-club

loan request time, as this would introduce bias. We impute the features that are redundant (e.g. grade and sub-grade) or too granular (e.g. address) to be useful for classification. Finally, we use one-hot encoding for categorical features. We obtain 47 input features and one target feature. We split the dataset using random sampling stratified on the target class and obtained a training set of 915K examples and a testing set of 305K. They are both unbalanced, with only 20% of charged-off loans (class 1). We trained a neural network to classify accepted and rejected loans. It has 3 fully connected hidden layers with 64, 32, and 16 neurons.

For each feature of this dataset, we define boundary constraints as the extremum value observed in the training set. We consider the 19 features that are under the control of the Lending Club as immutable. We identify 9 relationship constraints (3 linear, and 6 non-linear ones):

1. $\text{installment} = \text{loan\_amount} \times \frac{\text{rate} \times (1+\text{rate})^{\text{term}}}{(1+\text{rate})^{\text{term}}-1}$
2. $\text{open\_acc} \leq \text{total\_acc}$
3. $\text{pub\_rec\_bankruptcies} \leq \text{pub\_rec}$
4. $(\text{term} = 36) \vee (\text{term} = 60)$
5. $\text{ratio\_loan\_amnt\_annual\_inc} = \frac{\text{loan\_amnt}}{\text{annual\_inc}}$
6. $\text{ratio\_open\_acc\_total\_acc} = \frac{\text{open\_acc}}{\text{total\_acc}}$
7. $\text{ratio\_pub\_rec\_month\_since\_earliest\_cr\_line} = \frac{\text{pub\_rec}}{\text{month\_since\_earliest\_cr\_line}}$
8. $\text{ratio\_pub\_rec\_bankruptcies\_month\_since\_earliest\_cr\_line} = \frac{\text{pub\_rec\_bankruptcies}}{\text{month\_since\_earliest\_cr\_line}}$
9. $\text{ratio\_pub\_rec\_bankruptcies\_pub\_rec} = \frac{\text{pub\_rec\_bankruptcies}}{\text{pub\_rec}}, \text{if}, \text{pub\_rec} \neq 0, \text{else}, -1$

URL Phishing - ISCX-URL2016 (license CC BY 4.0) Phishing attacks are usually used to conduct cyber fraud or identity theft. This kind of attack takes the form of a URL that reassembles a legitimate URL (e.g. user's favorite e-commerce platform) but redirects to a fraudulent website that asks the user for their personal or banking data. [HY21] extracted features from legitimate and fraudulent URLs as well as external service-based features to build a classifier that can differentiate fraudulent URLs from legitimate ones. The feature extracted from the URL includes the number of special substrings such as "www", "&", ",", "$", "and", the length of the URL, the port, the appearance of a brand in the domain, in a subdomain or in the path, and the inclusion of "http" or "https". External service-based features include the Google index, the page rank, and the presence of the domain in the

DNS records. [HY21] provide a dataset of 5715 legit and 5715 malicious URLs. We use 75% of the dataset for training and validation and the remaining 25% for testing and adversarial generation.

We extract a set of 14 relation constraints between the URL features. Among them, 7 are linear constraints (e.g. length of the hostname is less or equal to the length of the URL) and 7 are Boolean constraints of the type $if a > 0$ then $b > 0$ (e.g. if the number of http > 0 then the number slash "/" > 0).

Botnet attacks - CTU-13 (license CC BY NC SA 4.0) This is a feature-engineered version of CTU-13 proposed by [CO19b]. It includes a mix of legit and botnet traffic flows from the CTU University campus. Chernikova et al. aggregated the raw network data related to packets, duration, and bytes for each port from a list of commonly used ports. The dataset is made of 143K training examples and 55K testing examples, with 0.74% examples labeled in the botnet class (traffic that a botnet generates). Data have 756 features, including 432 mutable features. We identified two types of constraints that determine what feasible traffic data is. The first type concerns the number of connections and requires that an attacker cannot decrease it. The second type is inherent constraints in network communications (e.g. maximum packet size for TCP/UDP ports is 1500 bytes). In total, we identified 360 constraints.

WiDS (license: PhysioNet Restricted Health Data License 1.5.0 [4]) [LRG+20] dataset contains medical data on the survival of patients admitted to the ICU. The goal is to predict whether the patient will survive or die based on biological features (e.g. for triage). This very unbalanced dataset has 30 linear relation constraints.

Malware (licence MIT) Malwares are a major threat to IT systems security. With the recent improvement of machine learning techniques, practitioners and researchers have developed ML-based detection systems to discriminate malicious software from benign software [UAB19]. Such systems are vulnerable to adversarial attacks as shown by [CSD19] with the AIMED attack: they successfully evade the classifier without reducing the malicious effect of the software. We use the dataset of benign and malicious portable executable provided in [AGM+20]. In the same paper, the authors showed that including packed and unpacked benign executables with malicious ones is less biased towards detecting the packing as a sign of maliciousness. Therefore, we select 4396 packed benign, 4396 unpacked benign, and 8792 malicious executables. As in [AGM+20], we extract a set of static features: PE headers, PE sections, DLL imports, API imports, Rich Header, File generic. In total, we obtain a dataset of 17 584 samples and 24 222 features. We use 85% of the dataset for training and validation and the remaining 15% for testing and adversarial generation.

[4] https://physionet.org/content/widsdatathon2020/view-license/1.0.0/

Table 10.11: The three model architectures of our study.

Family	Model	Hyperparameters
Transformer	TabTransformer	*hidden_dim*, *n_layers*, *learning_rate*, *norm*, θ
Transformer	TabNet	*n_d*, *n_steps*, γ, *cat_emb_dim*, *n_independent*, *n_shared*, *momentum*, *mask_type*
Regularization	RLN	*hidden_dim*, *depth*, *heads*, *weight_decay*, *learning_rate*, *dropout*
Regularization	STG	*hidden_dims*, *learning_rate*, *lam*
Encoding	VIME	p_m, α, K, β

From the 24 222 features, we identify 88 immutable features based on the PE format description from Microsoft. We also extract feature relation constraints from the original PE file examples we collected and those generated by AIMED. For example, the sum of binary features set to 1 that describe API imports should be less than the value of features api_nb, which represents the total number of imports on the PE file.

Model architectures

Table 10.11 provides an overview of the family, model architecture, and hyperparameters adjusted during the training of our models.

TabTransformer is a transformer-based model [HKC+20]. It employs self-attention to convert categorical features into an interpretable contextual embedding, which the paper asserts enhances the model's robustness to noisy inputs.

TabNet is another transformer-based model [AP21]. It utilizes multiple subnetworks in sequence. At each decision step, it applies sequential attention to select which features to consider. TabNet combines the outputs of each step to make the final decision.

RLN or Regularization Learning Networks [SS18] employs an efficient hyperparameter tuning method to minimize counterfactual loss. The authors train a regularization coefficient for the neural network weights to reduce sensitivity and create very sparse networks.

STG or Stochastic Gates [YLN+20] uses stochastic gates for feature selection in neural network estimation tasks. The technique is based on a probabilistic relaxation of the l_0 norm of features or the count of selected features.

VIME or Value Imputation for Mask Estimation [YZJ+20] employs self-supervised and semi-supervised learning through deep encoders and predictors.

Evaluation settings

Metrics The models are fine-tuned to maximize cross-validation AUC. This metric is threshold-independent and is not affected by the class unbalance of our dataset.

We only attack clean examples that are not already misclassified by the model and from the critical class, that is respectively for each aforementioned dataset the class of phishing URLs, rejected loans, malwares, botnets, and not surviving patients. Because we consider a single class, the only relevant metric is robust accuracy on constrained examples, which corresponds to the recall. Unsuccessful adversarial examples count as correctly classified when measuring robust accuracy.

We only consider examples that respect domain constraints to compute robust accuracy. If an attack generates invalid examples, they are defacto considered unsuccessful and are reverted to their original example (correctly classified).

We report in the Appendix 10.13 all the remaining performance metrics, including the recall, the precision, and the Mattheu Correlation Coefficient (MCC).

Attacks parameters CAA applies CAPGD and MOEVA with the following parameters.

CAPGD uses $N_{iter} = 10$ iterations. The step reduction schedule for CPGD uses $M = 7$. In CAPGD, checkpoints are set as $w_j = \lceil p_j \times N_{iter} \rceil \leq N_{iter}$, with $p_j \in [0, 1]$ defined as $p_0 = 0$, $p_1 = 0.22$, and

$$p_{j+1} = p_j + \max p_j - p_{j-1} - 0.03, 0.06.$$

The influence of the previous update on the current update is set to $\alpha = 0.75$, and $\rho = 0.75$ for step halving. MOEVA runs for $n_{gen} = 100$ iterations, generating $n_{off} = 100$ offspring per iteration. Among the offspring, $n_{pop} = 200$ survive and are used for mating in the subsequent iteration.

Hardware Our experiments are conducted on an HPC cluster node equipped with 32 cores and 64GB of RAM allocated for our use. Each node is composed of 2 AMD Epyc ROME 7H12 processors running at 2.6 GHz, providing a total of 128 cores and 256 GB of RAM.

Generator architectures

In our experimental study, we use the same five generative models as [SDC+24]: GOGGLE, TVAE, WGAN, TableGan, CTGAN.

The hyperameters for training these models is based on [SDC+24] as well:

For GOGGLE, we employed the same optimizer and learning rate configuration as described in [LQB+23]. Specifically, ADAM was used with five different learning rates: $\{1 \times 10^{-3}, 5 \times 10^{-3}, 1 \times 10^{-2}\}$.

For TVAE, ADAM was utilized with five different learning rates: $\{5 \times 10^{-6}, 1 \times 10^{-5}, 1 \times 10^{-4}, 2 \times 10^{-4}, 1 \times 10^{-3}\}$.

For the other DGM models, three different optimizers were tested: ADAM, RMSPROP, and SGD, each with distinct sets of learning rates.

For WGAN, the learning rates were $\{1 \times 10^{-4}, 1 \times 10^{-3}\}$, $\{5 \times 10^{-5}, 1 \times 10^{-4}, 1 \times 10^{-3}\}$, and $\{1 \times 10^{-4}, 1 \times 10^{-3}\}$, respectively.

For TableGAN, the learning rates were $\{5 \times 10^{-5}, 1 \times 10^{-4}, 2 \times 10^{-4}, 1 \times 10^{-3}\}$, $\{1 \times 10^{-4}, 2 \times 10^{-4}, 1 \times 10^{-3}\}$, and $\{1 \times 10^{-4}, 1 \times 10^{-3}\}$, respectively.

For CTGAN, the learning rates were $\{5 \times 10^{-5}, 1 \times 10^{-4}, 2 \times 10^{-4}\}$, $\{1 \times 10^{-4}, 2 \times 10^{-4}, 1 \times 10^{-3}\}$, and $\{1 \times 10^{-4}, 1 \times 10^{-3}\}$, respectively.

For each optimizer-learning rate combination, three different batch sizes were tested, depending on the DGM model: $\{64, 128, 256\}$ for WGAN, $\{128, 256, 512\}$ for TableGAN, $\{70, 280, 500\}$ for CTGAN and TVAE, and $\{64, 128\}$ for GOGGLE. The batch sizes for CTGAN are multiples of 10 to accommodate the recommended PAC value of 10 as suggested in [LKF+18], among other values.

Reproduction package and availability

The source code, datasets, and pre-trained models required to replicate the experiments in this paper are publicly accessible under the MIT license on the repository `https://github.com/serval-uni-lu/tabularbench`.

10.3.2 Detailed results

Baseline models performances

We compare in 10.12 the ID performance of XGBoost and our deep learning models under standard training. We confirm that DL models are on par with the performances achieved by shallow models.

Data augmentation detailed results

Clean performance after data augmentation We report in Table 10.13 the clean performances of our models under all the trianing scenarios. Notably, few training combinations lead to a collapse of performance ($MCC = 0$). It is the case on CTU dataset for all data augmentations with adversarial training, and CTGAN, Cutmix, and TVAE with standard training.

Table 10.13: Detailed results of clean performance for our augmented models

Dataset	Arch	Training	Augment	AUC	Accuracy	Precision	Recall	Mcc
URL	TabTr	Std	None	0.981	0.940	0.943	0.937	0.880
URL	TabTr	Adv	None	0.974	0.931	0.923	0.941	0.862
URL	TabTr	Std	ctgan	0.976	0.933	0.927	0.941	0.866
URL	TabTr	Adv	ctgan	0.963	0.916	0.903	0.932	0.832
URL	TabTr	Std	cutmix	0.968	0.930	0.954	0.905	0.862
URL	TabTr	Adv	cutmix	0.956	0.900	0.937	0.857	0.803
URL	TabTr	Std	goggle	0.974	0.931	0.932	0.930	0.862
URL	TabTr	Adv	goggle	0.964	0.915	0.913	0.918	0.830
URL	TabTr	Std	wgan	0.980	0.934	0.934	0.934	0.869
URL	TabTr	Adv	wgan	0.970	0.921	0.916	0.927	0.843
URL	TabTr	Std	tablegan	0.975	0.928	0.955	0.899	0.858
URL	TabTr	Adv	tablegan	0.967	0.919	0.935	0.900	0.839
URL	TabTr	Std	tvae	0.978	0.937	0.925	0.950	0.873
URL	TabTr	Adv	tvae	0.969	0.925	0.917	0.934	0.850
URL	STG	Std	None	0.973	0.920	0.908	0.934	0.839
URL	STG	Adv	None	0.949	0.862	0.812	0.943	0.734
URL	STG	Std	ctgan	0.967	0.910	0.898	0.925	0.820
URL	STG	Adv	ctgan	0.959	0.895	0.863	0.940	0.794
URL	STG	Std	cutmix	0.960	0.867	0.924	0.800	0.741
URL	STG	Adv	cutmix	0.954	0.842	0.909	0.760	0.694
URL	STG	Std	goggle	0.962	0.903	0.876	0.940	0.809
URL	STG	Adv	goggle	0.954	0.882	0.842	0.941	0.770
URL	STG	Std	wgan	0.970	0.913	0.903	0.926	0.826
URL	STG	Adv	wgan	0.963	0.896	0.862	0.943	0.796
URL	STG	Std	tablegan	0.968	0.908	0.933	0.878	0.817
URL	STG	Adv	tablegan	0.956	0.888	0.862	0.923	0.777
URL	STG	Std	tvae	0.969	0.913	0.892	0.940	0.827
URL	STG	Adv	tvae	0.961	0.889	0.843	0.956	0.786
URL	TabNet	Std	None	0.986	0.946	0.954	0.937	0.892
URL	TabNet	Adv	None	0.947	0.700	0.626	0.994	0.495
URL	TabNet	Std	ctgan	0.951	0.699	0.625	0.994	0.493
URL	TabNet	Adv	ctgan	0.943	0.853	0.819	0.905	0.709
URL	TabNet	Std	cutmix	0.947	0.860	0.802	0.958	0.735
URL	TabNet	Adv	cutmix	0.935	0.860	0.815	0.934	0.729
URL	TabNet	Std	goggle	0.934	0.851	0.803	0.932	0.712
URL	TabNet	Adv	goggle	0.939	0.868	0.880	0.852	0.736
URL	TabNet	Std	wgan	0.946	0.612	0.564	0.997	0.352
URL	TabNet	Adv	wgan	0.956	0.853	0.821	0.901	0.709
URL	TabNet	Std	tablegan	0.938	0.858	0.830	0.899	0.718
URL	TabNet	Adv	tablegan	0.929	0.504	1.000	0.008	0.063
URL	TabNet	Std	tvae	0.949	0.861	0.813	0.939	0.731
URL	TabNet	Adv	tvae	0.942	0.864	0.817	0.940	0.737
URL	RLN	Std	None	0.984	0.945	0.945	0.946	0.891
URL	RLN	Adv	None	0.977	0.933	0.917	0.953	0.867
URL	RLN	Std	ctgan	0.980	0.939	0.938	0.941	0.878
URL	RLN	Adv	ctgan	0.973	0.925	0.914	0.939	0.851
URL	RLN	Std	cutmix	0.983	0.944	0.945	0.942	0.887

URL	RLN	Adv	cutmix	0.977	0.933	0.924	0.944	0.866
URL	RLN	Std	goggle	0.978	0.938	0.937	0.940	0.877
URL	RLN	Adv	goggle	0.969	0.927	0.916	0.939	0.853
URL	RLN	Std	wgan	0.982	0.940	0.945	0.934	0.880
URL	RLN	Adv	wgan	0.976	0.927	0.923	0.933	0.855
URL	RLN	Std	tablegan	0.980	0.934	0.953	0.913	0.868
URL	RLN	Adv	tablegan	0.971	0.925	0.933	0.915	0.850
URL	RLN	Std	tvae	0.982	0.941	0.939	0.944	0.883
URL	RLN	Adv	tvae	0.976	0.927	0.916	0.941	0.855
URL	VIME	Std	None	0.974	0.928	0.929	0.927	0.856
URL	VIME	Adv	None	0.973	0.925	0.917	0.934	0.850
URL	VIME	Std	ctgan	0.968	0.916	0.906	0.927	0.831
URL	VIME	Adv	ctgan	0.965	0.912	0.913	0.911	0.824
URL	VIME	Std	cutmix	0.971	0.922	0.921	0.924	0.844
URL	VIME	Adv	cutmix	0.967	0.918	0.915	0.921	0.836
URL	VIME	Std	goggle	0.960	0.900	0.908	0.891	0.801
URL	VIME	Adv	goggle	0.955	0.904	0.892	0.920	0.809
URL	VIME	Std	wgan	0.968	0.917	0.913	0.923	0.835
URL	VIME	Adv	wgan	0.966	0.910	0.919	0.899	0.820
URL	VIME	Std	tablegan	0.963	0.905	0.930	0.875	0.811
URL	VIME	Adv	tablegan	0.960	0.906	0.923	0.887	0.813
URL	VIME	Std	tvae	0.968	0.914	0.919	0.907	0.828
URL	VIME	Adv	tvae	0.964	0.908	0.915	0.899	0.816
LCLD	TabTr	Std	None	0.717	0.633	0.314	0.699	0.254
LCLD	TabTr	Adv	None	0.711	0.590	0.293	0.738	0.233
LCLD	TabTr	Std	ctgan	0.711	0.614	0.304	0.715	0.244
LCLD	TabTr	Adv	ctgan	0.694	0.526	0.271	0.803	0.212
LCLD	TabTr	Std	cutmix	0.712	0.638	0.314	0.677	0.247
LCLD	TabTr	Adv	cutmix	0.702	0.596	0.294	0.723	0.230
LCLD	TabTr	Std	goggle	0.712	0.638	0.315	0.681	0.249
LCLD	TabTr	Adv	goggle	0.699	0.645	0.312	0.636	0.231
LCLD	TabTr	Std	wgan	0.711	0.634	0.313	0.684	0.247
LCLD	TabTr	Adv	wgan	0.688	0.615	0.296	0.664	0.214
LCLD	TabTr	Std	tablegan	0.710	0.636	0.313	0.678	0.245
LCLD	TabTr	Adv	tablegan	0.694	0.651	0.313	0.614	0.225
LCLD	TabTr	Std	tvae	0.716	0.634	0.314	0.693	0.252
LCLD	TabTr	Adv	tvae	0.702	0.620	0.304	0.691	0.235
LCLD	STG	Std	None	0.709	0.646	0.317	0.660	0.245
LCLD	STG	Adv	None	0.679	0.788	0.432	0.172	0.170
LCLD	STG	Std	ctgan	0.705	0.503	0.266	0.841	0.215
LCLD	STG	Adv	ctgan	0.700	0.505	0.266	0.833	0.212
LCLD	STG	Std	cutmix	0.707	0.766	0.404	0.347	0.231
LCLD	STG	Adv	cutmix	0.703	0.758	0.393	0.371	0.232
LCLD	STG	Std	goggle	0.704	0.677	0.331	0.591	0.242
LCLD	STG	Adv	goggle	0.698	0.616	0.300	0.687	0.229
LCLD	STG	Std	wgan	0.705	0.669	0.326	0.606	0.241
LCLD	STG	Adv	wgan	0.699	0.657	0.318	0.617	0.234
LCLD	STG	Std	tablegan	0.702	0.710	0.349	0.509	0.238
LCLD	STG	Adv	tablegan	0.699	0.657	0.318	0.621	0.235

LCLD	STG	Std	tvae	0.706	0.652	0.319	0.645	0.244
LCLD	STG	Adv	tvae	0.706	0.625	0.307	0.687	0.239
LCLD	TabNet	Std	None	0.722	0.656	0.326	0.668	0.262
LCLD	TabNet	Adv	None	0.656	0.799	0.000	0.000	0.000
LCLD	TabNet	Std	ctgan	0.687	0.785	0.270	0.042	0.031
LCLD	TabNet	Adv	ctgan	0.695	0.799	0.000	0.000	0.000
LCLD	TabNet	Std	cutmix	0.700	0.799	1.000	0.000	0.003
LCLD	TabNet	Adv	cutmix	0.638	0.799	0.000	0.000	0.000
LCLD	TabNet	Std	goggle	0.673	0.799	0.000	0.000	0.000
LCLD	TabNet	Adv	goggle	0.683	0.201	0.201	1.000	0.000
LCLD	TabNet	Std	wgan	0.665	0.799	0.000	0.000	0.000
LCLD	TabNet	Adv	wgan	0.688	0.799	0.000	0.000	0.000
LCLD	TabNet	Std	tablegan	0.689	0.793	0.255	0.016	0.015
LCLD	TabNet	Adv	tablegan	0.652	0.732	0.225	0.137	0.023
LCLD	TabNet	Std	tvae	0.667	0.799	0.248	0.000	0.002
LCLD	TabNet	Adv	tvae	0.696	0.799	0.000	0.000	0.000
LCLD	RLN	Std	None	0.719	0.641	0.318	0.685	0.255
LCLD	RLN	Adv	None	0.716	0.628	0.309	0.693	0.245
LCLD	RLN	Std	ctgan	0.709	0.620	0.306	0.703	0.242
LCLD	RLN	Adv	ctgan	0.704	0.582	0.290	0.749	0.232
LCLD	RLN	Std	cutmix	0.715	0.633	0.313	0.693	0.250
LCLD	RLN	Adv	cutmix	0.706	0.683	0.334	0.580	0.243
LCLD	RLN	Std	goggle	0.717	0.648	0.321	0.672	0.255
LCLD	RLN	Adv	goggle	0.710	0.644	0.317	0.666	0.247
LCLD	RLN	Std	wgan	0.712	0.644	0.317	0.668	0.248
LCLD	RLN	Adv	wgan	0.705	0.646	0.316	0.653	0.241
LCLD	RLN	Std	tablegan	0.712	0.642	0.316	0.672	0.249
LCLD	RLN	Adv	tablegan	0.704	0.629	0.308	0.679	0.239
LCLD	RLN	Std	tvae	0.717	0.633	0.314	0.697	0.253
LCLD	RLN	Adv	tvae	0.708	0.635	0.312	0.676	0.244
LCLD	VIME	Std	None	0.714	0.645	0.318	0.671	0.251
LCLD	VIME	Adv	None	0.713	0.651	0.321	0.657	0.250
LCLD	VIME	Std	ctgan	0.706	0.571	0.287	0.766	0.231
LCLD	VIME	Adv	ctgan	0.701	0.535	0.275	0.803	0.220
LCLD	VIME	Std	cutmix	0.710	0.710	0.353	0.528	0.249
LCLD	VIME	Adv	cutmix	0.701	0.682	0.332	0.575	0.239
LCLD	VIME	Std	goggle	0.714	0.666	0.328	0.633	0.253
LCLD	VIME	Adv	goggle	0.703	0.685	0.334	0.569	0.239
LCLD	VIME	Std	wgan	0.708	0.648	0.318	0.658	0.247
LCLD	VIME	Adv	wgan	0.699	0.660	0.320	0.618	0.237
LCLD	VIME	Std	tablegan	0.708	0.676	0.332	0.606	0.249
LCLD	VIME	Adv	tablegan	0.696	0.677	0.327	0.574	0.232
LCLD	VIME	Std	tvae	0.714	0.654	0.322	0.657	0.252
LCLD	VIME	Adv	tvae	0.705	0.628	0.308	0.684	0.240
CTU	TabTr	Std	None	0.979	1.000	0.982	0.953	0.967
CTU	TabTr	Adv	None	0.985	1.000	0.982	0.953	0.967
CTU	TabTr	Std	ctgan	0.630	0.044	0.008	1.000	0.017
CTU	TabTr	Adv	ctgan	0.627	0.045	0.008	1.000	0.017
CTU	TabTr	Std	cutmix	0.977	1.000	0.982	0.953	0.967

CTU	TabTr	Adv	cutmix	0.980	1.000	0.982	0.953	0.967
CTU	TabTr	Std	wgan	0.982	1.000	0.982	0.953	0.967
CTU	TabTr	Adv	wgan	0.984	1.000	0.982	0.953	0.967
CTU	TabTr	Std	tablegan	0.978	1.000	0.987	0.951	0.969
CTU	TabTr	Adv	tablegan	0.979	1.000	0.987	0.953	0.970
CTU	TabTr	Std	tvae	0.977	0.943	0.111	0.963	0.317
CTU	TabTr	Adv	tvae	0.974	0.609	0.018	0.983	0.103
CTU	STG	Std	None	0.988	1.000	0.982	0.953	0.967
CTU	STG	Adv	None	0.986	1.000	0.992	0.951	0.971
CTU	STG	Std	ctgan	0.990	0.999	0.890	0.956	0.922
CTU	STG	Adv	ctgan	0.986	0.930	0.092	0.961	0.286
CTU	STG	Std	cutmix	0.986	1.000	0.982	0.953	0.967
CTU	STG	Adv	cutmix	0.985	1.000	1.000	0.946	0.972
CTU	STG	Std	wgan	0.986	1.000	0.982	0.953	0.967
CTU	STG	Adv	wgan	0.985	1.000	0.982	0.953	0.967
CTU	STG	Std	tablegan	0.986	1.000	0.982	0.953	0.967
CTU	STG	Adv	tablegan	0.984	1.000	1.000	0.951	0.975
CTU	STG	Std	tvae	0.984	0.890	0.061	0.963	0.227
CTU	STG	Adv	tvae	0.981	0.436	0.013	0.983	0.072
CTU	TabNet	Std	None	0.996	0.999	0.958	0.961	0.959
CTU	TabNet	Adv	None	0.978	0.993	0.500	0.002	0.035
CTU	TabNet	Std	ctgan	0.986	0.993	0.000	0.000	0.000
CTU	TabNet	Adv	ctgan	0.977	0.016	0.007	1.000	0.008
CTU	TabNet	Std	cutmix	0.982	0.993	0.000	0.000	0.000
CTU	TabNet	Adv	cutmix	0.982	0.993	0.000	0.000	0.000
CTU	TabNet	Std	wgan	0.983	1.000	0.985	0.951	0.967
CTU	TabNet	Adv	wgan	0.987	0.993	0.000	0.000	0.000
CTU	TabNet	Std	tablegan	0.980	1.000	0.982	0.953	0.967
CTU	TabNet	Adv	tablegan	0.993	0.993	1.000	0.015	0.121
CTU	TabNet	Std	tvae	0.987	0.993	0.000	0.000	0.000
CTU	TabNet	Adv	tvae	0.976	0.007	0.007	1.000	0.000
CTU	RLN	Std	None	0.991	0.998	0.819	0.978	0.894
CTU	RLN	Adv	None	0.990	0.999	0.904	0.973	0.937
CTU	RLN	Std	ctgan	0.994	0.986	0.338	0.975	0.570
CTU	RLN	Adv	ctgan	0.992	0.985	0.327	0.975	0.561
CTU	RLN	Std	cutmix	0.989	1.000	0.987	0.953	0.970
CTU	RLN	Adv	cutmix	0.987	1.000	1.000	0.953	0.976
CTU	RLN	Std	wgan	0.991	0.999	0.887	0.966	0.925
CTU	RLN	Adv	wgan	0.990	0.999	0.923	0.975	0.949
CTU	RLN	Std	tablegan	0.992	0.999	0.880	0.975	0.926
CTU	RLN	Adv	tablegan	0.990	0.999	0.896	0.975	0.934
CTU	RLN	Std	tvae	0.988	0.987	0.362	0.973	0.589
CTU	RLN	Adv	tvae	0.988	0.986	0.338	0.975	0.570
CTU	VIME	Std	None	0.987	1.000	0.997	0.951	0.974
CTU	VIME	Adv	None	0.983	1.000	0.997	0.951	0.974
CTU	VIME	Std	ctgan	0.972	0.007	0.007	1.000	0.000
CTU	VIME	Adv	ctgan	0.741	0.007	0.007	1.000	0.000
CTU	VIME	Std	cutmix	0.991	1.000	0.997	0.951	0.974
CTU	VIME	Adv	cutmix	0.976	1.000	0.997	0.951	0.974

CTU	VIME	Std	wgan	0.977	1.000	1.000	0.953	0.976
CTU	VIME	Adv	wgan	0.979	1.000	0.997	0.953	0.975
CTU	VIME	Std	tablegan	0.984	1.000	0.997	0.951	0.974
CTU	VIME	Adv	tablegan	0.979	1.000	0.997	0.951	0.974
CTU	VIME	Std	tvae	0.950	0.008	0.007	1.000	0.001
CTU	VIME	Adv	tvae	0.727	0.007	0.007	1.000	0.000
WIDS	TabTr	Std	None	0.874	0.810	0.287	0.755	0.383
WIDS	TabTr	Adv	None	0.869	0.794	0.272	0.772	0.373
WIDS	TabTr	Std	ctgan	0.868	0.799	0.279	0.780	0.383
WIDS	TabTr	Adv	ctgan	0.859	0.769	0.249	0.782	0.349
WIDS	TabTr	Std	cutmix	0.866	0.835	0.314	0.708	0.395
WIDS	TabTr	Adv	cutmix	0.851	0.867	0.358	0.601	0.395
WIDS	TabTr	Std	goggle	0.873	0.805	0.285	0.784	0.392
WIDS	TabTr	Adv	goggle	0.853	0.784	0.261	0.764	0.357
WIDS	TabTr	Std	wgan	0.866	0.797	0.273	0.763	0.371
WIDS	TabTr	Adv	wgan	0.864	0.788	0.264	0.764	0.361
WIDS	TabTr	Std	tablegan	0.869	0.808	0.284	0.748	0.378
WIDS	TabTr	Adv	tablegan	0.858	0.806	0.277	0.724	0.363
WIDS	TabTr	Std	tvae	0.871	0.801	0.280	0.776	0.383
WIDS	TabTr	Adv	tvae	0.858	0.790	0.264	0.747	0.356
WIDS	STG	Std	None	0.866	0.782	0.260	0.776	0.361
WIDS	STG	Adv	None	0.865	0.875	0.381	0.627	0.424
WIDS	STG	Std	ctgan	0.852	0.638	0.183	0.878	0.285
WIDS	STG	Adv	ctgan	0.841	0.668	0.193	0.851	0.293
WIDS	STG	Std	cutmix	0.863	0.885	0.400	0.567	0.414
WIDS	STG	Adv	cutmix	0.851	0.880	0.380	0.530	0.385
WIDS	STG	Std	goggle	0.851	0.780	0.253	0.742	0.342
WIDS	STG	Adv	goggle	0.837	0.727	0.218	0.787	0.310
WIDS	STG	Std	wgan	0.863	0.800	0.274	0.744	0.366
WIDS	STG	Adv	wgan	0.855	0.855	0.334	0.625	0.384
WIDS	STG	Std	tablegan	0.861	0.846	0.326	0.676	0.396
WIDS	STG	Adv	tablegan	0.853	0.829	0.302	0.688	0.376
WIDS	STG	Std	tvae	0.857	0.776	0.252	0.758	0.345
WIDS	STG	Adv	tvae	0.845	0.807	0.271	0.678	0.341
WIDS	TabNet	Std	None	0.870	0.777	0.259	0.796	0.365
WIDS	TabNet	Adv	None	0.835	0.104	0.090	0.984	0.003
WIDS	TabNet	Std	ctgan	0.853	0.090	0.090	1.000	0.000
WIDS	TabNet	Adv	ctgan	0.863	0.090	0.090	1.000	0.000
WIDS	TabNet	Std	cutmix	0.866	0.910	0.000	0.000	0.000
WIDS	TabNet	Adv	cutmix	0.859	0.090	0.090	1.000	0.000
WIDS	TabNet	Std	goggle	0.856	0.090	0.090	1.000	0.000
WIDS	TabNet	Adv	goggle	0.862	0.090	0.090	1.000	0.000
WIDS	TabNet	Std	wgan	0.865	0.795	0.275	0.787	0.381
WIDS	TabNet	Adv	wgan	0.855	0.090	0.090	1.000	0.000
WIDS	TabNet	Std	tablegan	0.864	0.090	0.090	1.000	0.000
WIDS	TabNet	Adv	tablegan	0.860	0.090	0.090	1.000	0.000
WIDS	TabNet	Std	tvae	0.857	0.104	0.090	0.984	0.003
WIDS	TabNet	Adv	tvae	0.864	0.090	0.090	1.000	0.000
WIDS	RLN	Std	None	0.869	0.796	0.274	0.774	0.376

WIDS	RLN	Adv	None	0.867	0.789	0.268	0.779	0.370
WIDS	RLN	Std	ctgan	0.862	0.788	0.264	0.761	0.360
WIDS	RLN	Adv	ctgan	0.425	0.090	0.090	1.000	0.000
WIDS	RLN	Std	cutmix	0.870	0.802	0.280	0.769	0.381
WIDS	RLN	Adv	cutmix	0.859	0.834	0.307	0.681	0.379
WIDS	RLN	Std	goggle	0.864	0.797	0.276	0.774	0.378
WIDS	RLN	Adv	goggle	0.857	0.777	0.256	0.782	0.358
WIDS	RLN	Std	wgan	0.866	0.782	0.260	0.774	0.359
WIDS	RLN	Adv	wgan	0.858	0.770	0.249	0.776	0.347
WIDS	RLN	Std	tablegan	0.868	0.773	0.254	0.785	0.356
WIDS	RLN	Adv	tablegan	0.860	0.797	0.273	0.760	0.370
WIDS	RLN	Std	tvae	0.868	0.776	0.259	0.803	0.367
WIDS	RLN	Adv	tvae	0.854	0.756	0.237	0.774	0.332
WIDS	VIME	Std	None	0.865	0.823	0.298	0.721	0.384
WIDS	VIME	Adv	None	0.858	0.817	0.291	0.720	0.376
WIDS	VIME	Std	ctgan	0.482	0.090	0.090	1.000	0.000
WIDS	VIME	Adv	ctgan	0.482	0.090	0.090	1.000	0.000
WIDS	VIME	Std	cutmix	0.857	0.833	0.309	0.697	0.387
WIDS	VIME	Adv	cutmix	0.849	0.878	0.374	0.543	0.385
WIDS	VIME	Std	goggle	0.849	0.812	0.280	0.700	0.358
WIDS	VIME	Adv	goggle	0.840	0.802	0.268	0.700	0.346
WIDS	VIME	Std	wgan	0.861	0.796	0.270	0.753	0.365
WIDS	VIME	Adv	wgan	0.845	0.791	0.259	0.715	0.339
WIDS	VIME	Std	tablegan	0.864	0.828	0.305	0.716	0.389
WIDS	VIME	Adv	tablegan	0.853	0.882	0.388	0.553	0.399
WIDS	VIME	Std	tvae	0.858	0.808	0.280	0.726	0.367
WIDS	VIME	Adv	tva	0.846	0.787	0.256	0.721	0.339

For LCLD dataset only Goggle and WGAN data augmentations lead to $MCC = 0$. To uncover what happens with some generated data, we study the distribution of artificial examples on the LCLD dataset for 3 cases: Two cases where performance did not collapse: TableGAN and CTGAN and one problematic case WGAN.

Kernel Density Estimation. We first compare the artificial examples distributions in Figure 10.5. The results show that the labels and the main features of TableGAN, a "healthy" generator are closer to the distribution of the "problematic" generator WGAN than to the distribution of CTGAN, another "healthy" generator. Feature and label distributions are not problematic.

Statistical analysis. We perform the following statistical tests to compare the distributions quantitatively between the examples generated by the three generators. Kolmogorov-Smirnov test, t-test, or MWU test. We report the results in Table 10.14. Across all statistical tests, there is no specific pattern to the faulty generator "WGAN" compared to CTGAN and TableGAN.

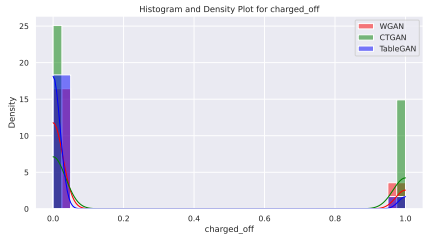
Classification performance. We build a new classifier to identify examples generated by WGAN and by TableGAN. We leverage Oodeel[5], a library that performs post-hoc deep OOD (Out-of-Distribution) detection.

The classifier reaches achieves a random accuracy (0.5) confirming that no specific features are sufficient to distinguish both generators.

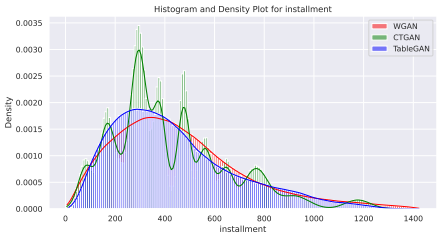
[5]https://github.com/deel-ai/oodeel

Table 10.12: AUC In-distribution performance of models

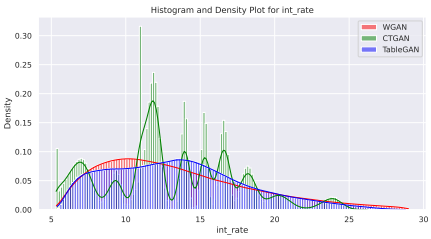
Dataset	CTU	LCLD	MALWARE	URL	WIDS
RLN	0.991	0.719	0.993	0.984	0.869
STG	0.988	0.709	0.991	0.973	0.866
TabNet	0.996	0.722	0.994	0.986	0.870
TabTr	0.979	0.717	0.994	0.981	0.874
VIME	0.987	0.714	0.989	0.974	0.865
XGBoost	0.994	0.723	0.997	0.993	0.887



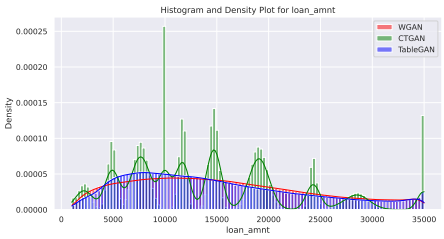
(a) Label: Charged off



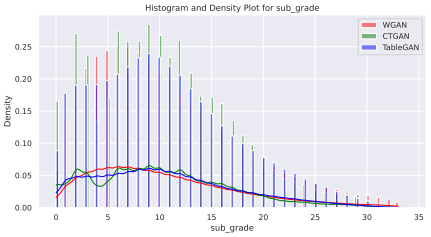
(b) Feature: Installment



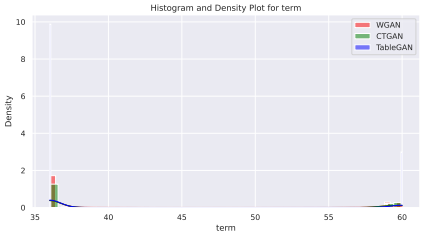
(c) Feature: Interest rate



(d) Feature: Loan amount



(e) Feature: Sub-grade



(f) Feature: Loan term

Figure 10.5: Impact of attack budget on the robust accuracy for LCLD dataset.

Table 10.14: Statistical tests between the distributions of the 3 generators: W:WGAN, T:TableGAN, C:CTGAN, MWU:Mann-Whitney U

GAN	Test	Amount	Term	Rate	Installment	Sub-grade	Label
(W/T)	KS Statistic	0.047	0.120	0.055	0.046	0.031	0.095
(W/T)	KS p-value	0.000	0.000	0.000	0.000	0.000	0.000
(W/T)	t-test Statistic	35.923	10.782	-7.687	40.512	0.224	140.654
(W/T)	t-test p-value	0.000	0.000	0.000	0.000	0.823	0.000
(W/T)	MWU Statistic	1.3×10^{11}	1.2×10^{11}	1.2×10^{11}	1.3×10^{11}	1.2×10^{11}	1.3×10^{11}
(W/T)	MWU p-value	0.000	0.000	0.000	0.000	0.000	0.000
(W/C)	KS Statistic	0.112	0.056	0.105	0.089	0.037	0.194
(W/C)	KS p-value	0.000	0.000	0.000	0.000	0.000	0.000
(W/C)	t-test Statistic	80.112	-21.286	40.896	61.097	30.043	-221.351
(W/C)	t-test p-value	0.000	0.000	0.000	0.000	0.000	0.000
(W/C)	MWU Statistic	1.3×10^{11}	1.2×10^{11}	1.2×10^{11}	1.3×10^{11}	1.2×10^{11}	9.8×10^{10}
(W/C)	MWU p-value	0.000	0.002	0.000	0.000	0.000	0.000
(T/C)	KS Statistic	0.079	0.070	0.093	0.044	0.027	0.289
(T/C)	KS p-value	0.000	0.000	0.000	0.000	0.000	0.000
(T/C)	t-test Statistic	-43.986	31.467	-51.028	-20.991	-30.376	364.250
(T/C)	t-test p-value	0.000	0.000	0.000	0.000	0.000	0.000
(T/C)	MWU Statistic	1.2×10^{11}	1.2×10^{11}	1.2×10^{11}	1.2×10^{11}	1.2×10^{11}	1.6×10^{11}
(T/C)	MWU p-value	0.000	0.000	0.000	0.000	0.000	0.000

Next, we evaluate the Maximum Logit Score (MLS) detector and report the histograms and AUROC curve of the detector in Figure 10.6.

Both the ROC curves and the histograms confirm that WGAN and TableGAN are not distinguishable.

Conclusion: From all our analysis, we confirm that the collapse of performance of training with WGAN data augmentation is not due to some evident properties in the generated examples.

Robust performance after data augmentation We report below the robustness of our 270 models trained with various combinations of arhcitecture, data augmentation, and adversarial training.

Table 10.15: Detailed results of Adv robustness with constrained (CTR) and unconstrained attacks (ADV) across our 5 seeds.

Dataset	Arch	Training	Augment	ID_{mean}	CTR_{mean}	ADV_{mean}	ID_{std}	CTR_{std}	ADV_{std}
CTU	STG	Adv	None	0.951	0.951	0.951	0.000	0.000	0.000
CTU	STG	Adv	ctgan	0.961	0.960	0.959	0.000	0.001	0.002
CTU	STG	Adv	cutmix	0.946	0.945	0.946	0.000	0.001	0.000
CTU	STG	Adv	tablegan	0.951	0.951	0.951	0.000	0.000	0.000
CTU	STG	Adv	tvae	0.983	0.983	0.982	0.000	0.000	0.001
CTU	STG	Adv	wgan	0.953	0.953	0.953	0.000	0.000	0.000
CTU	STG	Std	None	0.953	0.953	0.953	0.000	0.000	0.000
CTU	STG	Std	ctgan	0.956	0.953	0.956	0.000	0.000	0.000

CTU	STG	Std	cutmix	0.953	0.953	0.953	0.000	0.000	0.000
CTU	STG	Std	tablegan	0.953	0.953	0.953	0.000	0.000	0.000
CTU	STG	Std	tvae	0.963	0.961	0.963	0.000	0.000	0.000
CTU	STG	Std	wgan	0.953	0.953	0.953	0.000	0.000	0.000
CTU	TabNet	Adv	None	0.002	0.002	0.002	0.000	0.001	0.001
CTU	TabNet	Adv	ctgan	1.000	1.000	1.000	0.000	0.000	0.000
CTU	TabNet	Adv	cutmix	0.000	0.000	0.000	0.000	0.000	0.000
CTU	TabNet	Adv	tablegan	0.015	0.014	0.014	0.000	0.001	0.001
CTU	TabNet	Adv	tvae	1.000	1.000	1.000	0.000	0.000	0.000
CTU	TabNet	Adv	wgan	0.000	0.000	0.000	0.000	0.000	0.000
CTU	TabNet	Std	None	0.961	0.000	0.961	0.000	0.000	0.000
CTU	TabNet	Std	ctgan	0.000	0.000	0.000	0.000	0.000	0.000
CTU	TabNet	Std	cutmix	0.000	0.000	0.000	0.000	0.000	0.000
CTU	TabNet	Std	tablegan	0.953	0.953	0.953	0.000	0.000	0.000
CTU	TabNet	Std	tvae	0.000	0.000	0.000	0.000	0.000	0.000
CTU	TabNet	Std	wgan	0.951	0.951	0.951	0.000	0.000	0.000
CTU	TabTr	Adv	None	0.953	0.953	0.953	0.000	0.000	0.000
CTU	TabTr	Adv	ctgan	1.000	0.944	1.000	0.000	0.010	0.000
CTU	TabTr	Adv	cutmix	0.953	0.953	0.953	0.000	0.000	0.000
CTU	TabTr	Adv	tablegan	0.953	0.953	0.953	0.000	0.001	0.000
CTU	TabTr	Adv	tvae	0.983	0.983	0.983	0.000	0.000	0.000
CTU	TabTr	Adv	wgan	0.953	0.953	0.953	0.000	0.000	0.000
CTU	TabTr	Std	None	0.953	0.953	0.953	0.000	0.000	0.000
CTU	TabTr	Std	ctgan	1.000	0.944	1.000	0.000	0.005	0.000
CTU	TabTr	Std	cutmix	0.953	0.949	0.953	0.000	0.003	0.000
CTU	TabTr	Std	tablegan	0.951	0.939	0.951	0.000	0.001	0.000
CTU	TabTr	Std	tvae	0.963	0.961	0.963	0.000	0.000	0.000
CTU	TabTr	Std	wgan	0.953	0.953	0.953	0.000	0.000	0.000
CTU	RLN	Adv	None	0.973	0.971	0.973	0.000	0.000	0.000
CTU	RLN	Adv	ctgan	0.975	0.967	0.975	0.000	0.001	0.000
CTU	RLN	Adv	cutmix	0.953	0.953	0.953	0.000	0.000	0.000
CTU	RLN	Adv	tablegan	0.975	0.975	0.975	0.000	0.001	0.000
CTU	RLN	Adv	tvae	0.975	0.968	0.975	0.000	0.002	0.000
CTU	RLN	Adv	wgan	0.975	0.974	0.975	0.000	0.001	0.000
CTU	RLN	Std	None	0.978	0.940	0.978	0.000	0.003	0.000
CTU	RLN	Std	ctgan	0.975	0.956	0.975	0.000	0.002	0.000
CTU	RLN	Std	cutmix	0.953	0.953	0.953	0.000	0.000	0.000
CTU	RLN	Std	tablegan	0.975	0.814	0.975	0.000	0.026	0.000
CTU	RLN	Std	tvae	0.973	0.932	0.973	0.000	0.011	0.000
CTU	RLN	Std	wgan	0.966	0.950	0.966	0.000	0.001	0.000
CTU	VIME	Adv	None	0.951	0.940	0.942	0.000	0.005	0.006
CTU	VIME	Adv	ctgan	1.000	1.000	1.000	0.000	0.000	0.000
CTU	VIME	Adv	cutmix	0.951	0.943	0.947	0.000	0.004	0.002
CTU	VIME	Adv	tablegan	0.951	0.855	0.894	0.000	0.016	0.008
CTU	VIME	Adv	tvae	1.000	1.000	1.000	0.000	0.000	0.000
CTU	VIME	Adv	wgan	0.953	0.952	0.953	0.000	0.001	0.000
CTU	VIME	Std	None	0.951	0.408	0.951	0.000	0.049	0.000
CTU	VIME	Std	ctgan	1.000	1.000	1.000	0.000	0.000	0.000
CTU	VIME	Std	cutmix	0.951	0.350	0.951	0.000	0.029	0.000

CTU	VIME	Std	tablegan	0.951	0.670	0.951	0.000	0.021	0.000
CTU	VIME	Std	tvae	1.000	1.000	1.000	0.000	0.000	0.000
CTU	VIME	Std	wgan	0.953	0.229	0.953	0.000	0.022	0.000
LCLD	STG	Adv	None	0.156	0.121	0.156	0.000	0.001	0.000
LCLD	STG	Adv	ctgan	0.820	0.812	0.820	0.000	0.001	0.000
LCLD	STG	Adv	cutmix	0.376	0.362	0.376	0.000	0.000	0.000
LCLD	STG	Adv	goggle	0.694	0.682	0.694	0.000	0.000	0.000
LCLD	STG	Adv	tablegan	0.627	0.601	0.627	0.000	0.001	0.000
LCLD	STG	Adv	tvae	0.689	0.678	0.689	0.000	0.000	0.000
LCLD	STG	Adv	wgan	0.613	0.597	0.613	0.000	0.000	0.000
LCLD	STG	Std	None	0.664	0.536	0.664	0.000	0.001	0.000
LCLD	STG	Std	ctgan	0.833	0.595	0.833	0.000	0.004	0.000
LCLD	STG	Std	cutmix	0.352	0.222	0.352	0.000	0.002	0.000
LCLD	STG	Std	goggle	0.577	0.433	0.577	0.000	0.002	0.000
LCLD	STG	Std	tablegan	0.510	0.442	0.510	0.000	0.001	0.000
LCLD	STG	Std	tvae	0.649	0.505	0.649	0.000	0.001	0.000
LCLD	STG	Std	wgan	0.614	0.377	0.614	0.000	0.002	0.000
LCLD	TabNet	Adv	None	0.000	0.000	0.001	0.000	0.000	0.000
LCLD	TabNet	Adv	ctgan	0.000	0.000	0.001	0.000	0.000	0.000
LCLD	TabNet	Adv	cutmix	0.000	0.000	0.001	0.000	0.000	0.000
LCLD	TabNet	Adv	goggle	1.000	1.000	1.000	0.000	0.000	0.000
LCLD	TabNet	Adv	tablegan	0.116	0.114	0.117	0.000	0.000	0.000
LCLD	TabNet	Adv	tvae	0.000	0.000	0.001	0.000	0.000	0.000
LCLD	TabNet	Adv	wgan	0.000	0.000	0.001	0.000	0.000	0.000
LCLD	TabNet	Std	None	0.674	0.004	0.674	0.000	0.001	0.000
LCLD	TabNet	Std	ctgan	0.029	0.021	0.030	0.000	0.001	0.000
LCLD	TabNet	Std	cutmix	0.000	0.000	0.001	0.000	0.000	0.000
LCLD	TabNet	Std	goggle	0.000	0.000	0.001	0.000	0.000	0.000
LCLD	TabNet	Std	tablegan	0.013	0.010	0.014	0.000	0.001	0.000
LCLD	TabNet	Std	tvae	0.000	0.000	0.001	0.000	0.000	0.000
LCLD	TabNet	Std	wgan	0.000	0.000	0.001	0.000	0.000	0.000
LCLD	TabTr	Adv	None	0.739	0.703	0.739	0.000	0.001	0.000
LCLD	TabTr	Adv	ctgan	0.795	0.785	0.795	0.000	0.001	0.000
LCLD	TabTr	Adv	cutmix	0.725	0.710	0.725	0.000	0.001	0.000
LCLD	TabTr	Adv	goggle	0.636	0.605	0.636	0.000	0.002	0.000
LCLD	TabTr	Adv	tablegan	0.608	0.564	0.608	0.000	0.003	0.000
LCLD	TabTr	Adv	tvae	0.687	0.665	0.687	0.000	0.001	0.000
LCLD	TabTr	Adv	wgan	0.665	0.628	0.665	0.000	0.002	0.000
LCLD	TabTr	Std	None	0.695	0.079	0.695	0.000	0.006	0.000
LCLD	TabTr	Std	ctgan	0.724	0.081	0.724	0.000	0.004	0.000
LCLD	TabTr	Std	cutmix	0.677	0.073	0.677	0.000	0.008	0.000
LCLD	TabTr	Std	goggle	0.689	0.079	0.689	0.000	0.004	0.000
LCLD	TabTr	Std	tablegan	0.693	0.101	0.693	0.000	0.005	0.000
LCLD	TabTr	Std	tvae	0.703	0.048	0.703	0.000	0.003	0.000
LCLD	TabTr	Std	wgan	0.701	0.055	0.701	0.000	0.005	0.000
LCLD	RLN	Adv	None	0.695	0.630	0.695	0.000	0.001	0.000
LCLD	RLN	Adv	ctgan	0.737	0.543	0.737	0.000	0.001	0.000
LCLD	RLN	Adv	cutmix	0.581	0.470	0.581	0.000	0.003	0.000
LCLD	RLN	Adv	goggle	0.678	0.320	0.678	0.000	0.005	0.000

LCLD	RLN	Adv	tablegan	0.688	0.479	0.688	0.000	0.004	0.000
LCLD	RLN	Adv	tvae	0.670	0.643	0.670	0.000	0.000	0.000
LCLD	RLN	Adv	wgan	0.661	0.402	0.661	0.000	0.004	0.000
LCLD	RLN	Std	None	0.683	0.000	0.683	0.000	0.000	0.000
LCLD	RLN	Std	ctgan	0.705	0.001	0.705	0.000	0.001	0.000
LCLD	RLN	Std	cutmix	0.689	0.000	0.689	0.000	0.000	0.000
LCLD	RLN	Std	goggle	0.673	0.000	0.673	0.000	0.000	0.000
LCLD	RLN	Std	tablegan	0.693	0.001	0.693	0.000	0.001	0.000
LCLD	RLN	Std	tvae	0.700	0.000	0.700	0.000	0.000	0.000
LCLD	RLN	Std	wgan	0.679	0.005	0.679	0.000	0.002	0.000
LCLD	VIME	Adv	None	0.655	0.104	0.655	0.000	0.002	0.000
LCLD	VIME	Adv	ctgan	0.789	0.768	0.789	0.000	0.000	0.000
LCLD	VIME	Adv	cutmix	0.570	0.529	0.570	0.000	0.001	0.000
LCLD	VIME	Adv	goggle	0.568	0.532	0.568	0.000	0.002	0.000
LCLD	VIME	Adv	tablegan	0.563	0.537	0.563	0.000	0.000	0.000
LCLD	VIME	Adv	tvae	0.678	0.661	0.678	0.000	0.001	0.000
LCLD	VIME	Adv	wgan	0.617	0.530	0.617	0.000	0.002	0.000
LCLD	VIME	Std	None	0.670	0.024	0.670	0.000	0.001	0.000
LCLD	VIME	Std	ctgan	0.773	0.018	0.773	0.000	0.002	0.000
LCLD	VIME	Std	cutmix	0.523	0.020	0.523	0.000	0.001	0.000
LCLD	VIME	Std	goggle	0.644	0.005	0.644	0.000	0.001	0.000
LCLD	VIME	Std	tablegan	0.607	0.005	0.607	0.000	0.001	0.000
LCLD	VIME	Std	tvae	0.668	0.007	0.668	0.000	0.001	0.000
LCLD	VIME	Std	wgan	0.659	0.007	0.659	0.000	0.002	0.000
URL	STG	Adv	None	0.943	0.900	0.903	0.000	0.001	0.001
URL	STG	Adv	ctgan	0.939	0.798	0.803	0.000	0.012	0.014
URL	STG	Adv	cutmix	0.755	0.427	0.422	0.000	0.032	0.032
URL	STG	Adv	goggle	0.939	0.856	0.860	0.000	0.010	0.008
URL	STG	Adv	tablegan	0.921	0.809	0.816	0.000	0.004	0.003
URL	STG	Adv	tvae	0.957	0.795	0.804	0.000	0.017	0.015
URL	STG	Adv	wgan	0.942	0.812	0.813	0.000	0.003	0.003
URL	STG	Std	None	0.933	0.580	0.596	0.000	0.008	0.007
URL	STG	Std	ctgan	0.922	0.693	0.770	0.000	0.008	0.006
URL	STG	Std	cutmix	0.794	0.397	0.444	0.000	0.009	0.010
URL	STG	Std	goggle	0.939	0.745	0.759	0.000	0.005	0.006
URL	STG	Std	tablegan	0.876	0.469	0.575	0.000	0.005	0.008
URL	STG	Std	tvae	0.941	0.688	0.733	0.000	0.002	0.006
URL	STG	Std	wgan	0.925	0.655	0.752	0.000	0.007	0.006
URL	TabNet	Adv	None	0.995	0.918	0.919	0.000	0.002	0.001
URL	TabNet	Adv	ctgan	0.901	0.899	0.899	0.000	0.000	0.000
URL	TabNet	Adv	cutmix	0.930	0.897	0.896	0.000	0.001	0.001
URL	TabNet	Adv	goggle	0.848	0.665	0.666	0.000	0.022	0.019
URL	TabNet	Adv	tablegan	0.008	0.000	0.000	0.000	0.000	0.000
URL	TabNet	Adv	tvae	0.940	0.872	0.870	0.000	0.018	0.018
URL	TabNet	Adv	wgan	0.898	0.896	0.896	0.000	0.000	0.000
URL	TabNet	Std	None	0.934	0.110	0.299	0.000	0.005	0.004
URL	TabNet	Std	ctgan	0.994	0.948	0.948	0.000	0.002	0.001
URL	TabNet	Std	cutmix	0.954	0.893	0.894	0.000	0.001	0.001
URL	TabNet	Std	goggle	0.932	0.896	0.896	0.000	0.001	0.000

URL	TabNet	Std	tablegan	0.896	0.878	0.875	0.000	0.010	0.011
URL	TabNet	Std	tvae	0.938	0.891	0.892	0.000	0.002	0.003
URL	TabNet	Std	wgan	0.998	0.952	0.953	0.000	0.002	0.001
URL	TabTr	Adv	None	0.939	0.567	0.578	0.000	0.009	0.009
URL	TabTr	Adv	ctgan	0.930	0.660	0.664	0.000	0.004	0.004
URL	TabTr	Adv	cutmix	0.850	0.403	0.404	0.000	0.011	0.012
URL	TabTr	Adv	goggle	0.917	0.541	0.554	0.000	0.006	0.007
URL	TabTr	Adv	tablegan	0.898	0.409	0.421	0.000	0.010	0.011
URL	TabTr	Adv	tvae	0.934	0.612	0.615	0.000	0.008	0.003
URL	TabTr	Adv	wgan	0.927	0.569	0.580	0.000	0.008	0.010
URL	TabTr	Std	None	0.936	0.089	0.825	0.000	0.002	0.001
URL	TabTr	Std	ctgan	0.942	0.253	0.880	0.000	0.006	0.005
URL	TabTr	Std	cutmix	0.904	0.018	0.687	0.000	0.000	0.000
URL	TabTr	Std	goggle	0.930	0.049	0.051	0.000	0.001	0.001
URL	TabTr	Std	tablegan	0.899	0.020	0.020	0.000	0.000	0.000
URL	TabTr	Std	tvae	0.952	0.168	0.901	0.000	0.002	0.002
URL	TabTr	Std	wgan	0.936	0.200	0.887	0.000	0.006	0.002
URL	RLN	Adv	None	0.952	0.562	0.566	0.000	0.007	0.006
URL	RLN	Adv	ctgan	0.938	0.625	0.628	0.000	0.005	0.007
URL	RLN	Adv	cutmix	0.943	0.608	0.609	0.000	0.003	0.007
URL	RLN	Adv	goggle	0.939	0.661	0.665	0.000	0.008	0.006
URL	RLN	Adv	tablegan	0.913	0.555	0.557	0.000	0.009	0.005
URL	RLN	Adv	tvae	0.941	0.598	0.602	0.000	0.003	0.003
URL	RLN	Adv	wgan	0.933	0.547	0.552	0.000	0.002	0.005
URL	RLN	Std	None	0.944	0.108	0.901	0.000	0.002	0.001
URL	RLN	Std	ctgan	0.942	0.219	0.855	0.000	0.005	0.001
URL	RLN	Std	cutmix	0.941	0.086	0.926	0.000	0.002	0.002
URL	RLN	Std	goggle	0.936	0.039	0.039	0.000	0.000	0.000
URL	RLN	Std	tablegan	0.910	0.039	0.039	0.000	0.000	0.000
URL	RLN	Std	tvae	0.942	0.081	0.912	0.000	0.002	0.002
URL	RLN	Std	wgan	0.935	0.214	0.911	0.000	0.002	0.002
URL	VIME	Adv	None	0.934	0.698	0.727	0.000	0.006	0.004
URL	VIME	Adv	ctgan	0.910	0.669	0.690	0.000	0.005	0.007
URL	VIME	Adv	cutmix	0.920	0.686	0.707	0.000	0.010	0.012
URL	VIME	Adv	goggle	0.919	0.737	0.749	0.000	0.013	0.011
URL	VIME	Adv	tablegan	0.887	0.645	0.652	0.000	0.005	0.004
URL	VIME	Adv	tvae	0.899	0.636	0.711	0.000	0.004	0.004
URL	VIME	Adv	wgan	0.897	0.650	0.705	0.000	0.004	0.004
URL	VIME	Std	None	0.925	0.495	0.533	0.000	0.005	0.003
URL	VIME	Std	ctgan	0.927	0.548	0.910	0.000	0.004	0.001
URL	VIME	Std	cutmix	0.925	0.467	0.913	0.000	0.004	0.001
URL	VIME	Std	goggle	0.893	0.445	0.857	0.000	0.003	0.001
URL	VIME	Std	tablegan	0.875	0.430	0.750	0.000	0.005	0.003
URL	VIME	Std	tvae	0.909	0.444	0.886	0.000	0.005	0.003
URL	VIME	Std	wgan	0.922	0.519	0.905	0.000	0.008	0.003
WIDS	STG	Adv	None	0.626	0.452	0.626	0.000	0.002	0.000
WIDS	STG	Adv	ctgan	0.853	0.738	0.853	0.000	0.002	0.000
WIDS	STG	Adv	cutmix	0.532	0.412	0.523	0.000	0.001	0.003
WIDS	STG	Adv	goggle	0.788	0.660	0.788	0.000	0.002	0.000

WIDS	STG	Adv	tablegan	0.689	0.566	0.688	0.000	0.003	0.001
WIDS	STG	Adv	tvae	0.677	0.598	0.677	0.000	0.001	0.001
WIDS	STG	Adv	wgan	0.626	0.464	0.623	0.000	0.002	0.001
WIDS	STG	Std	None	0.776	0.638	0.773	0.000	0.002	0.000
WIDS	STG	Std	ctgan	0.878	0.712	0.877	0.000	0.003	0.000
WIDS	STG	Std	cutmix	0.567	0.385	0.559	0.000	0.004	0.000
WIDS	STG	Std	goggle	0.742	0.572	0.739	0.000	0.003	0.000
WIDS	STG	Std	tablegan	0.677	0.498	0.671	0.000	0.004	0.000
WIDS	STG	Std	tvae	0.759	0.621	0.755	0.000	0.003	0.000
WIDS	STG	Std	wgan	0.746	0.583	0.744	0.000	0.002	0.000
WIDS	TabNet	Adv	None	0.984	0.584	0.825	0.000	0.002	0.000
WIDS	TabNet	Adv	ctgan	1.000	1.000	1.000	0.000	0.000	0.000
WIDS	TabNet	Adv	cutmix	1.000	0.374	0.671	0.000	0.003	0.007
WIDS	TabNet	Adv	goggle	1.000	1.000	1.000	0.000	0.000	0.000
WIDS	TabNet	Adv	tablegan	1.000	1.000	1.000	0.000	0.000	0.000
WIDS	TabNet	Adv	tvae	1.000	1.000	1.000	0.000	0.000	0.000
WIDS	TabNet	Adv	wgan	1.000	0.992	0.996	0.000	0.004	0.002
WIDS	TabNet	Std	None	0.797	0.053	0.731	0.000	0.004	0.002
WIDS	TabNet	Std	ctgan	1.000	1.000	1.000	0.000	0.000	0.000
WIDS	TabNet	Std	cutmix	0.000	0.000	0.000	0.000	0.000	0.000
WIDS	TabNet	Std	goggle	1.000	1.000	1.000	0.000	0.000	0.000
WIDS	TabNet	Std	tablegan	1.000	1.000	1.000	0.000	0.000	0.000
WIDS	TabNet	Std	tvae	0.984	0.406	0.475	0.000	0.001	0.000
WIDS	TabNet	Std	wgan	0.786	0.000	0.456	0.000	0.000	0.003
WIDS	TabTr	Adv	None	0.773	0.651	0.767	0.000	0.002	0.000
WIDS	TabTr	Adv	ctgan	0.781	0.681	0.776	0.000	0.003	0.001
WIDS	TabTr	Adv	cutmix	0.600	0.508	0.599	0.000	0.003	0.001
WIDS	TabTr	Adv	goggle	0.765	0.675	0.755	0.000	0.002	0.001
WIDS	TabTr	Adv	tablegan	0.726	0.622	0.724	0.000	0.002	0.001
WIDS	TabTr	Adv	tvae	0.747	0.667	0.743	0.000	0.002	0.001
WIDS	TabTr	Adv	wgan	0.765	0.652	0.759	0.000	0.002	0.002
WIDS	TabTr	Std	None	0.755	0.459	0.746	0.000	0.003	0.000
WIDS	TabTr	Std	ctgan	0.780	0.441	0.776	0.000	0.005	0.000
WIDS	TabTr	Std	cutmix	0.710	0.434	0.705	0.000	0.003	0.000
WIDS	TabTr	Std	goggle	0.786	0.383	0.733	0.000	0.004	0.000
WIDS	TabTr	Std	tablegan	0.750	0.376	0.750	0.000	0.008	0.000
WIDS	TabTr	Std	tvae	0.776	0.493	0.763	0.000	0.003	0.001
WIDS	TabTr	Std	wgan	0.763	0.376	0.763	0.000	0.005	0.000
WIDS	RLN	Adv	None	0.780	0.666	0.773	0.000	0.002	0.000
WIDS	RLN	Adv	ctgan	1.000	1.000	1.000	0.000	0.000	0.000
WIDS	RLN	Adv	cutmix	0.681	0.599	0.675	0.000	0.002	0.001
WIDS	RLN	Adv	goggle	0.783	0.691	0.774	0.000	0.003	0.001
WIDS	RLN	Adv	tablegan	0.760	0.661	0.754	0.000	0.002	0.001
WIDS	RLN	Adv	tvae	0.775	0.711	0.772	0.000	0.003	0.002
WIDS	RLN	Adv	wgan	0.776	0.676	0.776	0.000	0.003	0.001
WIDS	RLN	Std	None	0.775	0.609	0.771	0.000	0.002	0.000
WIDS	RLN	Std	ctgan	0.762	0.472	0.759	0.000	0.007	0.000
WIDS	RLN	Std	cutmix	0.770	0.587	0.767	0.000	0.002	0.000
WIDS	RLN	Std	goggle	0.773	0.525	0.750	0.000	0.001	0.000

WIDS	RLN	Std	tablegan	0.788	0.589	0.786	0.000	0.004	0.000
WIDS	RLN	Std	tvae	0.802	0.621	0.796	0.000	0.004	0.000
WIDS	RLN	Std	wgan	0.775	0.574	0.775	0.000	0.002	0.000
WIDS	VIME	Adv	None	0.721	0.521	0.721	0.000	0.003	0.000
WIDS	VIME	Adv	ctgan	1.000	1.000	1.000	0.000	0.000	0.000
WIDS	VIME	Adv	cutmix	0.543	0.435	0.535	0.000	0.002	0.001
WIDS	VIME	Adv	goggle	0.702	0.592	0.699	0.000	0.002	0.001
WIDS	VIME	Adv	tablegan	0.553	0.423	0.553	0.000	0.002	0.000
WIDS	VIME	Adv	tvae	0.721	0.618	0.721	0.000	0.001	0.000
WIDS	VIME	Adv	wgan	0.715	0.606	0.715	0.000	0.002	0.000
WIDS	VIME	Std	None	0.723	0.503	0.713	0.000	0.002	0.000
WIDS	VIME	Std	ctgan	1.000	1.000	1.000	0.000	0.000	0.000
WIDS	VIME	Std	cutmix	0.699	0.476	0.694	0.000	0.002	0.000
WIDS	VIME	Std	goggle	0.702	0.491	0.697	0.000	0.003	0.000
WIDS	VIME	Std	tablegan	0.718	0.501	0.718	0.000	0.004	0.000
WIDS	VIME	Std	tvae	0.726	0.506	0.726	0.000	0.004	0.000
WIDS	VIME	Std	wgan	0.755	0.512	0.754	0.000	0.001	0.000

Correlations between ID and robust performances

Impact of budgets, detailed results

Generalization to other distances

We define for all attacks a distance function. This method is used for MOEVA (the evolution attack) to measure the fitness value related to the distance objective, and in the evaluation method to validate the correctness of the adversarial examples.

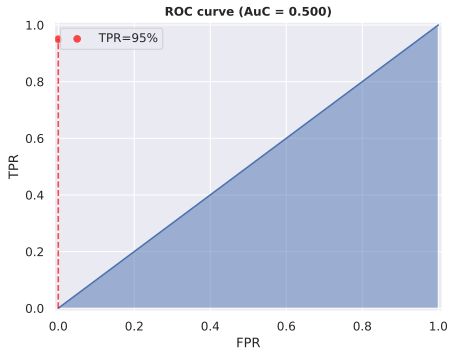
By default, it supports L_∞ and L_2 distances [6]:

```python
from tabularbench.utils.typing import NDBool, NDInt, NDNumber

def compute_distance(x_1: NDNumber, x_2: NDNumber, norm: Any) -> NDNumber:
    if norm in ["inf", np.inf, "Linf", "linf"]:
        distance = np.linalg.norm(x_1 - x_2, ord=np.inf, axis=-1)
    elif norm in ["2", 2, "L2", "l2"]:
        distance = np.linalg.norm(x_1 - x_2, ord=2, axis=-1)
    else:
        raise NotImplementedError

    return distance
```

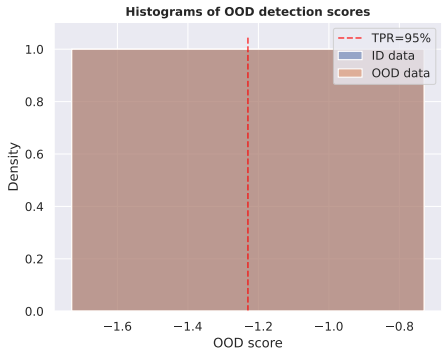
One can define any new distance metric, like structural similarity index measure (SSIM), or some semantic measure after embedding the features x_1 and x_2. The distance used here does not need to be differentiable and is not backpropagated in the gradient attacks.

Hence, for CAPGD component of the benchmark attack, we need to define a custom project mechanism for each distance. We implemented a projection over sphere of L_∞ and L_2 distances `https://github.com/serval-uni-lu/tabularbench/blob/main/tabularbench/attacks/capgd/capgd.py#L196`.

[6] `https://github.com/serval-uni-lu/tabularbench/blob/main/tabularbench/attacks/utils.py`



(a) AUC-ROC curve for the OOD detector

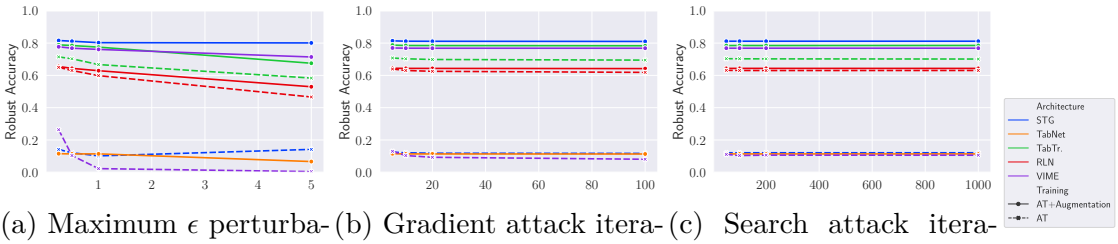


(b) Histograms of OOD detection scores

Figure 10.6: Performance of the OOD detector on the WGAN samples.

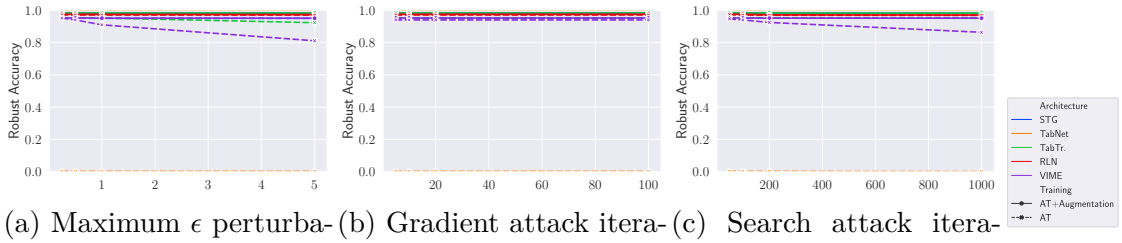
Table 10.16: Pearson correlations between constrained robust accuracy and: ID accuracy (ID), and non constrained-accuracy (ADV)

Dataset	Training	ID(corr)	ID(p-val)	ADV(corr)	ADV(p-val)
CTU	Adversarial	1	1.4e-26	1	1.9e-31
CTU	Standard	0.22	0.28	0.22	0.28
LCLD	Adversarial	0.76	1.8e-06	0.76	1.8e-06
LCLD	Standard	0.15	0.39	0.15	0.39
URL	Adversarial	0.7	3.6e-06	1	7.2e-37
URL	Standard	0.19	0.26	0.46	0.0053
WIDS	Adversarial	0.79	1e-06	0.91	7e-11
WIDS	Standard	0.031	0.87	0.62	0.00025



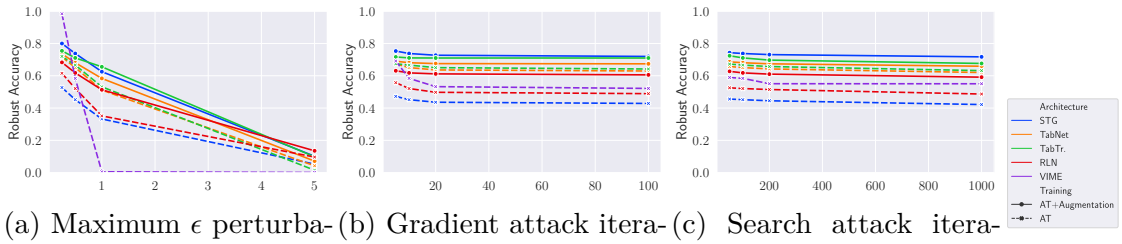
(a) Maximum ϵ perturbation (b) Gradient attack iterations (c) Search attack iterations

Figure 10.7: Impact of attack budget on the robust accuracy for LCLD dataset.



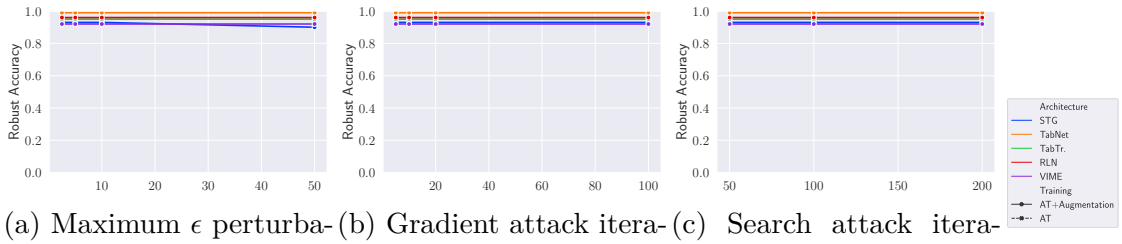
(a) Maximum ϵ perturbation (b) Gradient attack iterations (c) Search attack iterations

Figure 10.8: Impact of attack budget on the robust accuracy for CTU dataset.



(a) Maximum ϵ perturbation (b) Gradient attack iterations (c) Search attack iterations

Figure 10.9: Impact of attack budget on the robust accuracy for WIDS dataset.



(a) Maximum ϵ perturbation (b) Gradient attack iterations (c) Search attack iterations

Figure 10.10: Impact of attack budget on the robust accuracy for Malware dataset.

To extend the projected gradient attacks to other distances, custom projection mechanisms are then needed.

10.4 TabularBench: Benchmarking Adversarial Robustness for Tabular Deep Learning in Real-world Use-cases

10.4.1 API

The library `https://github.com/serval-uni-lu/tabularbench` is split into 4 main components. The *test* folder provides meaningful examples for each component.

Datasets

Our dataset factory supports 5 datasets: CTU, LCLD, MALWARE, URL, and WIDS. each dataset can be invoked with the following aliases:

```
from tabularbench.datasets import dataset_factory

dataset_aliases= [
 "ctu_13_neris",
 "lcld_time",
 "malware",
 "url",
 "wids",
 ]

for dataset_name in dataset_aliases:
 dataset = dataset_factory.get_dataset(dataset_name)
 x, _ = dataset.get_x_y()
 metadata = dataset.get_metadata(only_x=True)
 assert x.shape[1] == metadata.shape[0]
```

Each dataset can be defined in a single .py file (example: `https://github.com/serval-uni-lu/tabularbench/blob/main/tabularbench/datasets/samples/url.py`).

A dataset needs at least a source (local or remote csv) for the raw features and a definition of feature constraints. The said definition can be empty for non-constrained datasets.

Constraints

One of the features of our benchmark is the support of feature constraints, but in the dataset definition and in the attacks.

Constraints can be expressed in natural language. For example, we express the constraint $F_0 = F_1 + F_2$ such as:

```
from tabularbench.constraints.relation_constraint import Feature
constraint1 = Feature(0) == Feature(1) + Feature(2)
```

Given a dataset, one can check the constraint satisfaction over all constraints, given a tolerance.

```python
from tabularbench.constraints.constraints_checker import ConstraintChecker
from tabularbench.datasets import dataset_factory

dataset = dataset_factory.get_dataset("url")
x, _ = dataset.get_x_y()

constraints_checker = ConstraintChecker(
 dataset.get_constraints(), tolerance
)
out = constraints_checker.check_constraints(x.to_numpy())
```

In the provided datasets, all constraints are satisfied. During the attack, Constraints can be fixed as follows:

```python
import numpy as np
from tabularbench.constraints.constraints_fixer import ConstraintsFixer

x = np.arange(9).reshape(3, 3)

constraints_fixer = ConstraintsFixer(
 guard_constraints=[constraint1],
 fix_constraints=[constraint1],
 )

x_fixed = constraints_fixer.fix(x)

x_expected = np.array([[3, 1, 2], [9, 4, 5], [15, 7, 8]])

assert np.equal(x_fixed, x_expected).all()
```

Constraint violations can be translated into losses and one can compute the gradient to repair the faulty constraints as follows:

```python
import torch

from tabularbench.constraints.constraints_backend_executor import (
 ConstraintsExecutor,
)

from tabularbench.constraints.pytorch_backend import PytorchBackend
from tabularbench.datasets.dataset_factory import get_dataset

ds = get_dataset("url")
constraints = ds.get_constraints()
constraint1 = constraints.relation_constraints[0]

x, y = ds.get_x_y()
x_metadata = ds.get_metadata(only_x=True)
x = torch.tensor(x.values, dtype=torch.float32)

```

```
constraints_executor = ConstraintsExecutor(
 constraint1,
 PytorchBackend(),
 feature_names=x_metadata["feature"].to_list(),
)

x.requires_grad = True
loss = constraints_executor.execute(x)
grad = torch.autograd.grad(
 loss.sum(),
 x_1,
)[0]
```

Models

All models need to extend the class **BaseModelTorch**[7] . This class implements the definitions, the fit and evaluation methods, and the save and loading methods. Depending of the architectures, scaler and feature encoders can be required by the constructors.

So far, our API nateively supports: multi-layer preceptrons (MLP), RLN, STG, TabNet, TabTransformer, and VIME. Our implementation is based on Tabsurvey [BLS+21]. All models from this framework can be easily adapted to our API.

Benchmark

The leaderboard is available on `https://serval-uni-lu.github.io/tabularbench/`.

This leaderboard will be updated regularly, and all the models listed in the leaderboard are downloadable using our API

The benchmark leverages Constrained Adaptive Attack (CAA) by default and can be extended for other attacks.

```
clean_acc, robust_acc = benchmark(dataset='LCLD', model="TabTr_Cutmix",
                                  distance='L2', constraints=True)
```

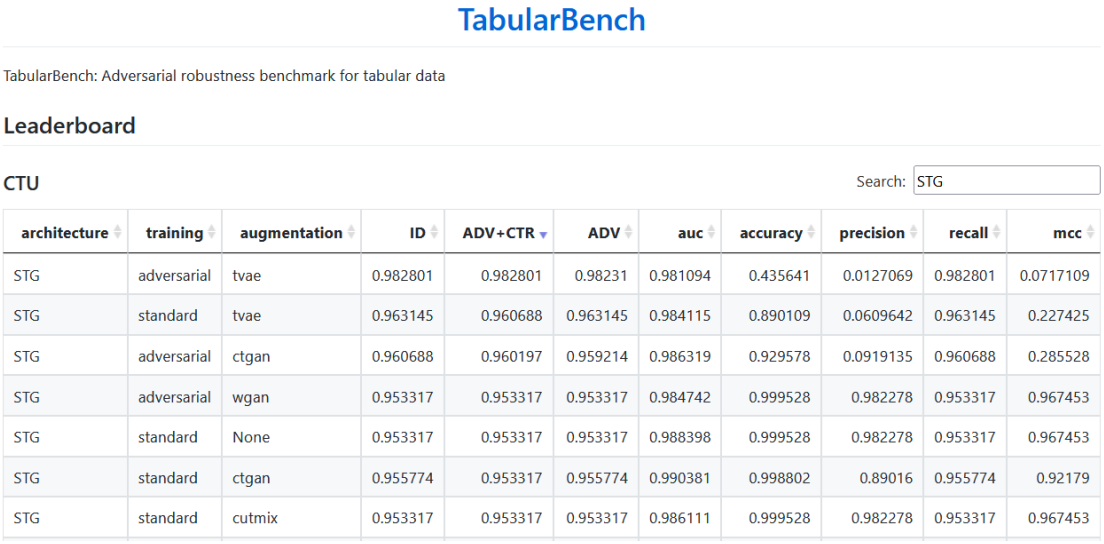
The model attribute refers to a pre-trained model in the relevant model folder. The API infers the architecture from the first term of the model name, but it can be defined manually. In the above example, a **TabTransformer** architecture will be initialized.

10.5 On the Impact of Industrial Delays when Mitigating Distribution Drifts: an Empirical Study on Real-world Financial Systems

10.5.1 Generalization study

In the following, we will assess the validity of each of our claims across our three datasets. We summarize in Table 10.17. In this generalization study we evaluate two variants for the electricity dataset when studying the impact of detectors and delay. A scenario where the total delay is 10 days and a scenario where the delay is 28 days.

[7] `https://github.com/serval-uni-lu/tabularbench/blob/main/tabularbench/models/torch_models.py`



architecture	training	augmentation	ID	ADV+CTR	ADV	auc	accuracy	precision	recall	mcc
STG	adversarial	tvae	0.982801	0.982801	0.98231	0.981094	0.435641	0.0127069	0.982801	0.0717109
STG	standard	tvae	0.963145	0.960688	0.963145	0.984115	0.890109	0.0609642	0.963145	0.227425
STG	adversarial	ctgan	0.960688	0.960197	0.959214	0.986319	0.929578	0.0919135	0.960688	0.285528
STG	adversarial	wgan	0.953317	0.953317	0.953317	0.984742	0.999528	0.982278	0.953317	0.967453
STG	standard	None	0.953317	0.953317	0.953317	0.988398	0.999528	0.982278	0.953317	0.967453
STG	standard	ctgan	0.955774	0.953317	0.955774	0.990381	0.998802	0.89016	0.955774	0.92179
STG	standard	cutmix	0.953317	0.953317	0.953317	0.986111	0.999528	0.982278	0.953317	0.967453

Figure 10.11: Screenshot of the TabularBench leaderboard on 12/06/2024

Table 10.17: Summary of the generalization study (Extended). ✓means the claim holds, ✗ means the claim does not hold, and ✓* means the claim holds in some cases.

	Claim	BGL	LCLD	Elec. A	Elec. B
Model Tuning	Tuning hyperparameters has a positive impact on performance	✓	✓	✓	✓
	Tuning hyperparameters outperforms model retraining	✓	✓*	✗	✗
	Re-tuning does not bring additional improvements	✓	✓*	✗	✗
	Training with all available data is counterproductive	✓	✓	✓	✓
Detector selection	Periodic retraining + error-based offer flexible compromise	✓	✗	✗	✓*
	Periodic retraining is effective for low retraining budgets	✓	✗	✗	✗
	Error-based detectors are best for high retraining budgets	✓	✓	✗	✓
	In lower budgets, error-based detectors outperforms periodic	✓	✓	✗	✓
Impact of delay	Not considering delays overestimates the trade-off	✓	✓	-	✓
	Enabling delay disrupt the ranking of drift detectors	✓	✓	-	✓
	Change in delay has an inverse effect on efficiency/effectiveness	✓	✓	-	✓
	Change in delay disrupts the ranking of drift detectors	✓	✓	-	✓

Overall, our study confirms that the use of all available training data is not necessary and even counter-productive, that the best detectors can notably vary across datasets and need specific evaluation, and that introducing the delay in the evaluation pipeline significantly impacts which detectors and retraining strategies are on the Pareto-front to achieve the best trade-off between effectiveness and efficiency. We will detail each of the claims in the following subsections.

Datasets

Lending Club Loan Data (LCLD) We apply our study to the LCLD dataset [Kag19] which is the public counterpart of BGL BNP Paribas' dataset. The objective of the ML model is to classify loan requests as accepted (high probability of reimbursement) or refused (low probability of reimbursement) based on information provided by the client (e.g. age, purpose of the loan, duration), publicly available data (e.g. credit score) and computed feature (e.g. installment in the case the loan would be accepted). By default, the LCLD dataset contains features that are not available at the time of the request. We ignored these features to avoid data leakage and use the remaining 28 features. The dataset contains 1,124,606 labeled inputs from transactions that occurred over a 5-year period. The timestamps associated with inputs are precise for a month. For each month, we uniformly distribute the inputs of that month on the weekdays and associate them with a new timestamp. The dataset is similar in size and time span to our partner's. Therefore, we reuse our partner parameters and constraints to conduct our experiment. The delays remain 10 days for labeling and four weeks for deployment. We trained the baseline model with 400,000 samples. The minimum period of retraining corresponds to the average label delay which is 5,000 samples.

Electricity Electricity [Har99] is a widely used dataset in the distribution shift on tabular data literature as shown in [LLD+19]. The classification task is to determine at any point in time, if the electricity price is going up or down. The 6 features include the day of the week, the current price, the electricity demand, as well as the price, demand, and transfer of the adjacent geographical region. This dataset is smaller than the two others. The dataset contains 45,312 labeled inputs that occur over 943 days with exactly one input recorded every 30 minutes. The dataset is precise to 30 minutes. This dataset is small than the two others, spans over a shorter period of time, and is drawn from another domain. Hence the labeling and deployment delays may differ in a real-world system. We started with our partner's delay. The minimum period of retraining corresponds to the average label delay, which is 480 samples. We trained the baseline model with one year of data. We round up to the next multiple of 480 to facilitate model reuse during the experiments and obtain 17760 inputs. With such high delays, we found that none of the schedules, including retraining every 10 days, can outperform the baseline. In fact, as shown in our experiments in Appendix 10.5.1 not retraining and keeping the original model is the most effective strategy with such delays. This confirms the importance of considering the delay in the evaluation of retraining schedules. We also consider a scenario with shorter delays. We conserve a total delay of 10 days corresponding to our previous label delay. We assume that at any given time the electricity price of the previous day is available. Hence, we split this total delay into 1 day for labeling and 9 days for production.

The metric (MCC for unbalanced data) and classifier (Random Forest for interpretability) remain unchanged for both datasets.

Model tuning

In Table 10.18 and Table 10.19, we compare the effectiveness of various training strategies when the model is deployed right before the three most important drifts, on LCLD and Electricity datasets respectively. Both retraining and hyper-parameter tuning yield improvements

in performance, however in the case of Electricity, retraining yields better improvements than hyper-parameter tuning.

Table 10.20 and table 10.21 confirm that hyper-parameter tuning remains relevant in the periodic training scenarios.

Table 10.18: ML effectiveness (MCC) of different training strategies for different testing windows for LCLD dataset. For not retrained models, the data used are the same as the initial model. For retrained model, we use the data that directly precede the test window.

Retrained	Model Hyperparameters	MCC on [T, T + 20k[at T = 400k	420k	600k	940k
No	No tuning	0.2791	0.2609	0.2691	0.2165
	Initial tuning	0.2820	0.2606	0.2798	0.2213
Yes	No tuning	-	0.2583	0.2743	0.2249
	Initial tuning	-	0.2650	0.2789	0.2290
	Re-tuning	-	0.2616	0.2831	0.2240

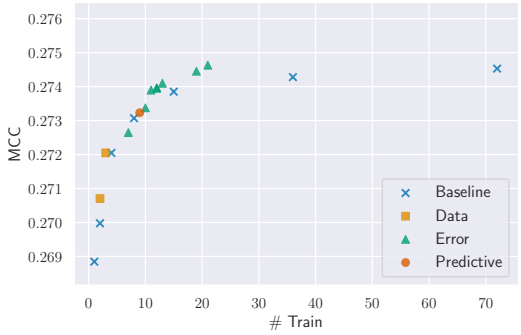
Table 10.19: ML effectiveness (MCC) of different training strategies for different testing windows for the Electricity dataset. For not retrained models, the data used are the same as the initial model. For retrained model, we use the data that directly precede the test window.

Retrained	Model Hyperparameters	MCC on [T, T + 480[at T = 20160	22560	32160
No	Initial tuning	0.1281	0.0	0.0
Yes	Initial tuning	0.1900	0.2415	0.2807
	Re-tuning	0.2162	0.3597	0.4633

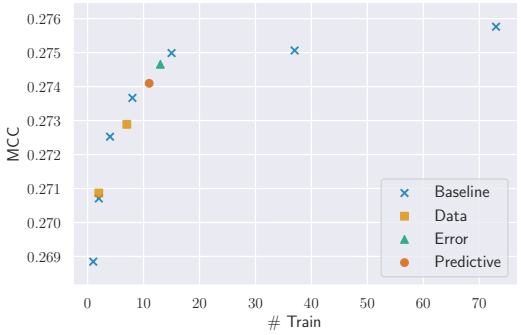
Next, when assessing which retraining window is the best for each dataset (Table 10.22 for LCLD and Table 10.23 for Electricity), we confirm our previous insights that reusing all the training data is not always the best. 200k examples are generally sufficient for LCLD and a retraining window of 4440 is sufficient for the Electricity dataset. For the Electricity dataset, increasing the period of retraining requires a larger window size. In four over six periodic schedules, the best window size is 4400, while for the two remaining larger periods, retraining with a window size larger than 35520 achieves the best effectiveness. Overall for each scenario, the best effectiveness regardless of the period is achieved with the smallest window size. This confirms that retraining with all data is counterproductive for periodic retraining.

Drift detectors

Our evaluation on LCLD (Fig 10.12) and on Electricity (Fig 10.13) confirms that driving the training with error-based detectors is a relevant strategy both in low and high retraining budgets.

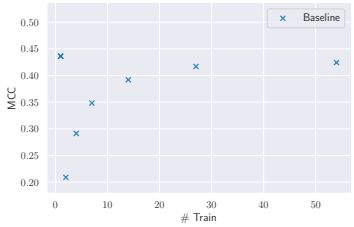


(a) With delays $\delta_l = 10$ days, $\delta_d = 4$ weeks

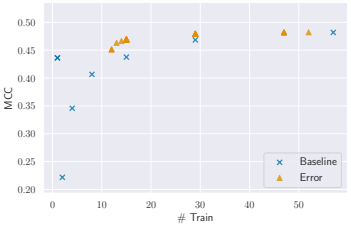


(b) Without delays

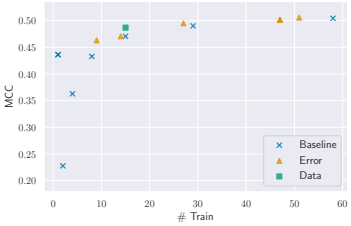
Figure 10.12: Pareto front of drift detectors, no retraining and periodic retraining schedules on LCLD dataset.



(a) With delays $\delta_l = 10$ days, $\delta_d = 4$ weeks



(b) With delays $\delta_l = 1$ day, $\delta_d = 9$ days



(c) Without delays

Figure 10.13: Pareto front of drift detectors, no retraining and periodic retraining schedules on Electricity dataset.

Table 10.20: ML effectiveness (MCC) of periodic retraining strategies for LCLD dataset.

Model hyperparameters	Retraining period 20,000	50,000	100,000	200,000	400,000
No tuning	0.2691	0.2688	0.2681	0.2678	0.2669
Initial tuning	**0.2742**	**0.2737**	**0.2725**	**0.2723**	**0.2709**
Re-tuning	**0.2751**	**0.2744**	**0.2734**	**0.2711**	**0.2685**

Table 10.21: ML effectiveness (MCC) of periodic retraining strategies for Electricity dataset.

Model hyperparameters	Retraining period 480	960	1920	3840	7680	15360
Initial tuning	0.3908	0.3800	0.3564	0.3346	0.3160	0.3174
Re-tuning	0.4266	0.4132	0.3950	0.3565	0.3386	0.3125

These error-based detectors outperform the other type of detectors under high retraining budgets. In addition, under limited retraining budgets, retraining with error-based detectors is always more effective than periodic retraining. Contrary to our industrial partner's setting, periodic retraining is not effective for low training budgets in the cases of LCLD and Electricity datasets. Finally, Our evaluation of Electricity A, which has high labeling and deployment delays, shows that no retraining strategy is better than the baseline. It is only when we reduce the delays (such as in Scenario Electricity (B) where error-based detectors outperform the baseline.

Delays

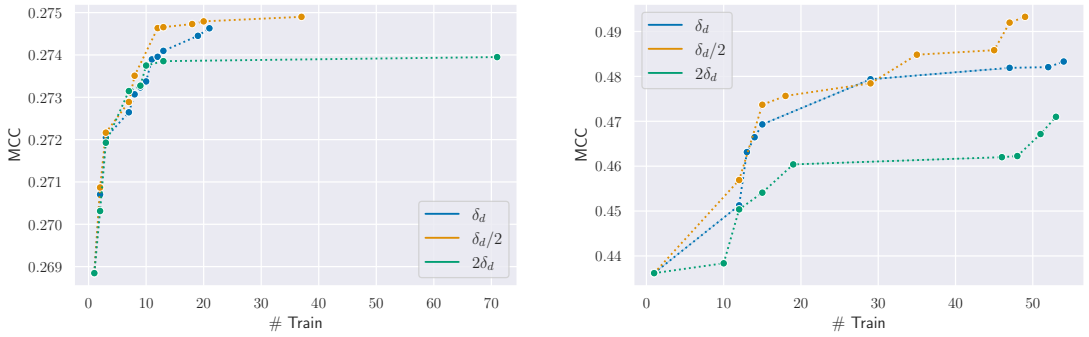
We do not evaluate the impact of delay for the scenarios Electricity A as none of the detectors outperforms the baseline (as studied in the previous section).

When comparing the occurrences on the Pareto-front of each schedule strategy between the scenario with and without delays (Table 10.24 for LCLD and Table 10.25 for Electricity), we confirm that the optimal schedule varies. KSWIN becomes optimal with delay for LCLD and Electrictity, while HDDM-A and ADWIN become relevant for Electricity. Similarly, changing the delay as shown in Table 10.26 for LCLD and Table 10.27 for Electricity has an impact on which schedule strategy remains optimal. ADWIN and KSWIN are no longer on the Pareto-front when halving the delay for Electricity and HDDM-A is no longer on the front when we double the delay.

Overall, doubling or halving the delay can have a significant impact on the Pareto-front of drift detectors. In Figure 10.13, the MCC drops by more than 5% between the scenarios $\delta_d/2$ and $2\delta_d$.

Table 10.22: ML effectiveness (MCC) of periodic retraining schedules for LCLD dataset.

Delays	Period	Window Size 50,000	100,000	200,000	400,000	All
Yes	5,000	0.2730	**0.2740**	**0.2746**	0.2736	0.2739
	10,000	0.2730	**0.2740**	**0.2745**	0.2735	0.2739
	20,000	0.2730	**0.2738**	**0.2743**	0.2735	0.2737
	50,000	0.2728	**0.2740**	**0.2739**	0.2732	0.2732
	100,000	0.2728	0.2728	**0.2731**	0.2725	**0.2729**
	200,000	0.2712	0.2718	**0.2720**	0.2716	**0.2723**
	400,000	0.2691	0.2696	0.2700	**0.2704**	**0.2709**
No	5,000	0.2736	0.2745	**0.2756**	**0.2747**	0.2742
	10,000	0.2737	**0.2744**	**0.2758**	0.2743	0.2742
	20,000	0.2732	**0.2747**	**0.2751**	0.2742	0.2741
	50,000	0.2737	**0.2748**	**0.2750**	0.2737	0.2739
	100,000	0.2726	**0.2733**	**0.2737**	0.2725	0.2730
	200,000	0.2716	0.2722	**0.2725**	0.2722	**0.2726**
	400,000	0.2697	0.2703	0.2707	**0.2709**	**0.2712**



(a) LCLD dataset.

(b) Electricity dataset.

Figure 10.14: Pareto front of retraining schedule with same parameters and different deployment delays.

Table 10.23: ML effectiveness (MCC) of periodic retraining schedules for Electricity dataset.

		Window Size				
Delays	Period	4440	8880	17760	35520	All
Scenario A $\delta_l = 10$ days $\delta_d = 4$ weeks	480	**0.4243**	**0.3879**	0.3531	0.3848	0.3827
	960	**0.4169**	0.3793	0.3442	0.3801	**0.3806**
	1920	**0.3920**	0.3603	0.3289	**0.3663**	0.3645
	3840	**0.3483**	0.3246	0.3114	0.3438	**0.3538**
	7680	0.2914	0.2926	0.3126	**0.3352**	**0.3377**
	15360	0.2090	0.2271	0.2977	**0.3064**	**0.3064**
Scenario B $\delta_l = 1$ day $\delta_d = 9$ days	480	**0.4818**	**0.4551**	0.4032	0.4224	0.4121
	960	**0.4684**	**0.4458**	0.3915	0.4211	0.4125
	1920	**0.4376**	**0.4247**	0.3753	0.3968	0.3923
	3840	**0.4066**	0.3664	0.3347	0.3725	**0.3762**
	7680	0.3457	0.3340	0.3293	**0.3615**	**0.3634**
	15360	0.2217	0.2378	0.3060	**0.3153**	**0.3153**
None	480	**0.5045**	**0.4818**	0.4266	0.4385	0.4288
	960	**0.4902**	**0.4713**	0.4132	0.4260	0.4175
	1920	**0.4705**	**0.4480**	0.3950	0.4102	0.4042
	3840	**0.4329**	**0.3899**	0.3565	0.3859	0.3872
	7680	0.3628	0.3531	0.3386	**0.3723**	**0.3738**
	15360	0.2276	0.2470	0.3125	**0.3228**	**0.3228**

Table 10.24: Evaluation of the Pareto-front of different schedules on LCLD dataset. For each schedule, we report the number of parameter settings (A) / (B). (A) are the settings on the efficiency/effectiveness Pareto front across all methods. (B) are the settings on the Pareto front local to the method.

Type	Schedule	Delay	
		No	$\delta_l = 10$ days $\delta_d = 4$ weeks
Baseline	No detection	1 / 1	1 / 1
	Periodic	5 / 7	1 / 7
Data-based detector	Statistical test	0 / 9	0 / 9
	Divergence	1 / 4	1 / 5
	PCA-CD	1 / 2	1 / 2
Error-based detector	ADWIN (CE)	1 / 3	3 / 6
	ADWIN (PE)	0 / 5	0 / 5
	DDM	0 / 5	0 / 2
	EDDM	0 / 6	0 / 6
	HDDM-A	0 / 19	0 / 4
	HDDM-W	0 / 1	0 / 17
	KSWIN (CE)	0 / 7	2 / 5
	KSWIN (PE)	0 / 4	2 / 4
	Page-Hinkley (CE)	0 / 1	0 / 2
	Page-Hinkley (PE)	0 / 1	0 / 1
Predictive-based detector	Uncertainty	0 / 3	0 / 5
	Aries ADWIN	1 / 5	1 / 3

Table 10.25: Evaluation of the Pareto-front of different schedules on Electricity B dataset. For each schedule, we report the number of parameter settings (A) / (B). (A) are the settings on the efficiency/effectiveness Pareto front across all methods. (B) are the settings on the Pareto front local to the method.

Type	Schedule	Delay	
		No	$\delta_l = 1$ day $\delta_d = 9$ days
Baseline	No detection	1 / 1	1 / 1
	Periodic	1 / 7	1 / 7
Data-based detector	Statistical test	2 / 9	6 / 7
	Divergence	1 / 4	0 / 5
	PCA-CD	0 / 0	0 / 0
Error-based detector	ADWIN (CE)	0 / 10	3 / 4
	ADWIN (PE)	0 / 4	3 / 4
	DDM	1 / 3	0 / 3
	EDDM	0 / 4	0 / 4
	HDDM-A	0 / 23	7 / 7
	HDDM-W	1 / 2	0 / 2
	KSWIN (CE)	0 / 5	3 / 5
	KSWIN (PE)	0 / 9	1 / 2
	Page-Hinkley (CE)	0 / 2	0 / 2
	Page-Hinkley (PE)	1 / 1	0 / 1
Predictive-based detector	Uncertainty	1 / 4	4 / 4
	Aries ADWIN	1 / 4	1 / 3

Table 10.26: Parameters on the Pareto front with delay δ_d that remains on the Pareto front for delays $\delta_d/2$ and $2\delta_d$ on LCLD dataset.

Type	Detector	Delay $\delta_d/2$	Delay δ_d	Delay $2\delta_d$
Baseline	No detection	1	1	1
	Periodic	0	1	1
Data-based detector	Statistical test	0	0	0
	Divergence	2	1	1
	PCA-CD	1	1	1
Error-based detector	ADWIN (CE)	2	3	1
	ADWIN (PE)	0	0	0
	DDM	0	0	0
	EDDM	0	0	0
	HDDM-A	1	0	1
	HDDM-W	0	0	0
	KSWIN (CE)	3	2	0
	KSWIN (PE)	0	2	0
	Page-Hinkley (CE)	0	0	0
	Page-Hinkley (PE)	0	0	1
Predictive-based detector	Uncertainty	0	0	0
	Aries ADWIN	0	1	1

Table 10.27: Parameters on the Pareto front with delay δ_d that remains on the Pareto front for delays $\delta_d/2$ and $2\delta_d$ on Electricity B dataset.

Type	Detector	Delay $\delta_d/2$	Delay δ_d	Delay $2\delta_d$
Baseline	No detection	1	1	1
	Periodic	2	1	1
Data-based detector	Statistical test	4	6	7
	Divergence	1	0	0
	PCA-CD	0	0	0
Error-based detector	ADWIN (CE)	0	3	2
	ADWIN (PE)	0	3	0
	DDM	0	0	0
	EDDM	0	0	0
	HDDM-A	1	7	0
	HDDM-W	1	0	2
	KSWIN (CE)	0	3	0
	KSWIN (PE)	1	1	0
	Page-Hinkley (CE)	2	0	0
	Page-Hinkley (PE)	1	0	1
Predictive-based detector	Uncertainty	3	4	4
	Aries ADWIN	0	1	0

List of publications and tools

Papers included in the dissertation

- Thibault Simonetto et al. A unified framework for adversarial attack and defense in constrained feature space. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pages 1313–1319, 2022
- Thibault Simonetto et al. Towards adaptive attacks on constrained tabular machine learning. In *ICML 2024 Next Generation of AI Safety Workshop*, 2024
- Thibault Simonetto et al. On the impact of industrial delays when mitigating distribution drifts: an empirical study on real-world financial systems. In *KDD 2024 Discovering Drift Phenomena in Evolving Landscape Workshop*, 2024
- Thibault Simonetto et al. Constrained adaptive attack: effective adversarial attack against deep neural networks for tabular data. In *Advances in Neural Information Processing Systems*, 2024
- Thibault Simonetto et al. Tabularbench: benchmarking adversarial robustness for tabular deep learning in real-world use-cases. In *Advances in Neural Information Processing Systems Datasets and Benchmarks Track*, 2024

Papers not included in the dissertation

- Salijona Dyrmishi et al. On the empirical effectiveness of unrealistic adversarial hardening against realistic adversarial attacks. In *2023 IEEE symposium on security and privacy (SP)*, pages 1384–1400. IEEE, 2023

Tools included in the dissertation

- TabularBench: Adversarial robustness benchmark for tabular data.
  - `https://github.com/serval-uni-lu/tabularbench`
- Drift robustness: Evaluation of distribution drift robustness under realistic settings.
  - `https://github.com/serval-uni-lu/drift-robustness`

List of figures

List of tables

Bibliography

[AAM+17] Mehreen Ahmed, Hammad Afzal, Awais Majeed, and Behram Khan. A survey of evolution in predictive models and impacting factors in customer churn. *Advances in Data Science and Adaptive Analysis*, 9(03):1750007, 2017 (cited on page 16).

[ABB+19] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. Software Engineering for Machine Learning: A Case Study. In *Proceedings - 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice, ICSE-SEIP 2019*, pages 291–300, 2019. ISBN: 978-1-72811-760-7. DOI: 10.1109/ICSE-SEIP.2019.00042. URL: https://docs.microsoft.com/en-us/azure/devops/learn/devops-at-microsoft/ (visited on 11/19/2020) (cited on page 92).

[ACB17] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017 (cited on pages 70, 86).

[AGM+20] Hojjat Aghakhani, Fabio Gritti, Francesco Mecca, Martina Lindorfer, Stefano Ortolani, Davide Balzarotti, Giovanni Vigna, and Christopher Kruegel. When malware is packin'heat; limits of machine learning classifiers based on static analysis features. In *Network and Distributed Systems Security (NDSS) Symposium 2020*, 2020 (cited on pages 10, 16, 26, 28, 34, 117, 118, 137, 139).

[AP21] Sercan Ö Arik and Tomas Pfister. Tabnet: attentive interpretable tabular learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35 of number 8, pages 6679–6687, 2021 (cited on pages 16, 17, 48, 72, 79, 86, 129, 140).

[ASE+18] Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. Generating natural language

adversarial examples. *arXiv preprint arXiv:1804.07998*, 2018 (cited on page 26).

[BAL+19] Vincent Ballet, Jonathan Aigrain, Thibault Laugel, Pascal Frossard, Marcin Detyniecki, et al. Imperceptible adversarial attacks on tabular data. In *NeurIPS 2019 Workshop on Robust AI in Financial Services: Data, Fairness, Explainability, Trustworthiness and Privacy (Robust AI in FS 2019)*, 2019 (cited on pages 17, 18, 44, 66, 84, 89, 111).

[BCF+06] Manuel Baena-García, José Campo-Ávila, Raúl Fidalgo-Merino, Albert Bifet, Ricard Gavald, and Rafael Morales-Bueno. Early drift detection method, January 2006 (cited on pages 3, 20, 21).

[BDD+21] Julian Blank, Kalyanmoy Deb, Yashesh Dhebar, Sunith Bandaru, and Haitham Seada. Generating well-spaced points on a unit simplex for evolutionary many-objective optimization. *IEEE Transactions on Evolutionary Computation*, 25(1):48–60, 2021. DOI: 10.1109/TEVC.2020.2992387 (cited on page 33).

[BDF+24] Matan Ben-Tov, Daniel Deutch, Nave Frost, and Mahmood Sharif. Cafa: cost-aware, feasible attacks with database constraints against neural tabular classifiers. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 227–227. IEEE Computer Society, 2024 (cited on page 111).

[BG07a] Albert Bifet and Ricard Gavaldà. Learning from time-changing data with adaptive windowing. In *Proceedings of the 7th SIAM International Conference on Data Mining*, pages 443–448, 2007. ISBN: 978-0-89871-630-6. DOI: 10.1137/1.9781611972771.42. URL: http://www.lsi.upc.edu/~%7Babifet, (visited on 02/01/2022) (cited on pages 3, 20, 21).

[BG07b] Albert Bifet and Ricard Gavaldà. Learning from time-changing data with adaptive windowing. In *Proceedings of the 7th SIAM International Conference on Data Mining*, pages 443–448, 2007. ISBN: 978-0-89871-630-6. DOI: 10.1137/1.9781611972771.42. URL: http://www.lsi.upc.edu/~%7Babifet, (visited on 02/01/2022) (cited on page 92).

[BH95] Yoav Benjamini and Yosef Hochberg. Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing. en. *Journal of the Royal Statistical Society: Series B (Methodological)*, 57(1):289–300, 1995. ISSN: 2517-6161. DOI: 10.1111/j.2517-6161.1995.tb02031.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.2517-6161.1995.tb02031.x (visited on

05/19/2023). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.2517-6161.1995.tb02031.x (cited on page 20).

[BHZ+23] Hongyan Bao, Yufei Han, Yujun Zhou, Xin Gao, and Xiangliang Zhang. Towards efficient and domain-agnostic evasion attack with high-dimensional categorical inputs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37 of number 6, pages 6753–6761, 2023 (cited on pages 18, 44, 66, 84).

[BLS+21] Vadim Borisov, Tobias Leemann, Kathrin Seßler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. Deep neural networks and tabular data: a survey. *arXiv preprint arXiv:2110.01889*, 2021 (cited on pages 61, 72, 84, 86, 161).

[BLS+22] Vadim Borisov, Tobias Leemann, Kathrin Sessler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. Deep neural networks and tabular data: a survey. *IEEE Transactions on Neural Networks and Learning Systems*:1–21, 2022. DOI: 10.1109/tnnls.2022.3229161. URL: https://doi.org/10.1109%2Ftnnls.2022.3229161 (cited on pages 16, 44, 48).

[BLS11] Gilles Blanchard, Gyemin Lee, and Clayton Scott. Generalizing from several related classification tasks to a new unlabeled sample. In *NeurIPS*, 2011 (cited on page 22).

[CAF+21] Francesco Cartella, Orlando Anunciaçao, Yuki Funabiki, Daisuke Yamaguchi, Toru Akishita, and Olivier Elshocht. Adversarial attacks for tabular data: application to fraud detection and imbalanced data, 2021 (cited on pages 10, 16, 18).

[CAP+19] Nicholas Carlini, Anish Athalye, Nicolas Papernot, Wieland Brendel, Jonas Rauber, D. Tsipras, I. Goodfellow, A. Madry, and A. Kurakin. On evaluating adversarial robustness. *ArXiv*, abs/1902.06705, 2019 (cited on page 37).

[Car23] Nicholas Carlini. A llm assisted exploitation of ai-guardian. *arXiv preprint arXiv:2307.15008*, 2023 (cited on pages 5, 19, 60, 67, 89).

[CAS+20] Francesco Croce, Maksym Andriushchenko, Vikash Sehwag, Edoardo Debenedetti, Nicolas Flammarion, Mung Chiang, Prateek Mittal, and Matthias Hein. Robustbench: a standardized adversarial robustness benchmark. *arXiv preprint arXiv:2010.09670*, 2020 (cited on pages 66, 84).

[CH20] Francesco Croce and Matthias Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In *ICML*, 2020 (cited on pages 6, 49).

[CHS18] Ramiro Camino, Christian Hammerschmidt, and Radu State. Generating multi-categorical samples with generative adversarial networks. *ICML workshop on Theoretical Foundations and Applications of Deep Generative Models*, 2018 (cited on page 70).

[CO19a] Alesia Chernikova and Alina Oprea. FENCE: Feasible Evasion Attacks on Neural Networks in Constrained Environments, 2019. URL: `http://arxiv.org/abs/1909.10480`. arXiv: 1909.10480 (cited on page 4).

[CO19b] Alesia Chernikova and Alina Oprea. Fence: feasible evasion attacks on neural networks in constrained environments. *arXiv preprint arXiv:1909.10480*, 2019 (cited on pages 10, 16, 18, 26, 28, 34, 117, 128, 139).

[CO22] Alesia Chernikova and Alina Oprea. Fence: feasible evasion attacks on neural networks in constrained environments. *ACM Transactions on Privacy and Security*, 25(4):1–34, 2022 (cited on pages 46, 47, 72, 86, 126, 137).

[CSD19] Raphael Labaca Castro, Corinna Schmitt, and Gabi Dreo. Aimed: evolving malware with genetic programming to evade detection. In *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pages 240–247. IEEE, 2019 (cited on pages 3, 118, 139).

[CWC20] Jiyu Chen, David Wang, and Hao Chen. Explore the transformation space for adversarial images. In *Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy*, pages 109–120, 2020 (cited on page 12).

[CXY+20] Jillian M Clements, Di Xu, Nooshin Yousefi, and Dmitry Efimov. Sequential deep learning for credit risk monitoring with tabular financial data. *arXiv preprint arXiv:2012.15330*, 2020 (cited on page 16).

[DDS+04] Nilesh Dalvi, Pedro Domingos, Sumit Sanghai, and Deepak Verma. Adversarial classification. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 99–108, 2004 (cited on page 26).

[DFY+20] Yinpeng Dong, Qi-An Fu, Xiao Yang, Tianyu Pang, Hang Su, Zihao Xiao, and Jun Zhu. Benchmarking adversarial robustness on image classification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 321–331, 2020 (cited on page 84).

[DGC23] Salijona Dyrmishi, Salah Ghamizi, and Maxime Cordy. How do humans perceive adversarial text? a reality check on the validity and naturalness of word-based adversarial attacks, 2023. arXiv: 2305.15587 [cs.CL] (cited on page 45).

[DGS+22] Salijona Dyrmishi, Salah Ghamizi, Thibault Simonetto, Yves Le Traon, and Maxime Cordy. On the empirical effectiveness of unrealistic adversarial hardening against realistic adversarial attacks. *arXiv preprint arXiv:2202.03277*, 2022 (cited on pages 3, 17, 72, 86, 89, 111, 112).

[DGS+23] Salijona Dyrmishi, Salah Ghamizi, Thibault Simonetto, Yves Le Traon, and Maxime Cordy. On the empirical effectiveness of unrealistic adversarial hardening against realistic adversarial attacks. In *2023 IEEE symposium on security and privacy (SP)*, pages 1384–1400. IEEE, 2023 (cited on page i).

[DJL+20] Tianyu Du, Shouling Ji, Jinfeng Li, Qinchen Gu, Ting Wang, and Raheem Beyah. Sirenattack: generating adversarial audio for end-to-end acoustic systems. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, pages 357–369, 2020 (cited on page 26).

[DLP+18] Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, and Jianguo Li. Boosting adversarial attacks with momentum. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9185–9193, 2018 (cited on page 49).

[DSO07] Kalyanmoy Deb, Karthik Sindhya, and Tatsuya Okabe. Self-adaptive simulated binary crossover for real-parameter optimization. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, GECCO '07, pages 1187–1194, London, England. Association for Computing Machinery, 2007. ISBN: 9781595936974. DOI: 10.1145/1276958.1277190. URL: https://doi.org/10.1145/1276958.1277190 (cited on pages 32, 33).

[DT17] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017 (cited on page 69).

[Dun61] Olive Jean Dunn. Multiple Comparisons among Means. *Journal of the American Statistical Association*, 56(293):52–64, March 1961. ISSN: 0162-1459. DOI: 10.1080/01621459.1961.10482090. URL: https://www.tandfonline.com/doi/abs/10.1080/01621459.1961.10482090 (visited on 05/19/2023). Publisher: Taylor & Francis _eprint: https://www.tandfonline.com/doi/pdf/10.1080/01621459.1961.10482090 (cited on page 20).

[EBM+21] Ecenaz Erdemir, Jeffrey Bickford, Luca Melis, and Sergul Aydore. Adversarial robustness with non-uniform perturbations. *arXiv preprint arXiv:2102.12002*, 2021 (cited on pages 19, 26).

[FCR+15] Isvani Frias-Blanco, Jose del Campo-Avila, Gonzalo Ramos-Jimenez, Rafael Morales-Bueno, Agustin Ortiz-Diaz, and Yaile Caballero-Mota. Online and Non-Parametric Drift Detection Methods Based on Hoeffding's Bounds. *IEEE Transactions on Knowledge and Data Engineering*, 27(3):810–823, March 2015. ISSN: 1041-4347. DOI: 10.1109/TKDE.2014.2345382. URL: http://ieeexplore.ieee.org/document/6871418/ (visited on 05/02/2023) (cited on pages 3, 20, 21).

[GBB14] Ao Gama, Albert Bifet, and Research Barcelona. A survey on concept drift adaptation. *ACM Comput. Surv*, 46, 2014. DOI: 10.1145/2523813. URL: http://dx.doi.org/10.1145/2523813 (visited on 03/02/2022) (cited on page 95).

[GCG+20] Salah Ghamizi, Maxime Cordy, Martin Gubri, Mike Papadakis, Andrey Boystov, Yves Le Traon, and Anne Goujon. Search-based adversarial testing and improvement of constrained credit scoring systems. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1089–1100, 2020 (cited on pages 2–4, 10, 16, 17, 19, 26, 28, 36, 44, 46, 66, 117).

[GCP+21] Salah Ghamizi, Maxime Cordy, Mike Papadakis, and Yves Le Traon. Adversarial robustness in multi-task learning: promises and illusions. *arXiv preprint arXiv:2110.15053*, 2021 (cited on page 36).

[Geo18] Nathan George. Lending club loan data. https://www.kaggle.com/datasets/wordsforthewise/lending-club, 2018 (cited on pages 47, 72, 86, 126, 137).

[GGB20] Maciej Grzenda, Heitor Murilo Gomes, and Albert Bifet. Delayed labelling evaluation for data streams. *Data Mining and Knowledge Discovery*, 34(5):1237–1266, 2020. ISSN: 1573756X. DOI: 10.1007/s10618-019-00654-y. URL: https://doi.org/10.1007/s10618-019-00654-y. Publisher: Springer US (cited on page 22).

[GHS+21] Gilad Gressel, Niranjan Hegde, Archana Sreekumar, Rishikumar Radhakrishnan, Kalyani Harikumar, Krishnashree Achuthan, et al. Feature importance guided attack: a model agnostic adversarial attack. *arXiv preprint arXiv:2106.14815*, 2021 (cited on page 18).

[GMC+04] João Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. Learning with drift detection. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 3171(May 2014):286–295, 2004. ISSN: 16113349. DOI: 10.1007/978-3-540-28645-5_29. ISBN: 3540232370 (cited on pages 3, 20, 21).

[GPS23] Joshua P Gardner, Zoran Popovi, and Ludwig Schmidt. Benchmarking distribution shift in tabular data with tableshift. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023. URL: https://openreview.net/forum?id=XYxNklOMMX (cited on page 84).

[GSS15] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015 (cited on pages 3, 12, 19).

[Gur22] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2022. https://www.gurobi.com/wp-content/plugins/hd_documentations/documentation/9.5/refman.pdf (cited on page 29).

[Har99] Michael Bonnell Harries. Splice-2 comparative evaluation: electricity pricing. In 1999. URL: https://api.semanticscholar.org/CorpusID:151207670 (cited on pages 106, 163).

[HGX+23] Qiang Hu, Yuejun Guo, Xiaofei Xie, Maxime Cordy, Lei Ma, Mike Papadakis, and Yves Le Traon. Aries: Efficient Testing of Deep Neural Networks via Labeling-Free Accuracy Estimation, February 2023. DOI: 10.48550/arXiv.2207.10942. URL: http://arxiv.org/abs/2207.10942 (visited on 05/02/2023). arXiv:2207.10942 [cs] (cited on pages 20, 21).

[HK20] John T Hancock and Taghi M Khoshgoftaar. Survey on categorical data for neural networks. *Journal of Big Data*, 7:1–41, 2020 (cited on page 44).

[HKC+20] Xin Huang, Ashish Khetan, Milan Cvitkovic, and Zohar Karnin. Tabtransformer: tabular data modeling using contextual embeddings. *arXiv preprint arXiv:2012.06678*, 2020 (cited on pages 16, 48, 72, 79, 86, 129, 140).

[HY21] Abdelhakim Hannousse and Salima Yahiouche. Towards benchmark datasets for machine learning based website phishing detection: an experimental study. *Engineering Applications of Artificial Intelligence*, 104:104347, 2021 (cited on pages 34, 47, 72, 86, 117–119, 126, 128, 137–139).

[Inc22] Evidently AI Inc. Evidently ai: data drift algorithm, version 0.2.1, December 7, 2022. URL: https://github.com/evidentlyai/evidently (cited on pages 3, 20).

[IPG+18] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018 (cited on page 19).

[Kag19] Kaggle. All Lending Club loan data, 2019. URL: https://www.kaggle.com/wordsforthewise/lending-club (visited on 01/13/2022) (cited on pages 33, 106, 163).

[KGB16] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016 (cited on pages 29, 69).

[KHS+18] Bogdan Kulynych, Jamie Hayes, Nikita Samarin, and Carmela Troncoso. Evading classifiers in discrete domains with provable optimality guarantees. *arXiv preprint arXiv:1810.10939*, 2018 (cited on pages 18, 55).

[KKT22] Klim Kireev, Bogdan Kulynych, and Carmela Troncoso. Adversarial robustness for tabular data through cost and utility awareness. *arXiv preprint arXiv:2208.13058*, 2022 (cited on pages 10, 17, 89, 111).

[KKT23] Klim Kireev, Bogdan Kulynych, and Carmela Troncoso. Adversarial robustness for tabular data through cost and utility awareness. In *The Second Workshop on New Frontiers in Adversarial Machine Learning*, 2023 (cited on pages 18, 55).

[KVD+14] Shankar Krishnan, Suresh Venkatasubramanian, Tamraparni Dasu, and Ke Yi. An Information-Theoretic Approach to Detecting Changes in MultiDimensional Data Streams An Information-Theoretic Approach to Detecting Changes in Multi-Dimensional Data Streams, 2014. URL: https://www.researchgate.net/publication/248542520 (visited on 02/15/2022) (cited on pages 3, 21).

[LGX+22] Teng Long, Qi Gao, Lili Xu, and Zhangbing Zhou. A survey on adversarial attacks in computer vision: taxonomy, visualization and future directions. *Computers & Security*, 121:102847, 2022. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2022.102847. URL: https://www.sciencedirect.com/science/article/pii/S0167404822002413 (cited on page 45).

[LH03] Chee Peng Lim and R.F. Harrison. Online pattern classification with multiple neural network systems: an experimental study. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 33(2):235–247, May 2003. ISSN: 1558-2442. DOI: 10.1109/TSMCC.2003.813150. Conference Name: IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) (cited on page 95).

[LKF+18] Zinan Lin, Ashish Khetan, Giulia Fanti, and Sewoong Oh. Pacgan: the power of two samples in generative adversarial networks. *Advances in neural information processing systems*, 31, 2018 (cited on page 142).

[LLD+19] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, Joao Gama, and Guangquan Zhang. Learning under Concept Drift: A Review. *IEEE Transactions on Knowledge and Data Engineering*, 31(12):2346–2363, December 2019. ISSN: 15582191. DOI: 10.1109/TKDE.2018.2876857. (Visited on 10/06/2021). arXiv: 2004.05785 Publisher: IEEE Computer Society (cited on pages 13, 92, 163).

[LLY+20] Jiangnan Li, Jin Young Lee, Yingyuan Yang, Jinyuan Stella Sun, and Kevin Tomsovic. Conaml: constrained adversarial machine learning for cyber-physical systems. *arXiv preprint arXiv:2003.05631*, 2020 (cited on pages 18, 26, 28).

[LQB+23] Tennison Liu, Zhaozhi Qian, Jeroen Berrevoets, and Mihaela van der Schaar. GOGGLE: generative modelling for tabular data by learning relational structure. In *The Eleventh International Conference on Learning Representations*, 2023. URL: https://openreview.net/forum?id=fPVRcJqspu (cited on pages 70, 87, 142).

[LRG+20] Meredith Lee, Jesse Raffa, Marzyeh Ghassemi, Tom Pollard, Sharada Kalanidhi, Omar Badawi, Karen Matthys, and Leo Anthony Celi. Wids (women in data science) datathon 2020: icu mortality prediction. PhysioNet, 2020 (cited on pages 47, 72, 86, 126, 128, 137, 139).

[MAT+18] Marius Mosbach, Maksym Andriushchenko, Thomas Trost, Matthias Hein, and Dietrich Klakow. Logit pairing methods can fool gradient-based attacks. *NeurIPS 2018 Workshop on Security in Machine Learning*, 2018 (cited on page 48).

[MGK+10] Mohammad Masud, Jing Gao, Latifur Khan, Jiawei Han, and Bhavani M Thuraisingham. Classification and novel class detection in concept-drifting data streams under time constraints. *IEEE Transactions on knowledge and data engineering*, 23(6):859–874, 2010 (cited on page 22).

[MH20] Gautam Raj Mode and Khaza Anuarul Hoque. Crafting adversarial examples for deep learning based prognostics (extended version). *arXiv preprint arXiv:2009.10149*, 2020 (cited on page 26).

[MLK+22] Yael Mathov, Eden Levy, Ziv Katzir, Asaf Shabtai, and Yuval Elovici. Not all datasets are born equal: on heterogeneous tabular data and adversarial examples. *Knowledge-Based Systems*, 242:108377, 2022 (cited on pages 17, 18).

[MMS+17] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017 (cited on pages 5, 6, 12, 19, 47, 59, 68, 69).

[NST+18] Maria-Irina Nicolae, Mathieu Sinn, Minh Ngoc Tran, Beat Buesser, Ambrish Rawat, Martin Wistuba, Valentina Zantedeschi, Nathalie Baracaldo, Bryant Chen, Heiko Ludwig, Ian Molloy, and Ben Edwards. Adversarial robustness toolbox v1.2.0. *CoRR*, 1807.01069, 2018. URL: `https://arxiv.org/pdf/1807.01069` (cited on page 119).

[PA16] Joshua Plasse and Niall Adams. Handling delayed labels in temporally evolving data streams. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 2416–2424. IEEE, 2016 (cited on page 22).

[Pag54] E. S. Page. Continuous Inspection Schemes. *Biometrika*, 41(1/2):100–115, 1954. ISSN: 0006-3444. DOI: `10.2307/2333009`. URL: `https://www.jstor.org/stable/2333009` (visited on 05/02/2023). Publisher: [Oxford University Press, Biometrika Trust] (cited on pages 3, 20, 21).

[PCvD+22] Lorena Poenaru-Olaru, Luis Cruz, Arie van Deursen, and Jan S. Rellermeyer. Are Concept Drift Detectors Reliable Alarming Systems? – A Comparative Study. en, November 2022. URL: `http://arxiv.org/abs/2211.13098` (visited on 12/19/2022). arXiv:2211.13098 [cs] (cited on pages 22, 92).

[PMG+18] Noseong Park, Mahmoud Mohammadi, Kshitij Gorde, Sushil Jajodia, Hongkyu Park, and Youngmin Kim. Data synthesis based on generative adversarial networks. *Proceedings of the VLDB Endowment*, 11(10):1071–1083, June 2018. ISSN: 2150-8097. DOI: `10.14778/3231751.3231757`. URL: `http://dx.doi.org/10.14778/3231751.3231757` (cited on pages 70, 87).

[PMG16] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv preprint arXiv:1605.07277*, 2016 (cited on page 36).

[PPC+20] Fabio Pierazzi, Feargus Pendlebury, Jacopo Cortellazzi, and Lorenzo Cavallaro. Intriguing properties of adversarial ml attacks in the problem space, 2020. arXiv: 1911.02142 [cs.CR] (cited on pages 2, 11, 17, 26).

[PXD+19] Tianyu Pang, Kun Xu, Chao Du, Ning Chen, and Jun Zhu. Improving adversarial robustness via promoting ensemble diversity. In *International Conference on Machine Learning*, 2019. URL: https://api.semanticscholar.org/CorpusID:59291958 (cited on page 19).

[QAW+15] Abdulhakim A. Qahtan, Basma Alharbi, Suojin Wang, and Xiangliang Zhang. A PCA-Based Change Detection Framework for Multidimensional Data Streams. en. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 935–944, Sydney NSW Australia. ACM, August 2015. ISBN: 978-1-4503-3664-2. DOI: 10.1145/2783258.2783359. URL: https://dl.acm.org/doi/10.1145/2783258.2783359 (visited on 05/02/2023) (cited on pages 3, 20, 21).

[RGC+21] Sylvestre-Alvise Rebuffi, Sven Gowal, Dan Andrei Calian, Florian Stimberg, Olivia Wiles, and Timothy A Mann. Data augmentation can improve robustness. *Advances in Neural Information Processing Systems*, 34:29935–29948, 2021 (cited on pages 5, 6, 12, 19, 68).

[RHS20] Christoph Raab, Moritz Heusinger, and Frank-Michael Schleif. Reactive Soft Prototype Computing for Concept Drift Streams. *Neurocomputing*, 416, April 2020. DOI: 10.1016/j.neucom.2019.11.111 (cited on pages 3, 20, 21).

[RIZ+17] Pranav Rajpurkar, Jeremy Irvin, Kaylie Zhu, Brandon Yang, Hershel Mehta, Tony Duan, Daisy Ding, Aarti Bagul, Curtis Langlotz, Katie Shpanskaya, et al. Chexnet: radiologist-level pneumonia detection on chest x-rays with deep learning. *arXiv preprint arXiv:1711.05225*, 2017 (cited on page 2).

[RWK20] Leslie Rice, Eric Wong, and Zico Kolter. Overfitting in adversarially robust deep learning. In *International conference on machine learning*, pages 8093–8104. PMLR, 2020 (cited on pages 19, 68).

[SA21] Ravid Shwartz-Ziv and Amitai Armon. Tabular Data: Deep Learning is Not All You Need. *arXiv preprint arXiv:2106.03253*, 2021 (cited on page 16).

[SDC+24] Mihaela C Stoian, Salijona Dyrmishi, Maxime Cordy, Thomas Lukasiewicz, and Eleonora Giunchiglia. How realistic is your synthetic data? constraining deep generative models for tabular data. In *The Twelfth International Conference on Learning Representations*, 2024. URL: https://openreview.net/forum?id=tBROYsEz9G (cited on pages 70, 87, 141).

[SDG+22] 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 *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pages 1313–1319, 2022 (cited on pages 44, i).

[SGC+24] Thibault Simonetto, Salah Ghamizi, Maxime Cordy, Yves Le Traon, Clément Lefebvre, Andrey Boystov, and Anne Goujon. On the impact of industrial delays when mitigating distribution drifts: an empirical study on real-world financial systems. In *KDD 2024 Discovering Drift Phenomena in Evolving Landscape Workshop*, 2024 (cited on page i).

[SGC24a] Thibault Simonetto, Salah Ghamizi, and Maxime Cordy. Constrained adaptive attack: effective adversarial attack against deep neural networks for tabular data. In *Advances in Neural Information Processing Systems*, 2024 (cited on page i).

[SGC24b] Thibault Simonetto, Salah Ghamizi, and Maxime Cordy. Tabularbench: benchmarking adversarial robustness for tabular deep learning in real-world use-cases. In *Advances in Neural Information Processing Systems Datasets and Benchmarks Track*, 2024 (cited on page i).

[SGC24c] Thibault Simonetto, Salah Ghamizi, and Maxime Cordy. Towards adaptive attacks on constrained tabular machine learning. In *ICML 2024 Next Generation of AI Safety Workshop*, 2024 (cited on page i).

[SGK17] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *International Conference on Machine Learning*. PMLR, 2017 (cited on page 40).

[SH20] Mohammad Hossein Shaker and Eyke Hüllermeier. Aleatoric and Epistemic Uncertainty with Random Forests, January 2020. URL: http://arxiv.org/abs/2001.00893. arXiv: 2001.00893 (cited on pages 14, 20, 21).

[SJH+18] Qingquan Song, Haifeng Jin, Xiao Huang, and Xia Hu. Multi-label adversarial perturbations. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 1242–1247. IEEE, 2018 (cited on page 36).

[SL19] Meysam Sadeghi and Erik G Larsson. Physical adversarial attacks against end-to-end autoencoder communication systems. *IEEE Communications Letters*, 23:847–850, 2019 (cited on page 26).

[SPW+20] Ryan Sheatsley, Nicolas Papernot, Michael Weisman, Gunjan Verma, and Patrick McDaniel. Adversarial examples in constrained domains. *arXiv preprint arXiv:2011.01183*, 2020 (cited on pages 18, 26).

[SRR+21] Sulaiman Somani, Adam J Russak, Felix Richter, Shan Zhao, Akhil Vaid, Fayzan Chaudhry, Jessica K De Freitas, Nidhi Naik, Riccardo Miotto, Girish N Nadkarni, et al. Deep learning and the electrocardiogram: review of the current state-of-the-art. *EP Europace*, 2021 (cited on page 16).

[SRR20] Sanchari Sen, Balaraman Ravindran, and Anand Raghunathan. Empir: ensembles of mixed precision deep networks for increased robustness against adversarial attacks. *ArXiv*, abs/2004.10162, 2020. URL: `https://api.semanticscholar.org/CorpusID:213392702` (cited on page 19).

[SS18] Ira Shavitt and Eran Segal. Regularization learning networks: deep learning for tabular datasets. *Advances in Neural Information Processing Systems*, 31, 2018 (cited on pages 16, 17, 48, 72, 79, 86, 129, 140).

[SZS+13a] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013 (cited on pages 3, 26).

[SZS+13b] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks, December 2013. URL: `http://arxiv.org/abs/1312.6199` (cited on page 12).

[TCB+20] Florian Tramer, Nicholas Carlini, Wieland Brendel, and Aleksander Madry. On adaptive attacks to adversarial example defenses. *Advances in Neural Information Processing Systems*, 33:1633–1645, 2020 (cited on pages 5, 19, 60, 67, 89).

[TPS20] Martin Teuffenbach, Ewa Piatkowska, and Paul Smith. Subverting network intrusion detection: crafting adversarial examples accounting for domain-specific constraints. In *International Cross-Domain Conference for Machine Learning and Knowledge Extraction*, pages 301–320. Springer, 2020 (cited on page 19).

[TWT+20] Yunzhe Tian, Yingdi Wang, Endong Tong, Wenjia Niu, Liang Chang, Qi Alfred Chen, Gang Li, and Jiqiang Liu. Exploring data correlation between feature pairs for generating constraint-based adversarial examples. In *2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 430–437. IEEE, 2020 (cited on pages 3, 18, 19, 26, 28).

[TXZ+20] Qi Tang, Guoen Xia, Xianquan Zhang, and Feng Long. A customer churn prediction model based on xgboost and mlp. In *2020 International Conference on Computer Engineering and Application (ICCEA)*, pages 608–612. IEEE, 2020 (cited on page 16).

[UAB19] Daniele Ucci, Leonardo Aniello, and Roberto Baldoni. Survey of machine learning techniques for malware analysis. *Computers & Security*, 81:123–147, 2019 (cited on pages 118, 139).

[UMC20] Dennis Ulmer, Lotta Meijerink, and Giovanni Cinà. Trust issues: uncertainty estimation does not enable reliable ood detection on medical tabular data. In *Machine Learning for Health*, pages 341–354. PMLR, 2020 (cited on page 16).

[VC22] Gaël Varoquaux and Veronika Cheplygina. Machine learning for medical imaging: methodological failures and recommendations for the future. en. *npj Digital Medicine*, 5(1):1–8, April 2022. ISSN: 2398-6352. DOI: 10.1038/s41746-022-00592-y. URL: https://www.nature.com/articles/s41746-022-00592-y (visited on 05/19/2023). Number: 1 Publisher: Nature Publishing Group (cited on page 14).

[VDB18] Yash Vesikar, Kalyanmoy Deb, and Julian Blank. Reference point based nsga-iii for preferred solutions. In *2018 IEEE symposium series on computational intelligence (SSCI)*, pages 1587–1594. IEEE, 2018 (cited on page 31).

[VKV+22] Arnaud Van Looveren, Janis Klaise, Giovanni Vacanti, Oliver Cobb, Ashley Scillitoe, Robert Samoilescu, and Alex Athorne. Alibi detect: algorithms for outlier, adversarial and drift detection, version 0.10.4, October 21, 2022. URL: https://github.com/SeldonIO/alibi-detect (cited on pages 3, 20).

[VS19] Gunjan Verma and Ananthram Swami. Error correcting output codes improve probability estimation and adversarial robustness of deep neural networks. In *Neural Information Processing Systems*, 2019. URL: https://api.semanticscholar.org/CorpusID:202783660 (cited on page 19).

[WCP+23] Boxin Wang, Weixin Chen, Hengzhi Pei, Chulin Xie, Mintong Kang, Chenhui Zhang, Chejian Xu, Zidi Xiong, Ritik Dutta, Rylan Schaeffer, Sang T. Truong, Simran Arora, Mantas Mazeika, Dan Hendrycks, Zinan Lin, Yu Cheng, Sanmi Koyejo, Dawn Song, and Bo Li. Decodingtrust: a comprehensive assessment of trustworthiness in GPT models. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023. URL: https://openreview.net/forum?id=kaHpo8OZw2 (cited on pages 66, 84).

[WHB+20] 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 *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1415–1425, 2020 (cited on pages 18, 44, 66, 84).

[WHC+16] Geoffrey I. Webb, Roy Hyde, Hong Cao, Hai Long Nguyen, and Francois Petitjean. Characterizing concept drift. *Data Mining and Knowledge Discovery*, 30(4):964–994, July 2016. ISSN: 1573756X. DOI: 10.1007/S10618-015-0448-4. (Visited on 03/02/2022). arXiv: 1511.03816 Publisher: Springer New York LLC (cited on page 92).

[WK96] Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Machine learning*, 23:69–101, 1996 (cited on page 4).

[XHR+23] Han Xu, Pengfei He, Jie Ren, Yuxuan Wan, Zitao Liu, Hui Liu, and Jiliang Tang. Probabilistic categorical adversarial attack and adversarial training. In *International Conference on Machine Learning*, pages 38428–38442. PMLR, 2023 (cited on pages 18, 44, 66, 84).

[XSC+19a] Lei Xu, Maria Skoularidou, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. Modeling tabular data using conditional gan, 2019. arXiv: 1907.00503 [cs.LG] (cited on page 87).

[XSC+19b] Lei Xu, Maria Skoularidou, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. Modeling tabular data using conditional GAN. In *Advances in Neural Information Processing Systems*, volume 33, 2019 (cited on page 70).

[XZZ19] Chang Xiao, Peilin Zhong, and Changxi Zheng. Enhancing adversarial defense by k-winners-take-all. *arXiv: Learning*, 2019. URL: https://api.semanticscholar.org/CorpusID:209954233 (cited on page 19).

[YAY20] Noam Yefet, Uri Alon, and Eran Yahav. Adversarial examples for models of code. *Proceedings of the ACM on Programming Languages*, 4:1–30, 2020 (cited on page 26).

[YHG+19] Tao Yu, Shengyuan Hu, Chuan Guo, Wei-Lun Chao, and Kilian Q. Weinberger. A new defense against adversarial images: turning a weakness into a strength. In *Neural Information Processing Systems*, 2019. URL: https://api.semanticscholar.org/CorpusID:202786467 (cited on page 19).

[YHO+19] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6023–6032, 2019 (cited on pages 69, 87).

[YKR19] Xuwang Yin, Soheil Kolouri, and Gustavo Kunde Rohde. Adversarial example detection and classification with asymmetrical adversarial training. *arXiv: Learning*, 2019. URL: https://api.semanticscholar.org/CorpusID:211259530 (cited on page 19).

[YLN+20] Yutaro Yamada, Ofir Lindenbaum, Sahand Negahban, and Yuval Kluger. Feature selection using stochastic gates. In *Proceedings of Machine Learning and Systems 2020*, pages 8952–8963. 2020 (cited on pages 17, 48, 72, 79, 86, 129, 140).

[YZJ+20] Jinsung Yoon, Yao Zhang, James Jordon, and Mihaela van der Schaar. Vime: extending the success of self-and semi-supervised learning to tabular domain. *Advances in Neural Information Processing Systems*, 33:11033–11043, 2020 (cited on pages 16, 48, 72, 79, 86, 129, 141).

[YZK+19] Yuzhe Yang, Guo Zhang, Dina Katabi, and Zhi Xu. Me-net: towards effective adversarial robustness with matrix estimation. *ArXiv*, abs/1905.11971, 2019. URL: https://api.semanticscholar.org/CorpusID:167217608 (cited on page 19).

[ZCD+17] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. Mixup: beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017 (cited on page 69).

[Žli10] Indré Žliobaité. Change with Delayed Labeling: When is it Detectable? In *2010 IEEE International Conference on Data Mining Workshops*, pages 843–850, December 2010. DOI: 10.1109/ICDMW.2010.49. ISSN: 2375-9259 (cited on page 22).

[ZLQ+22] Kaiyang Zhou, Ziwei Liu, Yu Qiao, Tao Xiang, and Chen Change Loy. Domain generalization: a survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*:1–20, 2022. ISSN: 1939-3539. DOI: 10.1109/tpami.2022.3195549. URL: http://dx.doi.org/10.1109/TPAMI.2022.3195549 (cited on page 22).

[ZYS+19] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep learning based recommender system: a survey and new perspectives. *ACM Computing Surveys (CSUR)*, 52(1):1–38, 2019 (cited on page 16).