Search-Based Adversarial Testing and Improvement of Constrained Credit Scoring Systems

Salah Ghamizi, Maxime Cordy, Martin Gubri,
Andrey Boystov, Mike Papadakis, Yves Le Traon
SnT, University of Luxembourg
email:{firstname.lastname}@uni.lu

Anne Goujon
BGL - BNP Parisbas
anne.goujon@bgl.lu

ABSTRACT

Credit scoring systems are critical FinTech applications that concern the analysis of the creditworthiness of a person or organization. While decisions were previously based on human expertise, they are now increasingly relying on data analysis and machine learning. In this paper, we assess the ability of state-of-the-art adversarial machine learning to craft attacks on a real-world credit scoring system. Interestingly, we find that, while these techniques can generate large numbers of adversarial data, these are practically useless as they all violate domain-specific constraints. In other words, the generated examples are all false positives as they cannot occur in practice. To circumvent this limitation, we propose CoEvA2, a search-based method that generates valid adversarial examples (satisfying the domain constraints). CoEvA2 utilizes multi-objective search in order to simultaneously handle constraints, perform the attack and maximize the overdraft amount requested. We evaluate CoEvA2 on a major bank's real-world system by checking its ability to craft valid attacks. CoEvA2 generates thousands of valid adversarial examples, revealing a high risk for the banking system. Fortunately, by improving the system through adversarial training (based on the produced examples), we increase its robustness and make our attack fail.

KEYWORDS

Search-based, Adversarial attacks, FinTech, Random Forest

ACM Reference Format:
Salah Ghamizi, Maxime Cordy, Martin Gubri, Andrey Boystov, Mike Papadakis, Yves Le Traon and Anne Goujon. 2018. Search-Based Adversarial Testing and Improvement of Constrained Credit Scoring Systems. In *Proceedings of The 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2020).* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

The banking industry increasingly relies on machine learning to support decision making based on customers' historical data. One prominent application case is *credit scoring*, i.e., "a set of decision models and their underlying techniques that aid credit lenders in the granting of credit" [24]. By learning from history – credit cases and their outcomes (whether the credit was returned in time) – supervised models can automate the approval and rejection of new credit requests with limited human intervention.

Our industrial partner, the Data Science Lab of BGL-BNP Paribas Luxembourg (henceforth referred to as"BGL") has recently engineered such a credit scoring system. Their system deals with the approval of overdraft requests, which occur when a transaction causes the balance of the account to drop below zero. Then, it is up to the bank employees to allow or reject this transaction. BGL implemented an automated system relying on random forests. That is, the approval of overdraft requests is seen as a binary classification problem (approved or rejected). The system approves or rejects overdrafts automatically, based on data about the requested transaction and the customers' history. If the overdraft is rejected by the system, an expert re-analyzes the request and may overrule the decision. If it is accepted, the system later checks whether the overdraft has been reimbursed in time.

The first challenges faced by our partner were feature engineering and model selection. As these have been widely researched (see, e.g., [8, 10, 22]), they benefited from the available body of knowledge and techniques to build a quality system that achieved a test accuracy of 80%. As of now, this system has processed more than 400,000 overdraft requests over a span of 30 months.

Yet, the stringent security requirements forced upon the banking sector oblige them to protect their credit scoring system against malicious third parties. In our partner's context, the threat lies in the capability of the third-party to modify the requested credits and the profile of customers to make the system accept overdrafts that it should have rejected.

In machine learning, such malicious inputs are called *adversarial examples* and are crafted by altering benign inputs in such a way that they fool the classification system. Adversarial examples are mainly studied in the context of computer vision and deep neural networks [1, 6, 23], where elusive alterations to the pixels of images cause misclassifications. Such research has shown that adversarial examples can be crafted by a systematic procedure – the *adversarial attack* – which typically utilizes information about the neural network's gradients to find the slightest perturbation that would change the output class.

Interestingly, the application of adversarial attacks to FinTech and random forests remains largely unexplored [19]. This is surprising given the widespread use of these techniques in industrial applications. To our knowledge, the state-of-the-art attack for random forest classification algorithms is the one designed by Papernot et al. [20]. It consists of a stochastic procedure that visits and attempts to flip the individual decision nodes of the forest's trees until the

ESEC/FSE 2020, 8 - 13 November, 2020, Sacramento, California, United States
© 2018 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/1122445.1122456

classification outcome is changed. An alternative approach could be to built a "surrogate" deep neural network (using the training data), based on which we could apply a prominent gradient-based attack (with the hope that this attack will be transferable to the random forest model).

Nevertheless, all adversarial attack techniques lean on the internal computations of the classification models and disregard the fact that altering the original input may produce false positives, i.e., *infeasible* in the real world, or *invalid* for the software system inputs that are acceptable by the classification model. While this phenomenon is less likely to occur in image recognition, where slightly altering an image can easily produce a valid image, application domains such as FinTech are subject to hard domain constraints delimiting the set of valid inputs. For instance, a credit scoring system relies on financial information such as customers' account balance, contracted credits, monthly income, and indebtment rate. Such data are naturally constrained (e.g., income is positive), interdependent (indebtment rate depends on contracted credits and monthly income) or bounded (e.g., the maximum overdraft amount authorized by the bank). Thus, any successful attack should respect these domain constraints and produce examples that satisfy them.

Moreover, we conduct experiments with the current state-of-the-art, i.e., the Papernot attack, on our partner's system.[1] Interestingly, we show that while the attack successfully generated adversarial examples that flipped the classification results for 75% of the cases (its *gross* success rate), none of them satisfied the domain constraints. This means that the attack has an *actual* success rate of 0%. These results indicate that state-of-the-art adversarial attacks cannot generate domain-constrained test inputs.

Dealing with domain constraints is a recurrent problem in software engineering [2]. In the case of generating adversarial examples, one cannot handle/satisfy the domain constraints independently of the attack technique. The issue is that on top of the constraints (many of which are imposed by other systems/components), one needs to craft the attacks and fulfil some additional objectives (e.g. cause misclassification, maximize the overdraft amount). Therefore, reducing the problem to constraint satisfaction is not enough.

To deal with this issue, we propose a search-based method that generates constrained adversarial examples for banking applications. We formulate the generation of adversarial examples satisfying the domain constraints as a multi-objective search problem and show that search-based techniques offer suitable solutions. Our method, called Constrained Evolutionary Adversarial Attack (CoEvA2), operates in a grey-box way; it relies on the feature representation of the inputs but is independent of the internal parameters of the classification model.

We apply CoEvA2 to BGL's credit scoring system and show that it can generate thirteen thousand of valid adversarial examples from 8.45% of the real overdrafts. This drastically improves over state-of-the-art adversarial attacks, which failed completely. Then, we show that we can make our partner's system more robust by performing adversarial training (i.e., retrain the model using the produced adversarial examples). After such training, the system resists to our attack (applied under similar conditions).

In summary, the contributions of this paper are:

[1]We report on these experiments in Section 5.

- We demonstrate the need for domain-constrained adversarial attack techniques for industrial financial systems. We also show that existing attacks are inapplicable to real-world credit scoring systems, such as the one of our BGL.
- We develop CoEvA2, a new adversarial attack method (for random forest applications) based on multi-objective search. Given a classification model and domain constraints, CoEvA2 effectively generates valid adversarial examples.
- We evaluate CoEvA2 on our partner's system and empirically show that it can craft adversarial examples with an actual success rate of 8.45%, leading to thousands of examples.
- We demonstrate that our method helps to improve the system's robustness (to adversarial attacks). Indeed, retraining the system on adversarial examples results in improving its robustness significantly.

2 RELATED WORK

2.1 Credit Scoring

The study of Louzada et al. [14] presents a comprehensive survey of classification methods in the context of credit scoring automation. While focusing on classification models, Louzada et al. also reported the different problems tackled by the surveyed papers, with none of them been related to model robustness or adversarial attacks.

Much research has been conducted on feature engineering and model selection for credit scoring. For example, De Melo and Banzhaf [8] combined Kaizen programming and logistic regression to find the best non-linear combination of features. Saia et al. [22] proposed a wavelet-based feature engineering method and evaluated its performance using multiple types of models. Feng et al. [10] proposed a feature selection approach based on filters and a novel index named *new separation degree*. These techniques are orthogonal to ours as they do not target adversarial attacks or robustness.

2.2 Adversarial Examples

Adversarial examples were first mentioned by the studies of Biggio et al. [5] and Szegedy et al. [23] in the context of deep neural networks for image classification. Their intriguing property resides in the small perturbations needed to change the predicted label.

According to Biggio et al. [5] white-box attacks assume perfect knowledge on the model, its parameters, training set, and features. Grey-box attacks use some knowledge about the targeted system but assume another part to be unknown. Black-box attacks rely on the (raw) input space and the system's outputs to generate adversarial examples.

In the recent years, the ever-increasing literature has studied adversarial examples, mainly for computer vision (e.g., [1, 6]) with applications to autonomous cars and facial recognition and, to a lesser extent, natural language processing [4] and software security [6, 12].

Papernot et al. [19] mention the potential threats of adversarial examples for financial fraud detection. Yet, to the best of our knowledge, there exists no prior work applying adversarial attacks to industrial systems from the financial domain.

In another paper [20], Papernot et al. present an attack to decision trees. While this attack is straightforward to extend to random forests, it does not support domain constraints. As we show later, this makes it incapable of generating adversarial examples satisfying the constraints.

Kantchelian et al. [13] have also proposed another random forest attack. It transforms the decision nodes to formula, forming them as misclassification objective, and uses a SAT solver to generate solutions. Thus, any solution corresponds to an adversarial example. While this attack can theoretically solve the problem of generating constrained adversarial examples (by adding constraints into the formula), in practice, it faces scalability issues due to the inherent problems and limitations of the SAT solvers.

Indeed, we conducted an exploratory experiment based on the HELOC dataset[2] which has half the number of features compared to our partner's dataset and simple constraints involving at most two features. After 20 hours, the Kantchelian attack could not generate any adversarial example satisfying the constraints.

Our method overcomes the limitations of state-of-the-art attacks and designs a search-based (evolutionary) algorithm to generate adversarial examples that cause misclassification and satisfy the domain constraints while minimizing the perturbation and maximizing the business impact (i.e., the accepted overdraft amount).

The idea of using search-based algorithms to perform adversarial attack is not new. Alzantot et al.[3] have proposed a black-box attack on image recognition models (viz. deep neural networks). Being focused on images, the problem they tackle is different and does not involve domain constraints.

2.3 Constrained Test Generation

The problem of generating test inputs under domain constraints is not new [17] and was tackled by several works in the context of traditional (code-based) software, as witnessed by the survey of McMinn [16]. More recently, Ali et al. [2] evaluate different search-based methods in generating test inputs satisfying OCL constraints. In the context of Combinatorial Interaction Testing (CIT), Garvin et al [11] propose to reorganize the search space of metaheuristics to reflect the structure of the CIT problems and their inherent constraints. Compared to such works, the novelty of our research is that it targets machine learning systems under adversarial settings.

3 INDUSTRIAL CREDIT SCORING SYSTEM

3.1 Process and Datasets

When a customer initiates, through any channel, a transaction whose amount exceeds the customer's account balance, the payment engine asks the credit scoring system (CSS) for permission. The CSS examines the customer's profile and either approves the credit overdraft or it suggests the operator reject the request. In the latter case, the operator can follow the suggestion of the CSS or overrule it and accept the request.

To make informed decisions, the CSS pulls information from a dozen sources. In addition to basic features like the transaction amount and the customer's current balance, much information about the customer's history is consolidated. In the end, an overdraft request is represented as a vector of 46 features.

After approving an overdraft request, the bank expects the customer to return the credit in due time. In case the customer does not do so, the bank considers that it was wrong to allow the overdraft; otherwise it considers that it was correct. Through this post-analysis, we can associate each approved overdraft credit with a binary label (true or false). Such labels form the *ground truth* and are used to assess the accuracy of the CSS. A similar process is used for rejected overdraft and analyzes, based on the customer's future transactions, whether the overdraft credit would have been returned in time should it have been approved.

Overall, the CSS dataset comprises 400,000 overdraft credit requests with their associated label, out of which 275,000 are used for training and 125,000 for testing.

3.2 Model Requirements and Characteristics

The rationale behind our partner's project is to reduce human intervention in overdraft approval by automatically approving safe overdraft requests (sending only rejected overdraft to human experts) while minimizing the acceptance of risky overdrafts (e.g. transactions of large amount). Our partner also expects the system to run online, in real-time, and efficiently so that it does not compromise the efficiency of the other services. Finally, the selected model should be *interpretable*, as explaining hardly-interpretable models can be inefficient and even dangerous in high-stake decision-making processes [21] such as overdraft approval.

To satisfy those requirements, our partner performed feature engineering in close collaboration with business experts. They performed model selection (considering decision trees, random forest and gradient boosted trees) and used grid search to find optimal model parameters. AUC for ROC curve was used as an optimization criterion for the grid search, while F1 score was the criterion to choose the optimal classification threshold. The final model is a random forest with 500 estimators up to 8-level deep.

This model is built and integrated within a Dataiku DSS pipeline [3]. It achieves acceptable performance: 0.99 AUC and 0.99 accuracy on the training set; 0.88 AUC, 0.80 accuracy and 0.70 F1-score on the test set.

4 PROBLEM FORMULATION

4.1 Unconstrained Adversarial Attack

Let $f(.)$ be a binary classification model defined over a input space I. For simplicity, assume I to be normalized such that $I = [0..1]^m$ and $f(i) \in \{1, 0\}$ for any $i \in I$. Let $\mathbf{x_0} \in I$ represents an original example correctly classified by $f(.)$.

Adversarial attacks generate altered inputs that are close to their original counterparts, yet are misclassified by the model. In traditional, unconstrained adversarial attacks, the ideal adversarial example $\mathbf{x}^*$ crafted from $\mathbf{x_0}$ to fool $f(.)$ is defined as:

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmin}} \|\mathbf{x} - \mathbf{x_0}\|_p$$

such that

$$f(\mathbf{x}) = 1 - f(\mathbf{x_0})$$
$$\{\mathbf{x}, \mathbf{x_0}\} \subset I$$

and where $\|.\|_p$ is the L_p norm (e.g. L_2).

[2]https://community.fico.com/s/explainable-machine-learning-challenge

[3]https://www.dataiku.com/

The p-norm distance between a perturbed input and an initial one is a good first indication of the effort required to generate the adversarial example. However, to be acceptable, the perturbed input has to satisfy inherent domain constraints. This is a fundamental difference with image recognition, where it is generally admitted that a small distance between $\mathbf{x}$ and $\mathbf{x_0}$ ensures that $\mathbf{x}$ has a strong perceptual similarity to $\mathbf{x_0}$ and, thus, constitutes a valid image.

Therefore, the problem of generating adversarial attacks for ML-based FinTech systems takes a different form: both the original and the adversarial examples must be part of the subspace of inputs that are considered valid. To characterize this subspace, we proceed by first eliciting the different domain constraints.

4.2 Formalization of the Constraints

A first validity criterion demands that the adversarial example still represents an overdraft, that is, the transaction amount remains above the current balance of the customer's account. Additionally, we consider this amount relevant if it is higher than 1,000.00 currency units. Features[4] can also be interdependent. For instance, the indebtment rate must be positive and is obtained by dividing the monthly credit reimbursement by the monthly income. There also exist categorical features that can only take values from a finite set. For example, each customer can be associated with a personal level of risk (e.g. on a 1–10 scale) based on its profile and past interviews with the bank. The corresponding feature can only take as value any integer between 1 and 10.

This highlights the types of constraints that our method must support: features can be bounded, each by a different bound, some may only take certain values, and there may exist numerical dependencies between them. Accordingly, we define that a formula ϕ encoding such constraints (i.e. a constraint formula) over a set F of features is formed according to the following grammar:

$$\begin{aligned}\phi &:= \phi_1 \wedge \phi_2 \;||\; f \geq \psi \mid f \in \{c_1 \dots c_k\}\\ \psi &:= c \mid f \mid \psi_1 \oplus \psi_2\end{aligned}$$

where $f \in F$; $c, c_1, \dots, c_k$ are constant values; ϕ, ϕ_1, ϕ_2 are constraint formulae; $\geq \in \{<, \leq, =, \neq, \geq, >\}$; ψ, ψ_1, ψ_2 are numerical formulae and $\oplus \in \{+, -, *, /\}$.

In addition to satisfying such formula, an adversarial attack may not be able to modify some features. For instance, the level of risk associated to a customer is under control of the bank and cannot be changed by the customer himself. The same holds for features resulting from the aggregation of data over time. Thus, we enforce the requirement that the attack can only alter the subset $\mathcal{F} \subseteq F$ of *mutable* features. The other features (which the attack cannot alter) are *immutable*. In BGL's CSS, 16 features are mutable and the other 30 are immutable. This means that the attacker's capability to succeed strongly depends on the 30 features it cannot change. We thus consider the features of different customers as different starting points for our search algorithm.

[4]Due to NDA we cannot reveal the exact features used. The examples of feature we provide are different from the ones used by our partner. However, their interrelations are of the same level of complexity.

4.3 Constrained Adversarial Attack

Let $I = [0..1]^m$ be the feature vector space over the feature set $F = \{f_1 \dots f_m\}$, $\mathcal{F} \subseteq F$ be the set of mutable features and ϕ be the formula over F encoding the domain constraints. Furthermore, let I_ϕ denote the subspace of valid feature vector, i.e. $I_\phi = \{i \in I : i \models \phi\}$.

Given a binary classification model $f(.)$ and an original input $\mathbf{x_0} = \{(\mathbf{x_0})_1, \dots (\mathbf{x_0})_m\} \in I_\phi$, the ideal adversarial example $\mathbf{x}^*$ generated from $\mathbf{x_0}$ to fool $f(.)$ is defined as

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmin}} \|\mathbf{x} - \mathbf{x_0}\|_p$$

such that

$$\begin{aligned}&f(\mathbf{x}) = 1 - f(\mathbf{x_0})\\ &\{\mathbf{x}, \mathbf{x_0}\} \subset I_\phi\\ &f_i \notin \mathcal{F} \Rightarrow (\mathbf{x_0})_i = (\mathbf{x})_i, \forall 1 \leq i \leq m.\end{aligned}$$

Here, the difficulty of performing such attack lies in that it can only alter features in $\mathcal{F}$ and in a way that ϕ remains satisfied.

5 MOTIVATION: HOW HELPFUL ARE EXISTING ATTACK TECHNIQUES?

We start our study by assessing the capability of existing (unconstrained) attacks to generate valid adversarial examples in our real-world use case. We assess the gross success rate of these attacks (percentage of times they manage to create an example misclassified by the model), their actual success rate (after removing the examples that do not satisfy the domain constraints) and the average amount of perturbation applied (measured as the L_2 distance to the original input). The amount of perturbation is meant to serve as a metric comparison between the attacks.

5.1 Random Forest Attack

First, we consider the attack proposed by Papernot et al. [20] – henceforth named the *Papernot attack* – which was originally designed to cause misclassifications in decision trees by visiting all nodes in the tree and making them flip until the misclassification is achieved.

This is the only attack relevant to our case. So, we adapt it (to random forests) by iteratively applying the Papernot attack to every tree of the forest until the classification outcome of the random forest changes. We call this method *Iterative Papernot*.

We evaluated *Iterative Papernot* on all original test inputs of our use case where the model makes correct classifications. The results are recorded in Table 1 and reveal that the attack seems successful, as it manages to generate adversarial examples (causing misclassification) in 74.86% of our starting points/inputs, with an average L_2 distance (to the corresponding original inputs) of 10.64. However, it turned out that none of the generated inputs satisfied the domain constraints, leading to an actual success rate of 0%.

5.2 Gradient-Based Attacks

Another popular family of adversarial attacks are the gradient-based attacks. These attacks were designed to generate adversarial examples on Deep Neural Networks (DNNs). We note that during learning, a DNN iteratively adjusts its neurons' weight according to the gradient of its cost function (which depends on the weights).

Table 1: Success rates and average perturbation produced by existing adversarial attacks applied on our partner's system. While every method manages to generate adversarial examples, none of these satisfy the domain constrains.

Attack	Gross success rate	Actual success rate	Avg L_2
Papernot	74.86%	**0.00%**	10.64
PGD	17.30%	**0.00%**	0.10
CW2	80.00%	**0.00%**	0.37

Gradient-based attacks exploit the same information to produce a perturbation that changes the output of the last neuron layer, thereby changing the classification outcome.

Being gradient-based, those methods can apply only on models relying on differentiable cost functions. Thus, they do not work *out of the box* on random forests.

A common way to circumvent this limitation is to build a *surrogate* model (a DNN) that mimics the random forest. That is, we train this DNN on the same input set and use the outputs (classification results) of the random forest as the ground truth for the DNN. Then, we perform the gradient-based attack on the surrogate DNN and obtain an adversarial example. The underlying assumption of this method is that any adversarial example that fools the DNN also fools the mimicked model.

For our experiments, we consider two gradient-based attacks: Projected Gradient Descent (PGD) [15] and CW2 [7], which are considered among the most effective attacks. We apply each attack on all original test inputs that the model correctly classifies. We implement a DNN model using the Tensorflow/Keras frameworks and we use the implementation of the gradient-based attacks provided by the IBM robustness library [18].

Results are shown in Table 1. PGD succeeds in generating adversarial examples (causing misclassification) in only 17.30% of the attempts, yet it does so with the smallest average amount of perturbation amongst all techniques (L_2 distance of 0.10). Nevertheless, none of the generated adversarial satisfy the domain constraints. CW2 has a much higher gross success rate (80%) at the cost of a higher perturbation than PGD (0.37), yet much lower than the Papernot attack. Like the other two methods, CW2 fails to generate a single example satisfying the constraints.

Overall, our analysis shows that, by focusing on classification method and outcome, while being unaware of the domain constraints, **state-of-the-art attacks fail to generate valid adversarial examples**. This fact demonstrates the need for new constraint-aware attacks, i.e., attacks that satisfy the constraints *by design.*

6 RESEARCH QUESTIONS

Having shown that state-of-the-art adversarial attacks are not useful in our case, we look for ways to circumvent their limitations and successfully generate valid adversarial examples. We focus more particularly on the use case where a malicious third party aims at fooling the system, i.e., making it approve overdrafts that should be rejected. This use case is deemed relevant by our partner as it induces a risk of financial loss for the bank.

To this end, we investigate whether simple methods satisfying the domain constraints can solve our problem. Thus, in our first question, we check whether altering the initial points while keeping constraints satisfied is sufficient. Hence, we ask:

RQ1 *Can we generate successful adversarial examples by just satisfying the domain constraints?*

To answer this question, we investigate two solutions. The first is to extend the Iterative Papernot attack in order to make it consider the constraints as it searches through the nodes. The second is to search for solutions (using single objective search) that satisfy the constraints. Then, we can check whether the produced examples are adversarial.

As our results shall show, these single-objective methods can craft examples that either change the classification outcome or satisfy the domain constraints, but not both at the same time. We conjecture that, on the one hand, the iterative nature of the Papernot attack blocks it into a narrow part of the landscape and, on the other hand, the random search does not benefit from the knowledge of the original input (causing arbitrary perturbation). This means that an effective search should not only be guided with additional criteria (e.g., minimize the perturbation) but also explore a diverse space.

To achieve this, we experimented with evolutionary (genetic) algorithms. Such techniques are directed by some feedback, aka fitness function, that quantifies how close the current solutions are to the sought ones. At the same time, the random alterations they apply to the candidate solutions create disruption in the search and, doing so, avoids falling into local optima.

Our definition of constrained adversarial attack (see Section 4.3) hints that such a search algorithm needs to handle multiple objectives: minimize perturbation, flip the classification, satisfy the domain constraints (changing only mutable features). Additionally, a malicious third party looks for optimizing a *domain-specific objective*: maximize the overdraft amount.

Thus, we design a genetic algorithm that handles all these constraints and objectives. We assess its performance and investigate, in particular, which fitness function (combination of objectives) performs the best. Thus, we ask:

RQ2 *How effective is our fitness function at generating constrained adversarial examples?*

We answer this question by presenting our algorithm, named *CoEvA2*, and empirically evaluating it using different variants of the fitness function.

Having shown that our method constitutes an effective attack, we aim to improve the defence mechanism of our partner's system, in order to eliminate any risk that real-world attacks succeed. Therefore, we turn our attention toward improving the robustness of the CSS. To achieve this, we used *adversarial training*, which consists of re-training the model with generated (successful) adversarial examples, together with their correct classification label. Such a practice is widely popular and has been shown to improve the robustness of machine learning models. We, therefore, use adversarial training to improve our partner's system and check the scale of this improvement. Hence, we ask:

RQ3 *How much adversarial training based on CoEvA2 can increase the robustness of the system?*

We answer this question by checking the success rate of CoEvA2 when applied on various starting points.

7 SEARCH-BASED GENERATION OF CONSTRAINED ADVERSARIAL EXAMPLES

Figure 1 displays an overview of the CoEvA2 process. Starting from a set of samples (randomly selected from the test set), we iterate over the elements of this set. At each iteration, CoEvA2 starts from the sampled element (named the *initial state*) and creates an initial population of new examples. Then, it evolves this population with the aim of finding valid adversarial examples.

7.1 Population

Since only a subset of the features are mutable (16 out of 46 in our industrial case), an adversarial example can differ from the initial state only by the value of its mutable features. Thus, given an initial state s and a feature vector space I, the **population** is a subset $P \subset I$ of feature vectors such that any **individual** $p \in P$ has the same value as s for all immutable features. We can, therefore, reduce the **genotype** of an individual as a single **chromosome**, which is the vector of its mutable features. Any **gene** is an element of this chromosome and contains the value of the corresponding mutable features.

Note that we do not require any individual to satisfy the domain constraints ϕ or to cause a misclassification. Indeed, we allow the algorithm to produce invalid and benign examples throughout the evolution process. This provides a smooth landscape for the search, allowing it to explore efficiently this large search space. Constraint satisfaction and misclassification are actually encoded into the fitness/objective functions (see Section 7.2), in a way that valid adversarial examples are considered better than invalid and benign ones. Since misclassification is one of the objective, the evaluation of the individuals makes use of the attacked model (in a black-box way, using only the output class probabilities).

7.2 Fitness Function

We formulate the generation of constrained adversarial examples as an optimization problem with four objectives. Each objective can be independently assessed through an objective function.

The first objective function f_1 models the requirements of causing misclassification, that is, maximizing the probability that the example is classified in the targeted class. It is defined as the distance between the example and the incorrect class targeted by the adversarial attack.

Without loss of generality we assume the target class is 0 (the correct class is 1). When provided with an input $\mathbf{x}$, a binary classification model outputs $p(\mathbf{x})$, the prediction probability that $\mathbf{x}$ lies in class 1. If $p(\mathbf{x})$ is above the classification threshold (a hyperparameter of the model), the model classifies it in class 1; otherwise, in class 0. Thus, we see $p(\mathbf{x})$ as the distance of $\mathbf{x}$ to class 0. By seeking an input $\mathbf{x}^*$ that minimizes this distance, we increase the likelihood of misclassification *regardless* of the actual classification threshold. Thus, we have:

$$f_1(\mathbf{x}) = p(\mathbf{x}).$$

The second objective is to minimize the amount of perturbation measured between the initial state and the adversarial example, a common requirement of adversarial attacks [5]. We use a conventional measure of this amount: the normalized L_2 distance between the two inputs. Thus, given an initial state $\mathbf{x_0}$, the distance from an example $\mathbf{x}$ and $\mathbf{x_0}$ is given by

$$f_2(\mathbf{x}) = L_2(\mathbf{x}, \mathbf{x0}).$$

The third objective is the actual domain objective, that is, maximizing the approved overdraft credit amount. By convenience, we transform this objective into a (normalized) minimization problem. Let $\mathbf{x}$ be an example and $(\mathbf{x})_t$ be the value of the feature encoding the requested overdraft amount. Then, the objective function f_3 can be defined as

$$f_3(\mathbf{x}) = \frac{1}{(\mathbf{x})_t}$$

Thus, this objective considers that the most successful solution is the one that reaches the highest overdraft amount. In practice, though, our partner (like most banks) specifies a maximal overdraft amount above which the transaction is always rejected.

The fourth and last objective concerns the satisfaction of the domain constraints. As mentioned, we allow individuals to violate the constraints as the evolution progresses. Yet, to converge towards valid adversarial examples, we transform the satisfaction of each (numerical) constraint into a (normalized) *penalty* function to minimize, representing how far an example $\mathbf{x}$ is from satisfying the constraint. More precisely, we transform each constraint into an inequality of the form of $C(X) \geq 0$ (e.g. $3f \geq g$ yields $3f - g \geq 0$). If the constraint is not satisfied, $C(X) < 0$ and we use the absolute value of C(X) as distance. The overall distance to constraint satisfaction is the mean of the normalized individual distances.

Thus, assuming $\phi = \bigwedge_{i=1..k} \phi_i$, the fourth objective function is defined as:

$$f_4(\mathbf{x}) = \frac{1}{k} \sum_{\phi_i} penalty(\mathbf{x}, \phi_i).$$

In our implementation, the transformation of the constraints into these penalty functions is automatically handled by the framework we use (see more in Section 8). Other heuristics to compute such distance to satisfaction exist [16, 17] and could be considered in future work.

Overall, we consider that the success of an adversarial example can be measured by the trade-off between the likelihood of flipping the classification outcome, the applied perturbation, the overdraft amount and the satisfaction of the constraints. To objectively quantify this trade-off, we define our fitness function as a linear equation over the four objective functions, that is:

$$fitness(\mathbf{x}) = \alpha \times f_1(\mathbf{x}) + \beta \times f_2(\mathbf{x}) + \gamma \times f_3(\mathbf{x}) + \delta \times f_4(\mathbf{x})$$

where $\alpha, \beta, \gamma, \delta > 0$ are meta-parameters that specify the relative importance of the four objective. Overall the search process will attempt to generate examples that minimize this fitness function and simultaneously fulfil the four objectives.

In practice, we set these meta-parameters according to our partner's requirements and experience. The rationale was to reflect the domain requirements:

- Constraints: These shape the valid input space, meaning that any non-conforming input is invalid/infeasible. It is imperative to satisfy the constraints and hence, we make them our most important objective ($\delta = 1,000$).

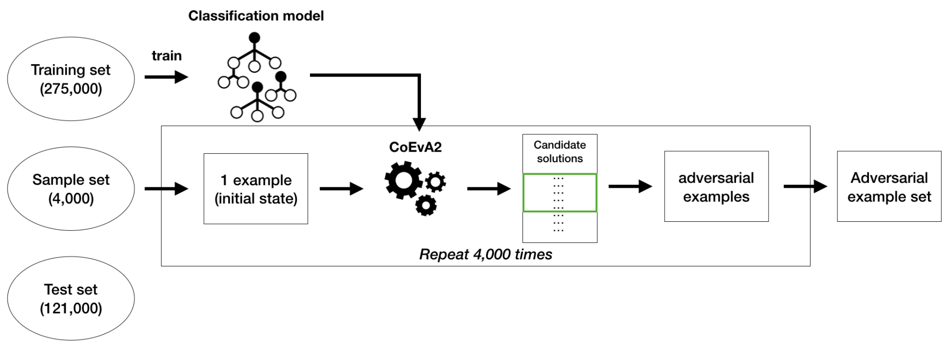


Figure 1: Overview of CoEvA2. Adversarial examples are generated from benign inputs (sampled from the test set).

```
Input: $\mathbf{x_0}$, an initial state;
       $fitness$, a fitness function;
       $N_{gen}$, a number of generations;
       $L$, a population size;
Output: A population $P$ of adversarial examples
        minimizing the $fitness$ function;
1 $P \leftarrow init(\mathbf{x_0}, L)$ ;
2 for $j = 1$ to $N_{gen}$ do
3     $P_{survive} \leftarrow binary\_tournament\_select(P, fitness)$;
4     $P_{offspring} \leftarrow SBX\_crossover(P_{survive})$;
5     $P \leftarrow P_{survive} \cup polyMutate(P_{offspring})$;
6 end
7 return $P$
```

Algorithm 1: Generation process of CoEvA2

- Maximise overdraft: For a bank, minimising the potential loss of money is of utmost importance. Indeed, the overdraft amount represents the potential gain for the attacker, which forms the objective to maximize ($\gamma = 100$).
- Cause misclassification: we also deemed it more important to cause misclassification than to minimizing perturbation ($\alpha = 2, \beta = 1$), such that the perturbation should only serve to rank adversarial examples that are successful and valid.

As revealed by our experiments, our fitness function provides a feasible and practical solution to our problem. Alternatively, we could have relied on search methods to automatically set the weights. Another option is to define four fitness functions (one per objective), thereby reducing our problem to multi-objective optimization and search for Pareto fronts. While studying these alternatives is of interest, it is unlikely that they will make major differences under such interdependent constraints.

7.3 Generation Process

Algorithm 1 formalizes the generation process of our genetic algorithm. From a given initial state $\mathbf{x_0}$, we generate an initial population P including L individuals, by randomly setting the mutable features of $\mathbf{x_0}$ (Line 1). The only constraints we enforce are the categorical constraints of the form $f \in \{c_, 1 \ldots, c_k\}$ and the boundary constraints of the form $f \geq c$ where f is a feature, $\geq \in \{<, \leq, =, \neq, \geq, >\}$, and $c, c_1, \ldots, c_k$ are constant values. This allows reducing the number of invalid examples without biasing the generation (since the boundary constraints involve only one feature each).

Then, we make the population evolve for a predefined number N_{gen} of generations (Lines 2–6). At each iteration (generation), we evaluate the fitness function of each individual of the current population P. This is achieved by, first, combining the genotype of each individual (its mutable features) with the immutable features of $\mathbf{x_0}$. Then, we can input any resulting example $\mathbf{x}$ into the *fitness* function (as defined previously) and obtain the fitness value of $\mathbf{x}$.

What follows is the application of selectors and alterers to form the next generation. We first use tournament selection that keeps the best individuals (according to the fitness function) out of samples of two (Line 3). Thus, half of the population disappear.

As for alterers, we randomly apply crossover and mutation operators. For the crossover (Line 4), we randomly pick pairs of individuals (that survived the tournament selection) and use a simulated binary crossover [9] to create two new offsprings from the numerical and categorical features of the parents. We assign the same probabilistic importance to each parent. At the end of the crossover, we obtain anew a population of size L (half parents, half offsprings).

Next, we apply mixed polynomial mutation to alter randomly the mutable features of the offspring (Line 5). Each feature has a probability p_m to be altered (set to $p_m = |\mathcal{F}|^{-1}$ in our experiments). Like the initialisation process of the population, the applied mutation operators take into account the nature (categorical/integer or real) and boundaries of each feature. At the end of the mutation process, we obtain a new population P_j to proceed in the next generation.

After the specified number of generations passed, the algorithm returns the examples of the last generation that satisfy the constraints. In addition to these individuals, the algorithm also returns the associated values of fitness and objective functions.

8 EMPIRICAL EVALUATION

8.1 Experimental Setup

To address our research questions, we implemented CoEvA2. The tool was developed in Python on top of PyMoo, an established framework for modelling and executing genetic algorithms in Python.

Our implementation is publicly available.[5] All experiments were run on our partner's internal server with about 6 cores allocated for our experiments.

We set the meta-parameters of the genetic algorithm as follows. Population size was set to 40 to maintain an acceptable computation time (CoEva2 run on 4,000 initial states takes about 24 days). Exploratory experiments showed that a higher population size does not affect our results. Also, we stop the algorithm after 10,000 generations. These numbers were found experimentally to be sufficient in making our technique to craft successful adversarial examples. For selection, mutation and crossover, we kept their default parameters which worked well in our case. During our experimentation, we performed exploratory trials with alternative settings and observed minor differences. This is in line with the study of Zamani and Hemmati [25] on the sensitivity of search-based testing methods to their hyper-parameters.

All our experiments focus on our partner's case study, i.e. generating feasible, adversarial overdraft requests approved by the CSS. To that end, we consider our partner's real-world data comprising 400,000 requests. The 275,000 were used by our partner to train the CSS's random forest. Out of the 125,000 remaining (the test set), we keep only those which are rejected overdraft requests correctly classified by the CSS. The rationale is that in realistic settings, an attacker can only manipulate future transactions and account status (which are inherently outside the training set) with the aim to make previously-rejected requests accepted by the system and, doing so, retrieving money illicitly.

This leaves us with 19,274 data points. We use two random samples of this set, each of which contains 4,000 initial states (customer account and transaction history): the first sample is used in RQ1 and RQ2 while the second is used to assess the adversarial training in RQ3. Thus, for each RQ we execute CoEvA2 4,000 times, once on each initial state. This is sufficient to rule out random effects (read more in Section8.5).

Here it must be noted, that the above settings are common to all RQs we investigate. Still the related settings required to answer each specific RQ are given at the beginning of the result Sections, i.e., those that answer RQ1 and RQ2 (Sections 8.2, 8.3, 8.4).

8.2 RQ1: Constrained Papernot and Random Search

Our first series of experiments consider (1) the Papernot attack extended to consider the domain constraint and (2) a random search that only considers the satisfaction of the constraints as objective (aka CoEvA2 with the same meta-parameters but using only f_4 as the fitness function). We regard these two attacks as baseline methods that we seek to improve.

Our extension of the Papernot attack differs from the original in three ways. First, it avoids visiting the nodes related to immutable features (thus, it never changes these features). Second, it checks the satisfaction of boundary constraints on the fly, each time a feature is altered. Third, it attempts to satisfy the other constraints by updating the dependent features.

To allow for fine-grained analysis of their results, we define four objective indicators. Each indicator reports the percentage of initial states from which a given method can produce a valid adversarial example. The objective corresponding to these indicators are:

O1: satisfy the domain constraints
O2: cause misclassification
O3: satisfy O1 and O2
O4: satisfy O3 and create a relevant overdraft (more than 1,000 currency units)

[5]URL anonymized for double-blind review

Table 2: Objective indicators of random search and constrained Papernot attacks

	Success rate	
Objective	Random search	Papernot
Constraints (O1)	**0.00%**	0.20%
Misclassification (O2)	57.15%	25.85%
O1 and O2 (O3)	**0.00%**	**0.00%**
O3 and overdraft amount (O4)	**0.00%**	**0.00%**

We evaluate the two baseline methods on a sample of 4,000 initial states (randomly picked from 19,274 rejected overdrafts). That is, we run each method 4,000 times (once per sampled initial state).

Results are shown in Table 2. Interestingly, none of the generated adversarial examples (by any of the two attacks) are valid (none of them satisfy **O4**). In the case of Papernot, a small number of the generated examples satisfy the domain constraints and about one-fourth overall cause misclassification. However, there is none that fulfil both objectives. This shows that straightforward extensions to unconstrained attacks (to make them consider the constraints) remain ineffective.

In the case of the random search, we observe that more than half of the returned examples cause misclassification. Interestingly, none of them satisfy the constraints although this is the only objective forced upon the search. A detailed investigation of the generated examples reveals that the perturbation amount ranges from 0.2 to more than 1,000. This is significantly more than the Papernot attack and the aforementioned gradient-based methods (see our preliminary study Section 5). From these observations, we hypothesize that minimizing the perturbation would allow restricting the exploration within a reasonable area around the initial state. Doing so, the search would increase the likelihood to find valid adversarial examples around this initial state (in particular, when initializing the population and performing mutation).

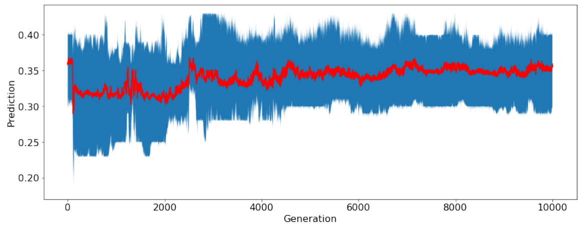
8.3 RQ2: CoEvA2 and its Fitness Function

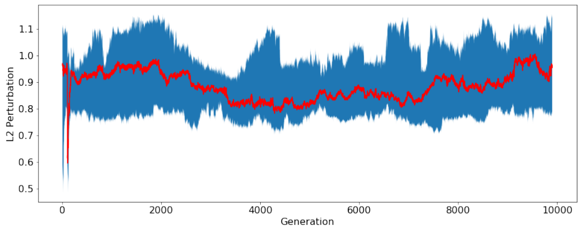
Given that the baseline methods do not generate valid adversarial examples, we implement and evaluate CoEvA2. We execute the algorithm on the same randomly-picked set of the 4,000 initial states that was used in RQ1. We also consider the same four objective indicators as in RQ1 to allow for fine-grained analysis.

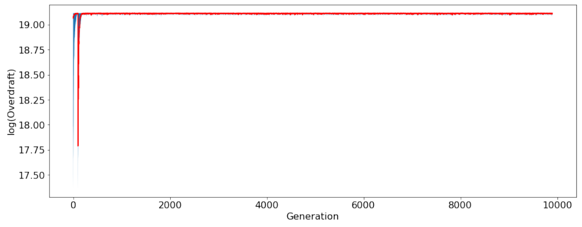
To identify and form a good fitness function, we consider multiple variants of CoEvA2, each of which uses a different subset of the objective functions. In addition to the random search guided only by the constraint satisfaction (previously studied in RQ1), we consider three variants: the full CoEvA2, another variant where only the f_2 (perturbation minimization) part is removed and another one

Table 3: Objective indicators achieved by CoEvA2, using different fitness functions.

Objective indicators	Random search (f_4)	CoEvA2 (all)	CoEvA2 (f_1, f_3, f_4)	CoEvA2 (f_1, f_2, f_4)
Constraints (O1)	0.00%	58.9%	0.00%	100.00%
Misclassification (O2)	57.15%	27.1%	31.18%	18.79%
O1 and O2 (O3)	0.00%	17.2%	0.00%	18.79%
O3 and overdraft amount (O4)	0.00%	**8.45%**	0.00%	0.00%


(a) f1: Prediction probability (lower is better)


(b) f2: Perturbation, L_2 distance (lower is better)


(c) f3: Overdraft amount (higher is better)

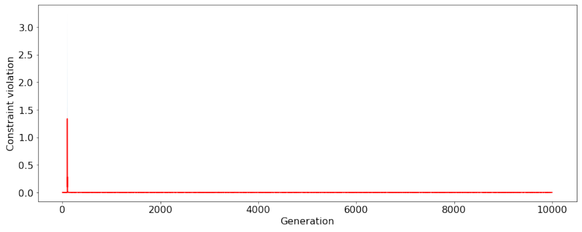

(d) f4: Constraints violation error (lower is better)

Figure 2: Mean value (red) and boundaries (blue between the maximum and the minimum values) of each objective function over 4,000 initial states and for 10,000 generations.

where only the f_3 (overdraft maximization) part is removed. Misclassification and constraint satisfaction are minimum mandatory criteria in order to generate valid examples and thus, all the three CoEvA2 variants we examine include them.

Table 3 summarizes our results. It shows that the variant of CoEvA2 with all parts of the objective function activated is the

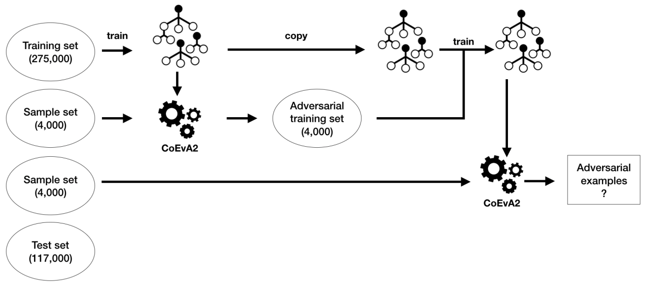


Figure 3: Adversarial training process.

only one capable of generating adversarial examples that cause misclassification, satisfy the constraints and engender relevant overdrafts.

CoEvA2 is successful for 8.45% of the initial states. Thus, on average, only 12 initial states are needed to perform a successful attack. An interesting observation here is that from one initial state we can generate more than one valid adversarial example. This results in more than thirteen thousand of valid adversarial examples bypassing the banking system. These results seem to suggest that the fitness function we form is effective and that all its parts are important.

This last point can be confirmed by the rest of the recorded results. These show that all the objective function parts are necessary to generate successful adversarial examples. Without the perturbation minimization objective (one-before-last column), CoEvA2 generates slightly more misclassified examples (for 31.18% of the initial states instead of 27.10%) but none of them satisfies the constraints. This confirms our previous hypothesis that not restricting the perturbation makes the algorithm create examples much different from the original (valid) example. In highly constrained search space, this increases the likelihood of generating invalid examples.

Finally, without the objective of maximizing the overdraft amount (last column), CoEvA2 generates examples satisfying the constraints in every case. However, only 18.79% of them cause misclassification. None of these achieve a sufficient overdraft amount (1,000 currency units). This is because the algorithm applies only small variations and only to the other mutable features, which reduces the likelihood of violating the constraints and of misclassification.

To better understand how CoEvA2 handles the trade-off between the four objective functions, we show in Figure 2 how the value of each of them evolves over the generations when applied to the initial states for which it managed to generate valid adversarial examples over 10,000 generations.

At each generation, we average the objective function scores obtained by the current population. Thus we obtain, for each initial state and each objective function, 10,000 values (one per generation).

Then, we show the minimum, mean and maximum values of each (averaged) score over all the initial states. The red line is the mean, whereas the blue area denotes the minimal and maximal scores.

The four plots confirm that constraint satisfaction is the first objective fulfilled by the algorithm, and it does so always in the early generations (after about 100). The figure also shows that the overdraft amount is the second-most dominant objective and is always achieved within the first 200 generations, reaching 10^8 currency units, the maximum amount authorized by the CSS. All the individuals of the population in all the next generations inherit this maximum value and keep satisfying the constrains. Meanwhile, the L_2 distance fluctuates around 0.9, which is 10 times less than the Papernot attack. The average prediction probability stabilizes around 0.35, which is slightly below the prediction threshold.

Interestingly, taken together these results suggest that a careful choice of the initial state allows CoEvA2 to find valid adversarial examples after a limited number of generations (200). Moreover, these examples make the system overdraft of high amount (close to the strict maximum authorized by the bank). While frightening, these results also mean that we can focus on specific initial states to build countermeasures and increase the robustness of the system.

Overall, our results corroborate the conclusion that all four parts of our fitness function play a crucial role in crafting valid adversarial examples. At the same time, our approach demonstrates that it is indeed feasible to craft valid adversarial examples in real-world critical systems. This motivates the need for appropriate defence mechanisms to reinforce the robustness of the system against such attacks. We investigate this in the next research question.

8.4 RQ3: Adversarial Training

Figure 3 shows the adversarial training process we designed to improve the robustness of our partner's system against our (previously successful) adversarial attack. First, we generate 4,000 valid adversarial examples (overdrafts accepted by the model that should be rejected) and re-train the model to classify them correctly. To do so, we use the 4,000 initial states used in RQ2. After re-training the model, we execute again CoEvA2 4,000 times using each time a new initial state that was not used to produce the adversarial training set. We check whether the attack managed to generate any adversarial examples.

It results that the adversarial training makes CoEvA2 incapable of generating valid adversarial examples. Thus, our adversarial training method grants protection against the very same attack that was previously effective. This is a positive outcome that can be used by our partner in order to improve the robustness of the CSS. As our system is still in testing phase, it is used in parallel to the original model. Thus, an overdraft approved by the existing model and rejected by ours is likely to be an adversarial example. In all the other cases, one should follow the decision of the first model.

8.5 Threats to validity

Validity threats to our results may arise by the implementations we used. Thus, potential bugs either in our or the underlying frameworks may influence our results. We do not consider this threat as important since we thoroughly checked our code and many of the adversarial examples we generated were verified by our partner. Moreover, we rely on widely used and relatively reliable frameworks, Scikit-Learn and Tensorflow for the machine learning algorithms, and reputable libraries like the Adversarial Toolbox from IBM[6] for adversarial attacks and Pymoo from the Michigan State University [7] for multi-objective genetic algorithms.

Another potential threat concerns the specificity of the dataset and classification model we used. Both are from our partner's real production system and since our partner is a major player, its data and practices should be representative of other companies. Moreover, a verification of historical data revealed remarkable results for the last year. Due to the specificity of our industrial case, the results we obtained may not fully transfer to other industries (namely outside of credit scoring domain). Nevertheless, our endeavour shows that the problem exists in the real world and formalises it to facilitate the design of similar solutions to other cases. Moreover, our algorithm and approach have been designed to be generic enough to be adjusted to other use cases, and we provide the algorithm and all the hyper-parameters of our approach for reproducibility.

To reduce the impact of random effects, all our experiments consider 4,000 different initial states (customer account and transaction history) and run the studied methods once per state. Since we make 4,000 independent executions, multiple runs per execution can only make a difference in isolated cases and not in the overall performance (expected case). This is because initial states can be seen as independent repetitions.

In our experiment, we perform a single run per state since we focus on trends. Thus, we run our approach on 4,000 cases and found adversarial cases in 341. These are sufficiently large numbers to rule out random effects. Yet, multiple repetitions and more generally additional search time-budget may improve the results of the search. We run the random method $4,000 \times 40,000$ times (4,000 initial states $\times$ 1,000 generations with 40 individuals), and found 0 adversarial cases, which demonstrates the ineffectiveness of the random method.

9 CONCLUSION

In this paper, we studied the problem of testing a machine-learning-based industrial credit scoring system against malicious inputs. In particular, we considered the case where an attacker manipulates the related features, with the aim to cause a misclassification by a binary classification model. To this end, we evaluated the current state-of-the-art adversarial attacks, both in a full-knowledge context and a limited-knowledge using our partner's dataset and system. Based on this study, we shew that approaches proposed in the literature can indeed generate adversarial examples but these are not useful since they do account for domain constraints. This limitation of the methods results in generating implausible examples. To deal with this situation, we proposed a search-based method overcoming these limitations. We showed that our new attack constitutes a real security threat to FinTech systems relying on machine learning. At the same time, we exploit this threat to improve the defence mechanisms of our industrial system. In the end, the system becomes immune to our attack.

[6]https://github.com/IBM/adversarial-robustness-toolbox/
[7]https://pymoo.org

REFERENCES

[1] Naveed Akhtar and Ajmal Mian. 2018. Threat of adversarial attacks on deep learning in computer vision: A survey. *IEEE Access* 6 (2018), 14410–14430.

[2] Shaukat Ali, Muhammad Zohaib Z. Iqbal, Andrea Arcuri, and Lionel C. Briand. 2013. Generating Test Data from OCL Constraints with Search Techniques. *IEEE Trans. Software Eng.* 39, 10 (2013), 1376–1402. https://doi.org/10.1109/TSE.2013.17

[3] Moustafa Alzantot, Yash Sharma, Supriyo Chakraborty, Huan Zhang, Cho-Jui Hsieh, and Mani Srivastava. 2018. GenAttack: Practical Black-box Attacks with Gradient-Free Optimization. arXiv:cs.LG/1805.11090

[4] Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. 2019. Generating Natural Language Adversarial Examples. Association for Computational Linguistics (ACL), 2890–2896. https://doi.org/10.18653/v1/d18-1316 arXiv:1804.07998

[5] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. 2013. Evasion attacks against machine learning at test time. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. 8190 LNAI. 387–402. https://doi.org/10.1007/978-3-642-40994-3_25 arXiv:1708.06131

[6] Battista Biggio and Fabio Roli. 2018. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition* 84 (December 2018), 317–331. https://doi.org/10.1016/j.patcog.2018.07.023 arXiv:1712.03141

[7] Nicholas Carlini and David Wagner. 2017. Towards Evaluating the Robustness of Neural Networks. *Proceedings - IEEE Symposium on Security and Privacy* (2017), 39–57. https://doi.org/10.1109/SP.2017.49 arXiv:arXiv:1608.04644v2

[8] Vinícius Veloso de Melo and Wolfgang Banzhaf. 2016. Improving Logistic Regression Classification of Credit Approval with Features Constructed by Kaizen Programming. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion - GECCO '16 Companion*. ACM Press, Denver, Colorado, USA, 61–62. https://doi.org/10.1145/2908961.2908963

[9] Kalyanmoy Deb, Karthik Sindhya, and Tatsuya Okabe. 2007. Self-Adaptive Simulated Binary Crossover for Real-Parameter Optimization. (2007), 1187–1194. https://doi.org/10.1145/1276958.1277190

[10] Hongwei Feng, Shuang Li, Dianyuan He, and Jun Feng. 2019. A novel feature selection approach based on multiple filters and new separable degree index for credit scoring. In *Proceedings of the ACM Turing Celebration Conference - China on - ACM TURC '19*. ACM Press, Chengdu, China, 1–5. https://doi.org/10.1145/3321408.3323928

[11] B. J. Garvin, M. B. Cohen, and M. B. Dwyer. 2009. An Improved Meta-heuristic Search for Constrained Interaction Testing. In *2009 1st International Symposium on Search Based Software Engineering*. 13–22.

[12] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. 2017. Adversarial examples for malware detection. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. 10493 LNCS. Springer Verlag, 62–79. https://doi.org/10.1007/978-3-319-66399-9_4

[13] Alex Kantchelian, J. Doug Tygar, and Anthony Joseph. 2016. Evasion and hardening of tree ensemble classifiers. In *International Conference on Machine Learning*. 2387–2396.

[14] Francisco Louzada, Anderson Ara, and Guilherme B. Fernandes. 2016. Classification methods applied to credit scoring: Systematic review and overall comparison. *Surveys in Operations Research and Management Science* 21, 2 (Dec. 2016), 117–134. https://doi.org/10.1016/j.sorms.2016.10.001

[15] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2017. Towards Deep Learning Models Resistant to Adversarial Attacks. (2017), 1–27. arXiv:1706.06083 http://arxiv.org/abs/1706.06083

[16] Phil McMinn. 2004. Search-based software test data generation: a survey. *Softw. Test. Verification Reliab.* 14, 2 (2004), 105–156. https://doi.org/10.1002/stvr.294

[17] Zbigniew Michalewicz and Marc Schoenauer. 1996. Evolutionary Algorithms for Constrained Parameter Optimization Problems. *Evol. Comput.* 4, 1 (1996), 1–32. https://doi.org/10.1162/evco.1996.4.1.1

[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. 2018. Adversarial Robustness Toolbox v1.1.1. *CoRR* 1807.01069 (2018). https://arxiv.org/pdf/1807.01069

[19] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. 2016. The limitations of deep learning in adversarial settings. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 372–387.

[20] Nicolas Papernot, Patrick D. McDaniel, and Ian J. Goodfellow. 2016. Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples. *CoRR* abs/1605.07277 (2016). arXiv:1605.07277 http://arxiv.org/abs/1605.07277

[21] Cynthia Rudin. 2019. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence* 1, 5 (May 2019), 206–215. https://doi.org/10.1038/s42256-019-0048-x

[22] Roberto Saia, Salvatore Carta, and Gianni Fenu. 2018. A Wavelet-based Data Analysis to Credit Scoring. In *Proceedings of the 2nd International Conference on Digital Signal Processing - ICDSP 2018*. ACM Press, Tokyo, Japan, 176–180. https://doi.org/10.1145/3193025.3193039

[23] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199* (2013).

[24] Lyn C. Thomas, David B. Edelman, and Jonathan N. Crook. 2002. *Credit Scoring and Its Applications*. Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9780898718317

[25] Shayan Zamani and Hadi Hemmati. 2019. Revisiting Hyper-Parameter Tuning for Search-Based Test Data Generation. In *Search-Based Software Engineering - 11th International Symposium, SSBSE 2019, Tallinn, Estonia, August 31 - September 1, 2019, Proceedings (Lecture Notes in Computer Science)*, Shiva Nejati and Gregory Gay (Eds.), Vol. 11664. Springer, 137–152. https://doi.org/10.1007/978-3-030-27455-9_10