A Fraud Detection System Robust to Adversarial Perturbations

Tiago Leon Melo*
tiago.melo@feedzai.com
Feedzai

João Bravo
joao.bravo@feedzai.com
Feedzai

Marco O. P. Sampaio
marco.sampaio@feedzai.com
Feedzai

Paolo Romano
romano@inesc-id.pt
INESC-ID and IST, U. de Lisboa

João Tiago Ascensão
joao.ascensao@feedzai.com
Feedzai

Pedro Bizarro
pedro.bizarro@feedzai.com
Feedzai

ABSTRACT

Adversarial attacks have quickly become a concern regarding many security-centered applications. Typically, the goal with these attacks is to mislead the classifier into producing the incorrect output while keeping the attack fairly similar to a clean example. Fraud detection systems have a naturally adversarial setting, in the sense that there is often an adversary (that is, a fraudster) trying to bypass a model, and a classifier trying to outguess this adversary. Fraudsters have been known to continuously adapt their strategies in order to maximize their chances of fooling the model, which leads the concept of fraud to change over time. To keep up with this concept drift, fraud detection systems often rely on frequently retraining the model on fresher data least they grow stale and their performance degrades. This is an expensive operation. In this work, we propose preempting this adversarial adaptation rather than reacting to it. We hypothesize that by training a model robust to adversarial attacks it will perform better against future attack strategies and thus require less model refreshes. We find that by generating a wide range of attacks and exposing the model to these adversarial samples during training, we obtain a significantly more robust model without considerable loss of performance in historical data, avoiding drops in performance that can be larger than 50% if attacks of the same type are used against a non-adversarially trained baseline.

CCS CONCEPTS

• **Computing methodologies** → **Learning paradigms**; **Supervised learning**.

KEYWORDS

adversarial robustness, tabular data, fraud detection, tree-based models, discrete optimization

ACM Reference Format:
Tiago Leon Melo, João Bravo, Marco O. P. Sampaio, Paolo Romano, João Tiago Ascensão, and Pedro Bizarro. 2023. A Fraud Detection System Robust to Adversarial Perturbations. In *Conference name*. ACM, New York, NY, USA, 10 pages.

*Work developed while employed at Feedzai.

TBD, TBD '23, Month DD–DD, 2023, City, Country

1 INTRODUCTION

Machine Learning (ML) models are a vital part of many security-sensitive applications [14]. Many such systems are vulnerable to malicious actors, who try to exploit them. This is the case in the financial services industry, where ML classification models are used to classify, for example, credit card transactions as legitimate or fraudulent, [4]. By construction, this is an adversarial setting, where fraudsters continuously try to adapt their techniques to bypass the fraud detection systems (for a profit), while the system maintainers strive to improve it to be able to stop such new attacks.

Being able to train a stable model that can withstand such adversarial changes (i.e., catch not only current but also future fraud strategies) can be of great value, since the easy alternative, which is to perform frequent model retraining operations with fresher data, is an expensive operation.

Developing adversarial training for tabular data in the fraud detection domain poses additional challenges. The data is often composed of many categorical and id fields in addition to numerical fields, and it is typically further enriched via feature engineering containing complex temporal and entity based aggregations, [2], before being used to train a ML model. Thus, the perturbations space available for fraudsters to attack the model is not the same space available to train the model, which requires a more complex setup to propagate perturbations from original features to engineered features.

Current approaches in adversarial robustness literature often focus on attacking image data against neural network models. Methods that tackle a problem with a similar setting [4] do not address the propagation of perturbations to engineered features and focus on attack generation strategies rather than boosting model robustness.

To address this challenge, we propose a new set of methods to generate, measure and inject adversarial attacks during training. We aim to employ these methods in the adversarial training of gradient boosting algorithms for tabular datasets and to perform an empirical study of the proposed methods on a credit card fraud detection dataset. In particular, in this work we highlight the following contributions:

- We develop a method to generate strong attacks against tabular transaction data.
- We propose different methods to efficiently update complex temporal aggregation features as a result of adversarial perturbations of the underlying fields they depend on in an approximate way.

- We introduce training frameworks that increase the robustness of a classifier against a broad range of attacks. Particularly, we develop a model that is well protected against multiple attack strategies, regardless of whether or not it was trained against them (i. e., generalizes well against novel attacks it has not been exposed to).
- We achieve the former contribution without sacrificing significant performance on historical data, despite there being a trade-off between robustness and accuracy [16, 17], while avoiding potential disastrous losses if the developed attacks arise.

2 RELATED WORK

Adversarial robustness has recently gained a lot of interest with many studies addressing new strategies to design attacks [3, 7, 8] and conducting reviews on current defense mechanisms [9]. However, most literature leans towards the image classification context, with relatively little research tackling this concern in the tabular domain. One common approach is to get the adversarial sample closer to the original sample through the usage of customized distance metrics [4, 5]. Another approach is to compute and update perturbations based on the gradient of the model [1], and to later enforce each feature's domain through additional operations (like rounding to the nearest integer, for example). Additionally, there are also genetic algorithms that introduce random noise (within a ball), sampled from each features acceptable set of values [11].

2.1 Problem Statement

Let us consider the loss function as $J(\theta, x, y)$, which corresponds to the cost of labelling an example x of label y using parameters θ. This cost function is what we (as the model developer) want to minimize when optimizing for θ. An adversary will often want to adjust the input, $\tilde{x}$, that is fed to the model (for example, an image in an image classification scenario) so that they maximize the loss function:

$$\max_{\tilde{x} \in x+\Delta} J(\theta, \tilde{x}, y) \tag{1}$$

Typically, these perturbations to x cannot be arbitrary. In most scenarios, the generated example still needs to be close "enough" to the original input. In some contexts, certain features of x cannot be edited or have to conform to the original input domain. We denote this allowable set of perturbations by Δ.

2.2 Attack Properties

It is important to distinguish between two classes of attacks: white and black box. As the names suggest, a white box attack assumes an attacker not only has access to the weights and architecture of the model being attacked, but also can perform infinite queries and try out as many attacks as desired. Naturally, this approach poses a more serious threat to a target model, since attackers can iteratively perfect their own attacks and devise strategies with high success rates [3]. However, it is argued [7] such approaches are somewhat detached from reality: in practice, attackers have no access to the real model being attacked, and more importantly, there is a limited number of queries an attacker can perform before being flagged as suspicious and denied any further attempts. This latter approach has been coined as a *"black-box"* attack, and has been studied in contexts that often resemble real world scenarios; i.e., when the attacker only has access to a hard label and not a score [7].

2.3 Attack Strategies

Adversarial attacks aim to solve the maximization problem posed in equation 1: given a set of allowed perturbations, find the adversarial sample that will maximize the loss of the target model. One of the first proposed methods to generate adversarial attacks was the Fast Gradient Sign Method (FGSM) [8]. FGSM solves the constrained problem by replacing the cost function J with a linear approximation around the victim sample x. This solution can be interpreted as being a one-step scheme for solving 1 [13]. Later, more elaborate approaches propose to develop a stronger attack by iteratively perfecting the adversarial perturbation; that is, performing projected gradient descent upon the negative loss function. If we consider FGSM corresponds to a single gradient step, we can consider PGD to try to solve equation 1 with multiple projected gradient steps, where the result is projected onto the set of valid adversarial examples [13].

Attacks reliant on the model gradients are naturally only applicable to differentiable models that the attacker has access to. This excludes tree-based models and adversarial attacks developed in a black box setting. A common approach in both of these settings is to compute the zeroth order gradients to generate attacks - the ZOO attack [7].

2.4 Defense Strategies

In order to increase the robustness of classifiers against adversarial examples, Szegedy et al. [15] propose *adversarial training*. Essentially, adversarial training consists of a slightly more sophisticated method for data augmentation. The intuition lies on the premise that if the model is exposed to adversarial samples, its performance against similar attacks is better. A core difference between adversarial training and standard data augmentation techniques is that the latter expands the training set using transformations that are likely to occur naturally within the training set (or in nature). The goal with adversarial training on the other hand is to expose flaws in how the classifier models its decision function; thus the generated example are less likely to occur naturally but rather target these specific sub-spaces of the input spaces [8]. So, when training a model, we can iteratively generate more adversarial examples as it updates. The primary reason for this to be done in train time and not prior is that the concept of an adversarial example depends on the current parameters of the model. An adversarial example generated prior to training is, most likely, very different to one generated post training. The authors show that by training the model using a mixture between clean and adversarial samples the model can thus be regularized [8, 15].

3 METHODS

In this section we detail the methods developed in this paper. We aim to provide a clear definition of the perturbation space we use to generate attacks (keeping the fraud detection use case in mind), the strategies that search through this perturbation space, and finally how these strategies are leveraged to adversarially train our models.

3.1 Overview

Our main goals in this study are to understand how robust models without adversarial training are, and how to boost their performance with adversarial training. Our methodology is as follows:

(1) Leveraging the domain knowledge collected from risk analyst interviews, we define a perturbation space suitable for the context of fraud detection.
(2) We then develop baselines according to common practice in the context of fraud detection. To do this, a baseline model is trained using historical data and then it is evaluated on a dataset with and without adversarial samples.
(3) We explore the perturbation space to generate adversarial attacks. This step entails three main stages: designing a norm to express the size of a given perturbation, defining and implementing an efficient method of searching through the space (that is, *an attack strategy*) and finally benchmarking, in which we compare the effectiveness of each attack strategy. We craft a range of attacks that can bypass some of the difficulties posed by tree-based classifiers by treating the setting as a black-box under partial observability.
(4) Finally, using the best attack strategy from the last step, we train the models to be adversarially robust. We experiment adversarial training in different conditions and assess the results.

3.2 Domain Specific Concepts

3.2.1 Feature Engineering. We now highlight some key concepts when discussing tabular data that has undergone feature engineering pipelines with complex aggregations. We distinguish between raw and engineered features, and discuss how the latter are obtained. Feature engineering essentially takes in a set of (usually raw) features and adds features that are computed from the existing ones in order to extract some signal that may be relevant to the model. We break down the most relevant concepts in play when other features are generated. There are two main types of derived features: profiles and mappings. Profiles (also known as time aggregations) are features generated from performing a grouping operation over some time window. A very practical example of a profile would be the count of transactions sent out by a card in the last 24 hours. Here the set of columns used to group data would be the card, and the time window would have a duration of 24 hours with no delay, computed in real time. Another example where profiles are calculated with aggregated features would be the total amount (sum) of money sent out by a card in the last hour. Besides profiles, through feature engineering we can also generate additional features through mappings, which, as the name suggests, consist of applying a specific mapping function to one or more features. A simple example would be the generation of a feature that is the logarithm of the amount sent in a transaction.

3.2.2 Metrics. The ROC curve is mostly used to display a binary classifier's performance over the possible choices for thresholds. It allows the use of a single metric to summarize the performance of a model: the AUC. The idea is that the larger the area, the better TPR-FPR trade-off we can incur in. However, all portions of the curve are considered when computing this metric - even overly large FPRs we might not be interested in (in a fraud detection scenario we would typically target a very low FPR). We circumvent this issue by only computing the AUC up to a given maximum FPR - i.e. the partial area under the curve (pAUC). Additionally, we normalize the pAUC by dividing it by the maximum FPR we are considering (see Equation 2). This metric conveys the average recall in the region of interest between 0 and our established maximum FPR.

$$pAUC(\mathrm{FPR}_{high}) = \frac{1}{\mathrm{FPR}_{high}} \int_0^{\mathrm{FPR}_{high}} \mathrm{TPR}\ d(\mathrm{FPR}) \qquad (2)$$

3.3 Perturbation space

We conduct interviews with risk analysts to gather domain knowledge on how fraud is typically done. In summary, given the information collected in the interviews, we consider there are, in total, 6 possible perturbations we can apply to the dataset with different repercussions:

- **Amount perturbations** - we consider it is possible for a fraudster to change the amount involved in a transaction. In our setup, the possible perturbations to the amount span from reducing it to 2% of its original value to multiplying five-fold.
- **Temporal perturbations** - temporal perturbations are particularly unique, since we are only changing the transactions that are used to compute profiles. We allow a perturbation to span anywhere between one minute to one week - meaning a fraudster can wait or anticipate their transaction for a duration of time between those two.
- **Card resets** - from the interviews we gather it is frequent for a fraudster to have access to an array of cards. That being said, we consider that it is possible for them to start using a new card, in which case we assume it has never been seen before in the dataset. Hence, the natural implication is that any profile with "card" in its grouping entities is reset under such a perturbation.
- **Card switches** - besides switching card, there is also the possibility to switch to a card with different features, for instance, switching to a card issued by a different entity. We distinguish the two perturbations: to simply change card but keep all the card-related features (for example, the identifier of the issuer that manages a given card) and to get a new card altogether with different card features. Naturally, the latter implies the former.
- **Geolocation perturbations** - we expect fraudsters to be fairly tech savvy. For this reason, we consider it should be possible for them to manipulate their perceived geolocation.
- **Network perturbations** - similarly to the previous point, a tech savvy fraudster should be able to manipulate the perceived characteristics of their network, or simply changing them altogether. This, along with card switches, make up all the possible categorical perturbations.

We include a visual representation of the perturbation pipeline we follow to apply perturbations to a dataset in Figure 1. We now discuss the implications of temporally perturbing samples, as well as some aspects to consider when perturbing the amount.

3.3.1 Timestamp Perturbations. In order to generate adversarial examples through timestamp perturbations, we must ask ourselves: what would have happened if this transaction had occurred at a different time? An exact solution would imply recomputing the profiles over an copy of the dataset, with the only difference being

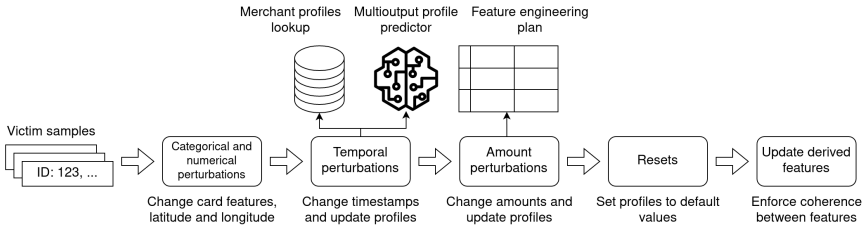

Figure 1: Diagram for the full perturbation pipeline

that each victim sample would have a new timestamp occurring at the new, perturbed time. This would always lead to correct results and a valid, realistic adversarial sample. This solution is nonetheless computationally very heavy, hence not feasible, since it would be fairly slow to run in the inner adversarial optimization loop for many transactions.

A second approach is to leverage our own advantage of hindsight to estimate the new values for the profiles: we know how each profile changes over time for any given values of the grouping entities. So, if we want to estimate what is the total amount sent to a specific merchant in a specific datetime, we need only to perform a look-up operation of a transaction that occurred around that time. For efficiency reasons, this method can be further improved upon if we preemptively bin the profiles over time (say, calculate the mean for each profile using bins of one hour). This way we only need to figure out in which bin a perturbed timestamp lies and fetch the corresponding mean as suggested in Figure 2.

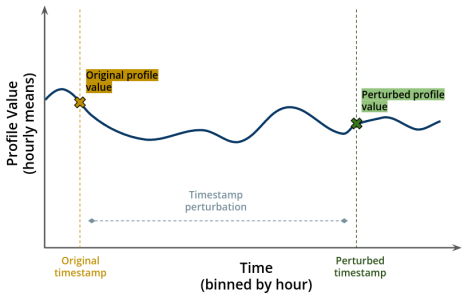

Figure 2: Diagram illustrating the intuition behind the estimation of high-volume profiles

The major assumption of this method is that the mean is a representative value of the profile, making it a good approximation. We hypothesize that this assumption holds for profiles calculated over a grouping entity with a large volume but cannot hold for more specific grouping entities with comparably little volume (for example, cards). This reduces the applicability of this method to predict changes to a subset of features.

A third approximate method to estimate how profiles change over time is to fit a regression model specifically for this task. We thus propose training a multi-output regression model to predict the changes to each of the profiles. As inputs to this model, we use a dataset where each row corresponds to the original features, the original values for the profiles, the amount of delay that was injected in the timestamp and, as regression targets, the resulting profile values if the transaction had taken place in the perturbed timestamp. In order to build this dataset, we run the feature engineering on a dataset of raw fields where, for each chosen card to perturb, a single transaction was perturbed in its history. This is because perturbing multiple instances would leave us with a too-different history compared to the original and the input space for the regression task would have to be dependent on the number of perturbations for each card, and on their features (so that the prediction of the targets would be sensitive to those multiple perturbations). We adopt this subset of perturbations in this study and leave an analysis of the effect of more complex timestamp perturbations to future work. Once the dataset is built, we proceed to train an off-the-shelf multi-output regressor.

We use the second method to predict merchant profiles and the third method for the remaining profiles. The second method is trivial, given that it consists of a look-up table we can query to draw estimations from. On the other hand, the third method requires building a dataset to train our regression models with, and a careful analysis of their performance which we address in section 4.

In order to train a model that can estimate changes successfully, we must produce a dataset that contains multiple perturbed instances corresponding to profile changes over time, spanning multiple perturbation time intervals. We thus start by defining a range and distribution of acceptable delays to inject on the samples that are victims of the adversarial perturbations. Thus, we consider that a delay (either positive or negative) should lie between one minute and one week to be consistent with the time sensitivity of the typical timescales of the window used for profiling, which should relate to the fraudster's "willingness" to delay or anticipate their attack. We choose to draw delays from a logarithmic distribution. We want to build a sample with a frequency of input delays that is similar in each order of magnitude - i.e., we want the frequency of 1-minute delays to be comparable with the frequency of 1-week delays. Thus, we draw delays according to the following equation:

$$delay = \text{randchoose}(\{-1, 1\}) \cdot e^{u}\,, \tag{3}$$

where randchoose$(\{-1, 1\})$ chooses either sign with equal probability and u is drawn from the uniform distribution

$$U([\log 1 \text{ minute}, \log 1 \text{ week}])\,. \tag{4}$$

3.3.2 Amount Perturbations. For these perturbations, we parameterize the perturbations with a scaling factor relative to the original amount. For instance, if the original amount is 100 and the

perturbation is 1.3, the resulting amount is 130. This has the advantage of allowing us to adjust derived features, such as original currency amount, without having to deal with "exchange rates". In this step we also update amount-related mappings, for example the log amount.

After perturbing the raw and other (non-profile) derived features, profiles must be updated to match the new amount values. Since this logic depends on each profile (update frequency, grouping operation, etc.), each type of profile requires a separate implementation for such an update. We highlight the profiles that are updated when we perturb the amount can consist of a sum, maximum, average and standard deviation aggregation operations.

3.4 Perturbation norm

To be able to quantify the magnitude of a perturbation, we define a norm that aims to capture how expensive it is for a fraudster to apply a given perturbation, henceforth referred to as the **fraudster cost norm**. In order to achieve this, we consider the raw perturbation (that is, before feature engineering) multiplied with a custom set of weights for each vector of attack. This norm is very practical, since it can effectively be computed before the perturbation is *applied*. Moreover, we also consider this norm to be relevant for the problem we are solving, because it relates more directly to the real costs that drive the fraudster's decisions when attacking the model.

We depart from the possible directions of attack (enumerated in list 3.3) and assign a cost to each raw perturbation. Since most perturbations are discrete, this cost is fixed in most cases. For example, we assign a cost of 33 units to resetting a card. The only two exceptions to this rule are temporal and amount perturbations. Here we found it best to design and employ a custom cost function depending on how great the alteration really is. Naturally, waiting one week should weigh considerably more than waiting a couple of hours. Along the same line of thought, changing the amount to be five-fold should be more expensive than doubling it. It is worth highlighting how the cost should vanish when the amount multiplier nears one (unperturbed amount). Moreover, lowering the amount involved in a transaction should also be costly. Conceptually, we can think this maps to a fraudster selecting a much less desirable basket of products to better their chances of bypassing the fraud detection system.

As for the range of values that this norm can assume, we allow it to go up to 100 when considering all the perturbation directions. We consider that reducing the original amount to 1% of its original value is as "expensive" for the fraudster as it is to get a new card, since both of these operations represent a major change in how a fraudster must actually commit fraud. These represent the largest two perturbations, and from then we adjust (i.e. waiting one day is similar to tripling the amount involved in the transaction).

3.5 Adversarial attack searching methods

In this section, we describe the procedure followed to generate attacks using the previously described perturbation space. We use random search as a baseline to assess the strategies we develop later on, all considering a black-box setting under partial observability.

3.5.1 Random Search. We define one pass through the pipeline in 1 to correspond to one iteration. Our random search will generate many adversarial samples, each independently from the other. Our goal is to have as many adversarial samples as possible, each with various perturbation sizes. At each step, we perform a Bernoulli trial for each perturbation direction to decide whether we perturb it. We kept each chance low, so that it is not very likely to perturb a transaction using all raw perturbations, which would skew the distribution of norms to large values. For instance, to perturb the timestamps of a victim sample of size M, we perform M Bernoulli trials, where it is more likely that the timestamp will go unperturbed. A similar strategy is followed for every other perturbation direction, with no dependence between each other with the exception of card features, for the aforementioned reasons.

In order to evaluate an adversarial strategy, it is important to consider both its success rate and the size of the perturbations it generates. The success rate can be computed by dividing the number of successful adversarial transactions over the total number of attacked transactions:

$$Success\ Rate = \frac{\#\ successful\ attacks}{\#\ generated\ attacks}\ . \tag{5}$$

An adversarial transaction is successful if any of the randomly generated attacks is successful. In other words, a transaction that was formerly labelled as positive, is now labelled as negative. Since we are attacking fraudulent examples exclusively, a set of successful attacks is a set of fraudulent samples that is now classified as legitimate and was previously classified as fraud. We evaluate the success rate of random search under various norm constraints.

3.5.2 Stochastic Coordinate Descent. Given the high prevalence of discrete variables in the search space of the problem we are trying to solve, we opted to start by employing standard algorithms from discrete optimization literature, namely Stochastic Coordinate Descent (SCD). In essence, SCD consists of iteratively picking some random direction to attack, exploring its values and updating the current perturbation to the best value along the direction if it is better than our current perturbation (i.e. results in a lower score). We repeat this for every direction until we conclude a directions sweep. We consider to have converged whenever we perform a full direction sweep which has yielded no improvements. Every time we pick a direction we generate a grid of possible values to explore within the bounds of our norm constraint. For categorical features we explore every possible value, for numerical features we generate a discrete grid of values.

This overview of the method indicates some possible variations that can be explored, namely regarding which perturbation to select from the generated grid (i.e., what is the criterion that defines the *best* perturbation). In this paper we explore a greedy method of selecting a perturbation and a cost-efficient method.

Greedy approach The first variation of SCD (i.e. the greedy method) consists of selecting the perturbation that yields the lowest score from the generated grid. If we observe that this score is lower than our current best perturbation, we replace the latter by the new perturbation. We expect this implementation to converge fairly fast, while being prone to get easily stuck in local minima. Regarding how well we expect this strategy to explore the space, due to the tendency to evolve quickly to local minima, we foresee that the distribution of norms may not be very close to the maximum norm constraint and, as a result, the success rate may be sub-optimal.

Cost-efficient approach The key idea of this approach is to try to avoid getting stuck in local minima. We hypothesize that without any restraint, being greedy will ultimately just result in picking the most expensive perturbation, effectively leaving us no room to keep exploring. For this reason, we propose a method that instead optimizes for cost-efficiency: in each iteration, we select the perturbation that offers the best norm change to score change ratio. We expect this version of SCD to offer higher success rates at the expense of more iterations and a longer time to converge.

3.5.3 Non-Stochastic Greedy Search. Besides the presented strategies, we can also consider taking the SCD greedy strategy one step further: at each iteration, we generate a grid for all directions **individually**, effectively considering 1) what direction would be best and 2) what would the best step along that direction be. In other words, we explore every direction simultaneously and pick the best one. This arguably circumvents the local minima problem, allowing us to more safely be greedy. We expect the success rate results for this method to be the best, at the expense of more iterations and longer times to converge.

3.6 Adversarial training method

Our implementation of adversarial training can be split into three main parts: attacking, validating and training. We start by generating attacks against a baseline model. These attacks take as victims a percentage of all positives in the training set and all positives in validation. The attack strategy to be used can naturally be adapted and has its own set of parameters - the most important being the norm constraint. This parameter establishes the "most expensive" attack the strategy is able to generate, which we expect to play a vital role in determining the attack's success rate.

We then validate the attacks generated as well as the model's performance against them. We compute three metrics to assess the model. All metrics are computed using the validation set. In particular, we compute the **Clean pAUC at a 1% FPR** - the area under the curve, thresholding at some fixed FPR (in our case, 1%), using exclusively non-adversarial data. This lets us quantify the performance of the model against a dataset without any adversarial attacks - as a result, we have an unbiased estimate of how well we expect our model to fare in the clean test set used in the last step. Similarly, we compute the **Adversarial pAUC at a 1% FPR** to measure the performance of our model in an adversarial setting. This metric is computed exactly as the former, except that it is calculated over the validation set that has had its fraud replaced with their adversarial counterparts. We highlight that this score can only be as high as our clean AUC, since the attack strategy will never generate perturbations that are easier to classify by the model. Lastly, we also assess the **Success Rate**, as it is the most intuitive way of quantifying the effectiveness of the attacks generated.

Finally, we include these newly generated samples in the training set. As mentioned, we generate attacks from a fraction of the training set and all the validation set positives - meaning that for the training set a fraction of fraud is replaced with adversarial samples, and for evaluating the adversarial validation metrics, all fraud is replaced with adversarial attacks. Conceptually, we can think of this as a model being attacked every few iterations. While training, the model uses as validation the adversarial validation set generated in the "attacking" step. This process repeats for a number of times, with the model being iteratively attacked. It is possible to stop early, should there be no significant improvement in Adversarial AUC for a given number of iterations in a row.

4 EXPERIMENTAL SETUP

4.1 Data preparation and data splits

From the conducted risk analyst interviews, we opted to use a payment processor dataset. This is mainly due to 1) the fact that fraud tends to change faster within this use case and 2) the intersection with Feedzai's core business.

To build a sample that is going to be feature engineered, it is important to consider that a very frequent grouping entity for building profiles is the card used in a transaction. Hence, the sampling strategy followed consists of including 100% of cards with at least one fraudulent transaction and 5% of cards with exclusively legitimate transactions. Additionally, we only keep a subset of cards that are meant to be scored in production by the fraud detection system. After employing this sampling strategy, the resulting dataset contained ∼180 million transactions and a fraud rate of 0.2873%.

The derived features were engineered by Feedzai's internal teams for a production system, which ensures we have will have a strong baseline model. One of the derived features flags a transaction to be either Card Present (CP) or Card Not Present (CNP). In this project we focus on CNP transactions since adversarial adaptation should be more of a concern in online transactions. After computing the engineered features, we proceeded to split the resulting dataset between CP and CNP transactions. After keeping only CNP transactions the sample contained ∼34 million transactions with a fraud rate of 1.22%.

We split the data in training, validation and test sets over time, in the sense that each split takes place after the former. Our train set is comprised of 10 weeks, validation of 4 weeks and our test set contains 6 weeks.

4.2 Baselines

Tree-based models have been shown to perform better when classifying tabular data in comparison to deep learning approaches [10]. The predictive power of ensembles of decision trees comes from the principle that many weak learners can be used together to create a strong model. In boosted trees, we can consider we start off with a single weak learner. In each boosting round, we fit another weak learner which should improve the performance of the joint model in its previous round. We create this new estimator by using the gradient of the loss function w.r.t. to the output of the model in the previous boosting round. A popular implementation for gradient boosted trees is proposed by XGBoost [6]. LightGBM [12] further improves this implementation by reducing the number of data points that are scanned when boosting trees, effectively reducing computational cost. We thus use LightGBM models.

We develop baselines according to common practice in the context of fraud detection. To do this, a LightGBM model is trained using historical data and then it is evaluated on a dataset without adversarial samples. We later compare this classifier with a model trained to be adversarially robust.

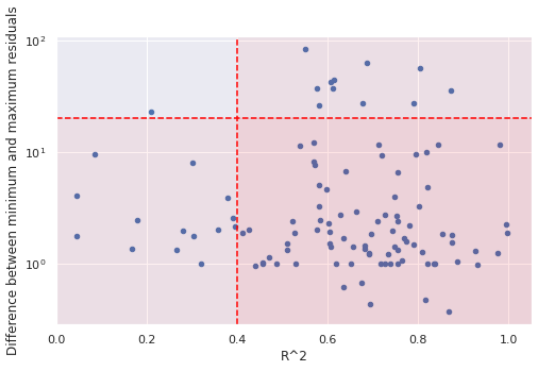

Figure 3: Region of interest for profile predictions. We discard large normalized residuals and profiles the regression model struggles to predict.

5 RESULTS

5.1 Baselines and Profile predictors

We now present the baseline performance we obtained. To assess the performance of the model we compute two metrics: the normalized partial AUC up to an FPR of 1% and the recall at a fixed FPR of 1%. We obtained a pAUC of 0.3715 and a recall at 1% FPR of 0.5155.

As for the profile predictors, we train LightGBMRegressor models across a hyperparameter space, validating the results with the validation set. Then the final model is evaluated using the test set. We performed hyperparameter tuning for a single profile regression model and then used these hyperparameters to train the remaining models (each one responsible for predicting each of the remaining profiles).

We evaluate the regression models using two metrics: the RMSE and the R^2 metric. The RMSE consists of averaging the residuals of the predictions our model is making. R^2 corresponds to the coefficient of determination and it yields a score of 1 if the model outputs perfect predictions and a score of 0 if corresponds to the performance of a model that predicts the mean value. In that sense, if this score is below 0 an estimator that predicts the mean performs better. We compare the achieved results with a baseline that always assumes no change in the profiles. We verify there are significant improvements, but still some fairly low scores, which raises concerns. To investigate this further, we conducted an analysis over the residuals produced by the model - here we are interested in observing what are the maximum and minimum deviations we can expect. We found some large residuals that our model would produce when attempting to predict certain profiles - for example, profiles that attempt to measure the time since a previous occurrence. In order to identify under-performers, we normalize residuals by dividing them by their corresponding feature's standard deviation. We then compute the difference of the maximum and the minimum of these normalized residuals since ideally both would be kept low. We plot this difference across a two-dimensional plane along with their respective R^2 performance. We highlight the desirable region of interest in darker red (bottom right quadrant) in Figure 3. We experimented removing the features outside this region for the model training altogether and repeated the procedure followed to obtain the baseline models. Since the drop in performance was not statistically significant, we opted to discard these features for the rest of the project. We consider this approach satisfactory because 1) the reduced features set contains enough signal to obtain a strong baseline model, and ii) the performance of the regressors for the surviving features is high enough to predict many of the profile shifts with good enough accuracy for adversarial training.

5.2 Adversarial attacks benchmarking

We compare the four formerly presented attack strategies: random search, the two SCD variants (greedy and cost-efficient) and greedy search. As previously mentioned, we use the random search as our attack baseline. We draw comparisons between strategies by analyzing the different norm distributions (i. e. how well we are exploiting the search space); the different success rate *versus* norm constraint trade-offs; and the score distribution shifts (i. e. how much we can influence the scores produced by the model under attack).

5.2.1 Random search baseline. In Figure 4 we show the success rate versus the norm constraint on the perturbation size. In this benchmark, we ran 500 iterations of Random Search. We display two curves to show the difference in success rates when including and excluding temporal perturbations.

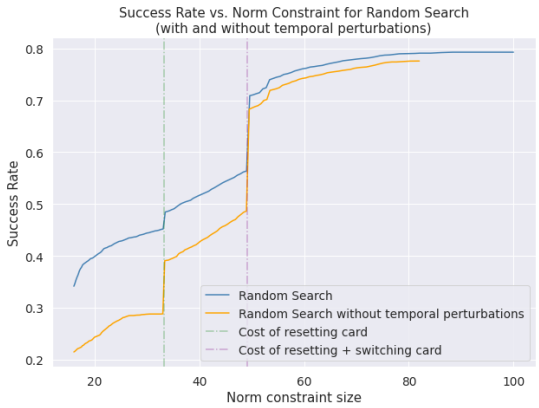

Figure 4: Success rate against norm constraint for Random Search with and without temporal perturbations

We visualize the success rate as a function of the allowed perturbation magnitude, measured using the fraudster cost. As expected, as we loosen the norm constraint we get naturally higher success rates, that is, more successful adversarial attacks. This figure also reveals that there are two very effective discrete perturbations we can do: resetting a card and switching a card (which also demands resetting it). The cost of each of these is represented in the figure by the vertical dashed dotted lines. We point out that, given the attack evaluation procedure, this analysis is unfavourable against this strategy, as higher norm constraints will have a larger number of attacks being considered since in these large spaces it is harder for the random search to saturate the bound. Nonetheless, we observe the designed perturbation space can be reasonably well explored with a simple random search, finding attacks from as little

as 20% success rate up to a little over 80%. When looking at the figure we can see the strategy reaches its peak well before it hits the maximum norm constraint (100), suggesting that other attack strategies can maybe find more successful attacks in larger, less constrained domains.

Before exploring SCD, we reflect upon the role of temporal perturbations. Measurements of execution times show that injecting temporal perturbations increases execution times fourfold, compared to runs without temporal perturbations. As a result, we opted to ignore them in most of the following experiments - meaning the fraudster cost norm can now assume a maximum value of 82.

5.2.2 Attack strategy comparison. We now analyze the results obtained using the remaining strategies.

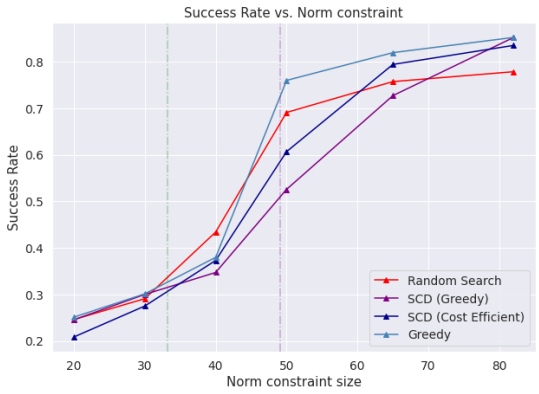

Figure 5: Success rate curves for multiple attack strategies

In Figure 5 we compare the performance of each strategy. We remark how the cost-efficient approach is able to get more efficient attacks than its greedy counterpart for most of the norm constraints. However, both of these still seem to struggle with beating our baselines, having only both achieved so after the 80 fraudster cost mark. These results are especially surprising considering we are including progressively more random search attacks as we relax the constraint - so lower norms arguably injure random search's performance as previously mentioned. Nonetheless, we can easily see the greedy approach can beat the benchmark for nearly all norm constraints. Moreover, it even seems that if we allowed perturbations to be bigger we could hit higher success rates. We detail the success rates for the various norm constraints across all four strategies in table 1.

Attack Strategy	20	30	40	50	65	82
Random Search	0.246	0.291	**0.434**	0.691	0.757	0.778
SCD Greedy	0.245	0.300	0.347	0.526	0.727	0.852
SCD Cost-effic.	0.208	0.275	0.373	0.607	0.794	0.835
Greedy	**0.251**	**0.301**	0.380	**0.760**	**0.819**	**0.852**

Table 1: Success Rate at different norm constraints

5.3 Adversarial training experiments

In this section we detail the results obtained in the conducted adversarial training experiments. We choose to fix the same hyperparameters for training the model adversarially to those already found for the baseline without adversarial training. We hypothesize the results should not vary too much if we were to perform hyperparameter tuning every adversarial round, since a large portion of the data is still clean (i.e. the same data used during training in the baselines). Nevertheless, there are certain parameters that we may vary when performing adversarial training. We vary three parameters: the norm constraint used to generate attacks, the fraction of positives that are adversarial in the training set and the frequency with which attacks are generated during the model training. We start by exploring a few possible combinations of frequency and adversarial fractions for two norm constraints. With this information, we select the best configuration to then sweep over the norm constraint space.

We expect the adversarial fraction to play an important role in balancing the performance in clean and adversarial settings - namely, as we increase the adversarial prevalence in the training dataset, we expect the model to yield a better adversarial performance at the expense of a worse performance on clean data. We consider an adversarial fraction of 0.05 and of 0.25. Moreover, we also consider two approaches regarding the frequency with which we attack the model. We might want to attack it every few boosting rounds, or we might want to wait until it converges, as measured by adversarial pAUC, before attacking it again. We compare the two approaches. It is worth highlighting that we start this introductory exploration with a norm constraint of 30, meaning that no "strong" perturbations (like card resets) are present, and a norm constraint of 65, where all the perturbations are already available.

After fixing the frequency and victim rate, we generate adversarial models using four distinct norm constraints. Once we obtain these adversarially robust models, we can assess their adversarial robustness. We attack each of these models with the same four norm constraints with no temporal perturbations and then, for fewer norm constraints, we conduct the same analysis including timestamp perturbations. We then score all models on a clean dataset and asses their performance over historical test data.

5.3.1 Parameter Tuning. We now explore the results obtained in validation in all four settings with a norm constraint of 30 and 65. One of the key metrics to indicate whether our models are getting more robust is the adversarial pAUC. We analyze the adversarial pAUC, and illustrate this evolution in Figure 6. The results seem to suggest that larger adversarial fractions in the training set can yield larger improvements in adversarial performance in early iterations, but that eventually both converge to similar values. We observe similar results for a norm constraint of 30 and 65, and so we only present the results for the former norm constraint.

Another important aspect to consider is the performance of the new adversarially robust model on clean data. In Figure 7 we visualize the trade-off between clean and adversarial performance at each iteration, for each experiment. We remark how clean performance seems to vary significantly depending on the combination of parameters being explored. Indeed, it is evident that the performance on clean data suffers when including a larger adversarial

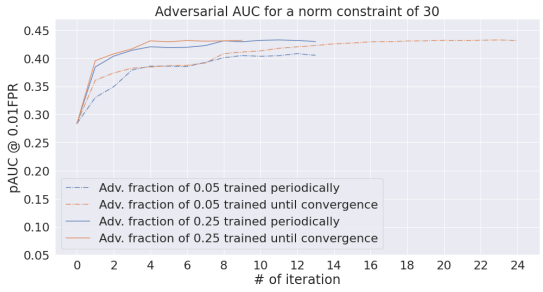

Figure 6: Adversarial training - adversarial pAUC evolution through epochs

sample in the training set. In order to pick the best combination of parameters and iteration, we observe that if we prioritize gains in adversarial robustness by picking the point that offers the highest adversarial pAUC, we are not incurring in significant clean pAUC costs. Therefore, the parameters we pick are an adversarial fraction of 0.05, and training the model until convergence every adversarial round.

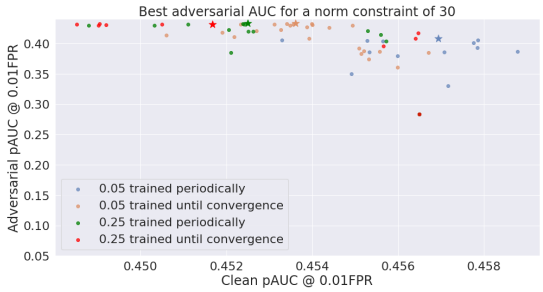

Figure 7: Adversarial training - adversarial against clean performance for each iteration in each experiment for a norm constraint of 30. At each experiment, we star the iteration with the best adversarial pAUC which effectively will be the model chosen from each experiment

5.3.2 Test Set Results. With the frequency of adversarial rounds and adversarial fraction fixed, our experiments will now consist of sweeping a range of norm constraints when adversarially training the model.

Robustness should improve as we use stronger attacks to train the model, the underlying assumption being that a model robust against a strong attack will also be robust against a weak one. We assess these hypotheses by evaluating the pAUC after attacking each model and replacing the positives with these newly-generated adversarial attacks.

From the results presented in Figure 8 we can draw some relevant conclusions. First, we see that a larger norm does not necessarily produce a more robust model - observe how a model with norm constraint of 65 achieves comparable robustness compared to one with 82 (that is, the least constrained setting) - provided that both capture the same type of perturbations. We can also observe that a model that has been trained against amount and categorical perturbations (norm constraint of 30) still has a better-than-baseline performance when it comes to defending against larger spaces with card resets - even though it was not exposed to this type of perturbations during training. Our reasoning is that this difference in success rate between the baseline and the model with norm constraint of 30 comes from the fact that our model should now be more robust against amount and some categorical perturbations. All in all, we observe that all the trained models have become significantly more robust to the perturbations used to train them. In other words, we observe that when we cross the norm constraint threshold to train a model with attacks that capture the entirety of the perturbation space (larger than 49), there is a limited gain in including larger attacks during adversarial training. It is also evident in this figure the decrease in performance in the baseline models (i.e. that were not adversarially trained) as we inject increasingly larger attacks.

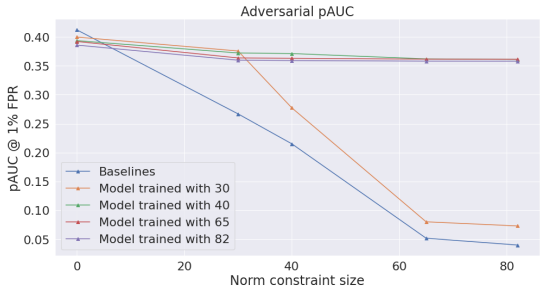

Figure 8: Adversarial training - adversarial pAUC scores for different norm constraints.

We show the results for the experiments where we include timestamp perturbations when performing adversarial training. As previously mentioned, timestamp perturbations are very computationally demanding, so performing adversarial training with this full perturbation space is an expensive task. For this reason, we have opted to explore two norm constraints: a norm constraint of 30, effectively only allowing small perturbations (that is, without card resets/switches); and a norm constraint of 100, allowing the full exploration of the developed perturbation space. When including timestamp perturbations, the findings resemble the ones already presented: as we include larger perturbations when performing adversarial training, we get a model that is more robust to a broader spectrum of perturbations as can be seen in Figure 9.

Finally, we assess the performance of the adversarially robust models over clean, historical data. To reiterate, we expect performance on clean datasets to decay substantially as we include adversarial attacks during training. In Figure 10 we have plotted the performance of each model along with their respective standard deviation. The standard deviation of pAUC was computed by randomly building a large number of bootstrap replications of the test set and computing the performance for each one of them. It is worth highlighting that this method only captures the estimation error in the test set, and not the variability from running adversarial training with different seeds. In the figure, note the black line which represents the baseline performance. These results are as expected: we are sacrificing some performance on clean, historical data in exchange for gains in robustness. In other words, when

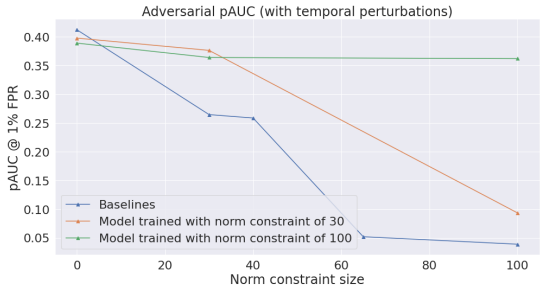

Figure 9: Adversarial training with temporal perturbations - pAUC against norm constraint used to train the model.

we include adversarial attacks during training, we "pay" a price in clean performance to gain adversarial robustness. We also find that the performance of a model trained with a perturbation space that includes timestamp perturbations with a norm constraint of 30 is similar to the previously seen model trained under the same constrain (Figure 10). Interestingly, a model trained with a norm constraint of 100, fully using the space, obtains a slightly superior performance in clean performance compared to the model with a norm constraint of 82 without timestamp perturbations. This suggests that this broader space might result in models that generalize better for future data (reasonable performance in historical data with excellent robustness against future attacks), and reinforces the importance of designing a complete perturbation space.

Figure 10: Adversarial training with and without temporal perturbations - clean pAUC against norm constraint used to train the model.

6 CONCLUSIONS

In this work, we have presented a framework to boost the adversarial robustness of fraud detection models. By defining a perturbation a space, exploring it and including the best attacks found therein in the training set we have successfully developed models able to withstand otherwise strong attacks.

We reiterate how important it is to define a complete, meaningful perturbation space. Moreover, we recall that, in this domain, this can only be done with recourse to domain expertise - more so considering that we are handling tabular data. If the perturbation space we generate is disconnected from reality, we might be making models more robust against perturbations that do not occur and still pay a price in historical performance.

In this work we have verified the improvements in adversarial robustness for all experiments when attacking a model with the same norm it was trained with. Moreover, we also conclude that there is a limited gain in generating arbitrarily large attacks - both in terms of clean and adversarial performance - as long as the perturbation space is fully captured. Lastly, we have verified the trade-off between clean and adversarial performance, especially as we train attacks with larger perturbation spaces. Particularly, we remark that when we include adversarial attacks during training, the performance on clean, historical data is lower.

REFERENCES

[1] Vincent Ballet, Xavier Renard, Jonathan Aigrain, Thibault Laugel, Pascal Frossard, and Marcin Detyniecki. 2019. Imperceptible Adversarial Attacks on Tabular Data. arXiv:1911.03274 [stat.ML]
[2] Bernardo Branco, Pedro Abreu, Ana Sofia Gomes, Mariana S. C. Almeida, João Tiago Ascensão, and Pedro Bizarro. 2020. Interleaved Sequence RNNs for Fraud Detection. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (Virtual Event, CA, USA) *(KDD '20)*. Association for Computing Machinery, New York, NY, USA, 3101–3109. https://doi.org/10.1145/3394486.3403361
[3] Nicholas Carlini and David Wagner. 2017. Towards Evaluating the Robustness of Neural Networks. arXiv:1608.04644 [cs.CR]
[4] Francesco Cartella, Orlando Anunciacao, Yuki Funabiki, Daisuke Yamaguchi, Toru Akishita, and Olivier Elshocht. 2021. Adversarial Attacks for Tabular Data: Application to Fraud Detection and Imbalanced Data. arXiv:2101.08030 [cs.CR]
[5] Jianbo Chen, Michael I. Jordan, and Martin J. Wainwright. 2019. HopSkipJumpAttack: A Query-Efficient Decision-Based Attack. https://doi.org/10.48550/ARXIV.1904.02144
[6] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. *CoRR* abs/1603.02754 (2016). arXiv:1603.02754 http://arxiv.org/abs/1603.02754
[7] Minhao Cheng, Thong Le, Pin-Yu Chen, Jinfeng Yi, Huan Zhang, and Cho-Jui Hsieh. 2018. Query-Efficient Hard-label Black-box Attack:An Optimization-based Approach. arXiv:1807.04457 [cs.LG]
[8] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. arXiv:1412.6572 [stat.ML]
[9] Shreya Goyal, Sumanth Doddapaneni, Mitesh M. Khapra, and Balaraman Ravindran. 2022. A survey in Adversarial Defences and Robustness in NLP. https://doi.org/10.48550/ARXIV.2203.06414
[10] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. 2022. Why do tree-based models still outperform deep learning on tabular data? https://doi.org/10.48550/ARXIV.2207.08815
[11] Masoud Hashemi and Ali Fathi. 2020. PermuteAttack: Counterfactual Explanation of Machine Learning Credit Scorecards. https://doi.org/10.48550/ARXIV.2008.10138
[12] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems* 30 (2017), 3146–3154.
[13] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2017. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083* (2017).
[14] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. 2016. Distillation as a Defense to Adversarial Perturbations against Deep Neural Networks. arXiv:1511.04508 [cs.CR]
[15] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. arXiv:1312.6199 [cs.CV]
[16] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. 2019. Robustness May Be at Odds with Accuracy. arXiv:1805.12152 [stat.ML]
[17] Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric P. Xing, Laurent El Ghaoui, and Michael I. Jordan. 2019. Theoretically Principled Trade-off between Robustness and Accuracy. arXiv:1901.08573 [cs.LG]