TUM School of Computation, Information and Technology
Technische Universität München

Integrity and Correctness of Machine Learning Data

Nicolas Müller

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitz:

Prof. Dr. Gudrun Klinker

Prüfende der Dissertation:

1. Prof. Dr. Claudia Eckert
2. Prof. Dr. Marc-Oliver Pahl

Die Dissertation wurde am 04.07.2022 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 05.12.2022 angenommen.

Abstract

In the last few years, machine learning (ML) has solved many problems where traditional algorithms have failed. However, as a data-driven technology, ML relies on large sets of training data, and only when correct data can be supplied, the model has a chance to learn. Yet, there are several threats to the correctness and integrity of training data: These range from mere labelling mistakes to adversarial attacks, which introduce malicious data into the training set. These attacks can break a model entirely and have already been observed in real-world scenarios.

From this flows the research question motivating this thesis: What influences (i.e., both attacks and data errors) compromise the integrity and correctness of ML datasets, and how can this be counteracted? We structure such threats and present appropriate mitigation strategies:

First, we present the analysis of a severe data artifact we found in arguably the most established deepfake-detection dataset [1]. This artefact has caused the anti-spoof community to systematically overestimate their deepfake-detection models' performance. In this thesis, we propose an approach for identifying such errors in the future. Second, we examine Active Learning, a popular tool for creating labelled data, and identify a previously unpublished adversarial weakness [2] and present a corresponding defence. Third, we present an approach to identifying mislabeled data, caused by negligence or an adversary that changes *random* instances' labels [3]. Fourth, we present defence strategies when such an adversary acts deliberately, i.e. changes *specific* instances' data or labels in order to maximize impact in the context of regression [4]. Fifth, we examine the same scenario in the context of classification learning [5].

Finally, we observe that for all of the threats to ML datasets, the product of occurrence probability and damage (which we call 'riskiness') is approximately constant. Nevertheless, we find that in the scientific community, some of the threats presented in this thesis are significantly understudied. Thus, we argue that there is some unjustified research favouritism and that the research community should not neglect these areas. From this flow our suggestions for future research.

Zusammenfassung

In den letzten Jahren hat das maschinelle Lernen (ML) viele Probleme gelöst, bei denen traditionelle Algorithmen versagt haben. Da es sich jedoch um eine datengesteuerte Technologie handelt, ist ML auf große Mengen von Trainingsdaten angewiesen, und nur wenn korrekte Daten geliefert werden können, hat das Modell eine Chance zu lernen. Es gibt jedoch eine Reihe von Bedrohungen für die Korrektheit und Integrität der Trainingsdaten: Diese reichen von einfachen Beschriftungsfehlern bis hin zu adversariellen Angriffen, die bösartige Daten in den Trainingssatz einbringen. Diese Angriffe können ein Modell vollständig zerstören und wurden bereits in realen Szenarien beobachtet.

Daraus ergibt sich die Forschungsfrage, die diese Arbeit motiviert: Welche Einflüsse (d.h. sowohl Datenfehler als auch Angriffe) können die Integrität und Korrektheit von ML-Datensätzen beeinträchtigen, und wie kann dem entgegengewirkt werden? Wir strukturieren solche Bedrohungen und stellen geeignete Abhilfestrategien vor:

Zunächst stellen wir die Analyse eines schwerwiegenden Datenartefakts vor, das wir in dem wohl bekanntesten Datensatz zur Erkennung von Audio-Deepfakes gefunden haben [1]. Dieses Artefakt hat die Anti-Spoof-Community dazu veranlasst, die Leistung ihrer Modelle zur Erkennung von Fälschungen systematisch und deutlich zu überschätzen. In dieser Arbeit schlagen wir einen Ansatz vor, um solche Fehler in Zukunft zu erkennen. Zweitens untersuchen wir Active Learning, ein beliebtes Tool zur Erstellung von gelabelten Daten, identifizieren eine bisher unveröffentlichte adversarielle Verwundbarkeit [2] und präsentieren eine entsprechende Verteidigung. Drittens stellen wir einen Ansatz zur Identifizierung falsch gelabelter Daten vor, die durch Nachlässigkeit oder einen Gegner zustande kommen, der die Labels von *zufällig* Instanzen ändert [3]. Schließlich stellen wir Verteidigungsstrategien für den Fall vor, dass ein solcher Angreifer vorsätzlich handelt, d.h. *spezifische* Instanzdaten oder -labels ändert, um die Wirkung im Kontext von Regression [4] zu maximieren. Fünftens untersuchen wir dasselbe Szenario im Kontext des Klassifikationslernens [5].

Schließlich stellen wir fest, dass bei allen Bedrohungen für ML-Datensätze das Produkt aus Eintrittswahrscheinlichkeit und Schaden (das wir als "Riskiness" bezeichnen) annähernd konstant ist. Dennoch stellen wir fest, dass einige der in dieser Arbeit vorgestellten Bedrohungen in der wissenschaftlichen Gemeinschaft noch deutlich zu wenig erforscht sind. Wir argumentieren daher, dass es eine ungerechtfertigte Bevorzugung gibt und dass die Forschungsgemeinschaft diese Bereiche nicht vernachlässigen sollte. Daraus ergeben sich unsere Vorschläge für die künftige Forschung.

Contents

List of Figures

List of Tables

Acronyms

AI	Artificial Intelligence.
AL	Active Learning.
ASV	Automatic Speaker Verification.
CNN	Convolutional Neural Network.
DBLP	Digital Bibliography & Library Project.
DNN	Dense Neural Network.
DSGVO	Datenschutz-Grundverordnung.
EER	Equal Error Rate.
GDPR	General Data Protection Regulation.
HMM	Hidden Markov Model.
IMDB	Internation Movie Database.
IWPC	International Warfarin Pharmacogenetics Consortium.
KDE	Kernel Density Estimation.
KNN	k-Nearest Neighbour.
MAE	Mean Absolute Error.
MAE	Mean Squared Error.
MFCC	Mel Frequency Cepstral Coefficients.
ML	Machine Learning.
NN	Neural Network.
PCA	Principal Component Analysis.
RNN	Recurrent Neural Network.
SOTA	State of the Art.
SVM	Support Vector Machine.
SVR	Support Vector Regressor.

TL	Transfer Learning.

1 Introduction

In this chapter, the reader is given a brief overview of this thesis. In section 1.1, we present the research challenge. In the section 1.2, we detail how this thesis addresses the aforementioned research question. Finally, in section 1.3, the impact of our research is discussed.

This chapter does not introduce prerequisites such as basic machine learning knowledge or theory on adversarial machine learning; such introduction is deferred to chapter 2.

1.1 Motivation & Research Challenge

This chapter motivates and presents the research question of this thesis.

The success of machine learning

There are many challenges in the field of computer science which have long been considered too complex to solve algorithmically. Some of these include provably hard problems such as the NP-hard *Travelling Salesman Problem*, for which there exist approximations or size-constrained algorithms [8] which work well enough in practice. However, a great deal of problems has long since eluded both exact and approximate solutions, such as

- agent control in complex, non-stationary environments with limited and even with perfect knowledge (e.g. autonomous driving, StarCraft II, Go, ...),
- classification of text, speech, and visual data (e.g. malware detection, handwriting recognition, radiological diagnosis, speech-detection, spam filtering, ...),
- generative modelling (speech synthesis, image manipulation, ...),
- anomaly detection (identifying credit card fraud, predictive maintenance, ...), and many more.

Yet, in the last few years, machine learning-based techniques have started to tackle many of these problems both effectively and efficiently. Interestingly enough, the underlying machine-learning algorithms themselves have been formulated well before the advent of this millennium. For example, the foundation of the popular neural network architecture has been laid by F. Rosenblatt in 1958 [9]. However, due to a lack of computing power and data, these algorithms could not be applied satisfactorily in practice for a long time.

As of today, the requirement for sufficient computing power is satisfied: The tremendous advances in the design of semiconductors allow to readily obtain powerful hardware for little money.

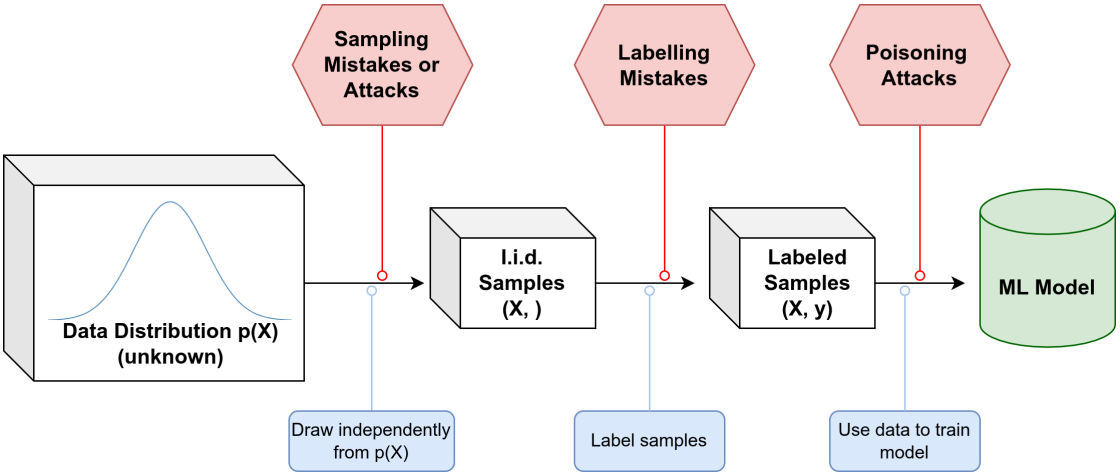


Figure 1.1: A diagram illustrating how new supervised ML-datasets are created. Given some underlying data distribution $p(X)$, n samples are drawn independently and randomly. This yields an unlabeled dataset X. This dataset is then labelled, either by a human annotator, a crowd working platform such as Amazon's Mechanical Turk, or even (semi)-automatically, depending on the data. Finally, the labelled data (X, y) is used to train a model. As shown in the red boxes, the dataset can be compromised during all of these stages.

The need for training data

Thus, data becomes the bottleneck for training quality machine learning algorithms. The best algorithm is useless when data is too small or too bad in quality.

However, compared to twenty years ago, data has become more easily available through the ubiquity of digitization. For popular use cases, large and high-quality datasets can just be downloaded in high quality for free (e.g. for handwritten digit recognition [10] or object detection [11]).

However, when the use case is not as mainstream, training data often is not available that easily. It has to be collected from a variety of (untrusted) entities, scraped from the web, or even created manually by human annotators, for example by student workers (as we did in [12]) or using crowd-labelling techniques such as Amazon Mechanical Turk. The many moving parts and external factors make the resulting dataset susceptible to a variety of errors and attacks, as we'll explain in the following section.

The Correctness of Training Data is Paramount

We now turn our attention to this process and shortly describe how ML datasets are created. Figure 1.1 provides a flowchart that presents the dataset creation steps in blue and the various intermediate results in white. First, some underlying distribution is queried for random samples. Then, the resulting data is stored and labelled. Finally, it is used to train a model. In practice, the drawing from the underlying distribution $p(X)$ is realized by collecting evidence in the real world, i.e. measuring or observing the features

in question. However, there are several avenues for compromising this data, as indicated in Figure 1.1 in red:

- **Sampling mistakes**. First, the data may be inconsistent due to sampling mistakes. This is a more subtle error than labelling mistakes, since the analysis of $(X,)$ or even (X, y) alone cannot catch this error. Usual sanity checks such as the train/test split do not work in this context (since the sampling incongruencies affect all data, i.e. both train and test data). Rather, a human expert is required who compares the data-generating distribution $p(X)$ (which humans often have a pretty good idea about) to the drawn samples $(X,)$.

  Thus, while hard to detect, this kind of error can have extreme consequences: Users or even whole research communities who rely on incongruent data may systematically overestimate the performance of their machine-learning models, which perform poorly on separate, real-world test data, c.f. chapter 3.

- **Sampling attacks**. Second, an adversary may also compromise auxiliary data-creation pipelines such as active learning. This corresponds to attacks during the sampling stage. This results in a poisoned dataset where the adversary has full control over the injected instances, and model performance is severely diminished (c.f. Chapter 4).

- **Labeling mistakes**. Third, the data may contain mislabeled instances, i.e. mistakes due to negligence on the human annotators' side. Data annotation is a monotonous task, and errors (i.e. assigning label '0' instead of '1') can easily happen. As we will show in Chapter 5, even professional data sets such as Fashion-MNIST by Zalando [13] contain a considerable fraction of instances whose labels are just plain wrong. One may envision the quality of data collected by individuals with fewer resources.

- **Adversarial poisoning attacks**. Finally, the labelled data may be compromised by an adversary. This means that some third parties may have deliberately introduced instances with wrong labels into the dataset. Why would an adversary want to perform such an attack? Adversarial data poisoning makes it possible to either a) break the model, or b) control the model. This allows an attacker to achieve their goal, such as financial gain, unauthorized access, the exercise of revenge, etc. We will discuss this in more detail in Section 2.2.1. For now, suffice it to say that adversarial data poisoning is one of the two major threats to machine-learning systems; the second one being test-time 'adversarial evasion attacks', c.f. Section 2.2.1.1, which is not in focus in this thesis.

Research Question

These considerations thus raise the following research question: What influences (i.e., both attacks and data errors) compromise the integrity and correctness of ML datasets, and how can this be counteracted? Here, integrity means 'free from attacks', while correctness means 'free from errors'.

1.2 Research Contribution

The contribution of this thesis is to answer the aforementioned research question. We do this by examining the most relevant failure points in the ML-data creation pipeline, c.f. figure 1.1, and examining appropriate mitigation. Concretely:

- First, we present an analysis of a severe sampling artifact in the ASVSpoof 2019 dataset, c.f. chapter 3.
  - We find that the ASVSpoof-community has been systematically overestimating the performance of their spoof-detection (colloquial: audio-deepfake detection) models. This is due to a severe data artifact: We show that the length of the leading silence strongly hints at the target label (spoof or authentic).
  - We show [1] that because of this, most related work from the last three years has considerably overestimated the generalization capabilities of audio-spoof detection.
  - We present an algorithmic approach to detecting this kind of error in the future. We implement a corresponding prototype and show its applicability.
- As the second part of our contribution, we examine active learning, a popular set of auxiliary techniques and algorithms used to maximize efficiency when creating a labelled dataset, c.f. chapter 4.
  - We show that algorithms previously thought unaffected by adversarial attacks (active learning algorithms) are indeed highly susceptible.
  - We suggest, discuss and evaluate appropriate defence strategies [2], thereby showing how to apply these algorithms in practice while minimizing exposure and susceptibility to adversarial poisoning attacks.
- Third, in Chapter 5, we examine mislabeled instances. Specifically:
  - We present a novel algorithm [3] to identify mislabeled instances in classification data and thoroughly evaluate this algorithm on 29 data sets.
  - We apply this algorithm to the most popular machine-learning data sets (e.g. CIFAR, MNIST, Fashion-MNIST, Twenty-Newsgroup) and find significant quantities of real-world mislabeled data which have not been described before.
- The final part of our contribution is two algorithms to identify adversarial data in regression and classification data, respectively (c.f. Chapter 6 and Chapter 7). This includes the design of a new attack algorithm, which we use in addition to existing attacks to evaluate our proposed defence. Specifically:
  - We present a new defence against data poisoning on regression learning [4].
  - We present a new defence against data poisoning on classification learning [5].

  Both defences improve over related work in several ways; but most notably, they are designed to be applicable in practice, i.e. without the need for a controlled environment.

1.3 Research Output

The results of this thesis w.r.t. research output can be summarized as follows:

- **Publications**. During the course of writing this thesis, eight publications have emerged. These can be separated into those that lead towards the research question motivating this thesis [14], [12], those that supply core research w.r.t. the research question [3, 4, 5, 2, 1] and finally, work that rounds off the topic of adversarial machine learning [15].
- **Source code and tooling**. For some of these works, source code has been released [3]. This tooling has already been used by other researchers and has been extended with follow-up ideas, such as Atkinson et Al. [16]. Also, a dataset on GDPR requirements for privacy policies has been built, vetted, and published online [12].
- **Identification of mislabeled instances**. We identify mislabeled instances in highly popular, real-world datasets such as CIFAR10, MNIST, etc. These instances contribute to understanding why even the most sophisticated model does not achieve a 0% error rate [17] in these benchmarks.
- **Impact on the anti-spoofing community**. Our discovery of the data artifact in the ASVSpoof 2019 and 2021 datasets has had an impact on the anti-spoofing community. A reviewer summarizes: 'This work could be considered controversial and it can make us question part of the work presented so far.' After the acceptance of the publication [1], the community has been informed via the official conference summary presentation (pages 9 and 14 in [18]). The conference organizers re-evaluated all of the official baselines, both with and without the presence of the artifact, and verified our results. It has been announced that this topic will be given more consideration in future ASVspoof conferences.

2 Prerequisites

This chapter provides an introduction to machine learning and adversarial attacks, concepts that are essential to understanding the rest of this thesis. This chapter does not present novel research.

2.1 Artificial Intelligence: An Overview

2.1.1 Algorithmic vs. Data-Driven Problem Solving

There are two fundamental approaches to telling a computer program how to solve a given problem.

- **The algorithmic approach.** In many cases, it is possible to provide a sequence of finitely many computer-implementable steps, i.e. an algorithm. For example, given a graph consisting of vertices and nodes, Dijkstra's algorithm [19] is guaranteed to find the shortest path between two given nodes in a finite number of steps. This algorithm has been proven to be correct and can be implemented efficiently. It is a perfect example of a problem where applying artificial intelligence does not make sense.

  However, there is another class of problems for which algorithms struggle to find solutions efficiently. Let us examine the toy problem of classifying images of animals, for example, dogs vs. cats. Writing down an exact rule set to differentiate between these two is not trivial: First, there is the problem to extract semantic features from a given image, such as the size of the animal, colour, and the presence of features such as whiskers, snout, etc. Second, given a set of features, it is not clear which values uniquely *constitute* a cat: Is it always smaller than a dog, or leaner, or of a certain colour, or does it have a rounder snout, ...? We see that it is very hard to define an exact set of rules, and thus an algorithm for this problem. Note that this is not a problem of ambiguity (humans usually have no problem telling a dog from a cat), rather, the difficulty lies in defining and separating the feature space.

- **The data-driven approach.** The difficulty in solving such problems with traditional methods and algorithms leads to the second approach: the data-driven approach. In this approach, there is no explicit rule set or sequence of steps to derive a solution. Rather, we provide a set of examples (in this case, a set of images of cats and dogs) and, by clever application of statistics, let the computer figure out which is which. Before diving into how such a 'clever application' is implemented, we briefly summarize the advantages and disadvantages of both approaches. We will see that they are complementary; where one is weak, the other is strong.

2.1.1.1 Comparison

The advantages of algorithmic approaches are as follows. First, they are usually understood in their entirety. We can compute all relevant properties such as run-time, memory consumption, and quality of solution exactly or at least provide lower or upper bounds. Second, algorithmic solutions are reliable and interpretable; one can often prove their correctness and understand exactly how they work. Third, no training data is required. On the other hand, the disadvantages are as follows: First, for many problems, computer science has struggled to find algorithms that solve the problem satisfactorily. This is because either we have simply not found a suitable algorithm, or we cannot formulate the problem in a suitable (mathematical) way. Remember our example: how to define the difference between images of cats and dogs? Second, algorithms tend to miss edge cases. Any special case that the designer of the algorithm has failed to integrate may result in erratic behaviour.

The data-driven approach excels where traditional approaches fail: problems do not have to be formulated explicitly, and if enough data is supplied, edge cases are learned by the model naturally. Classifying dogs vs. cats becomes easy, state-of-the-art achieves an accuracy of 99.70 percent [20]. On the other hand, data-driven solutions can behave unpredictably (an extreme example of this are 'adversarial attacks', see Section 2.2). Additionally, their solutions are usually approximate, and one cannot prove correctness nor has satisfactory model explainability. Also, the process of developing data-driven approaches can feel like a series of 'educated guesses' as opposed to the rigorous application of statistics. Finally, a big downside is a need for sufficient and correct training data - an issue we hope to at least partly alleviate with this thesis.

2.1.2 Distinguishing AI, Machine Learning, and Deep Learning

This section provides a distinction between artificial intelligence, machine learning, neural networks, and deep learning. We feel that such a recapitulation is necessary since these terms are often confounded.

Figure 2.1 shows a Venn diagram detailing the relationship between the field of artificial intelligence as a whole, machine learning, and neural networks. The outermost category represents the field of AI. Machine learning is a subset of this discipline. It is concerned with data-driven approaches. However, there are other areas within the field of artificial intelligence that do not employ the data-driven approach, such as Automated Planning and Scheduling (AI Planning). For example, 'state-space search' is the process of finding the states of an instance with the desired property. This problem can be encoded in a graph, and techniques such as breadth-first-search or depth-first-search may be employed to algorithmically derive a solution. In summary, machine learning does not equal artificial intelligence, even though the terms are often used interchangeably. Rather, machine learning is the branch or subset of artificial intelligence that is interested only in data-driven approaches (to learn from data, thus the name).

Finally, there are several different tools and techniques to extract information from training data. These techniques differ in goal and applicability, but all share the underlying

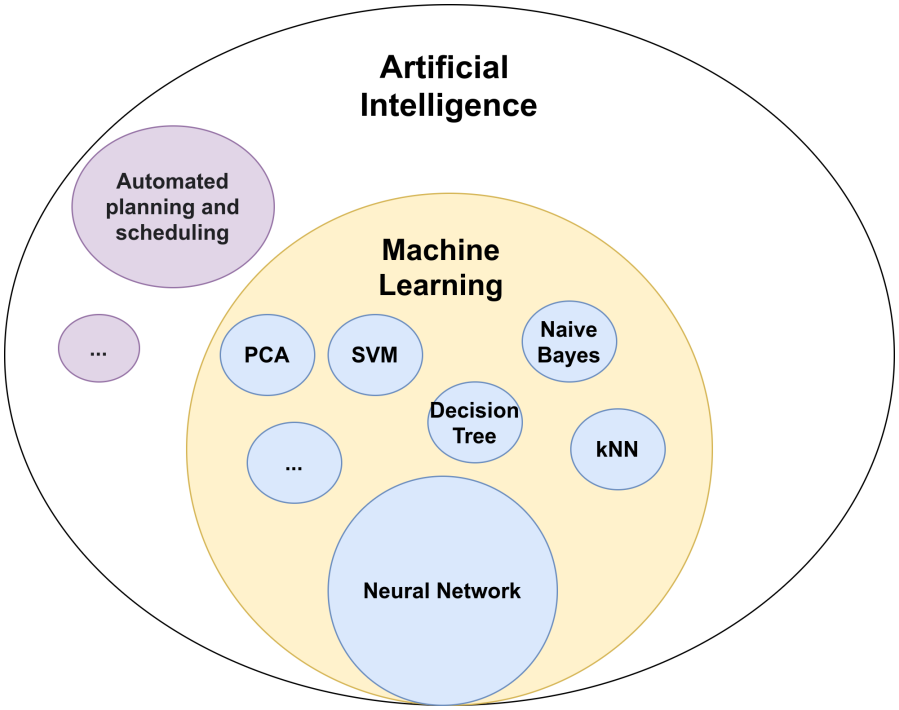


Figure 2.1: A Venn diagram illustrating the relation between the field of AI, machine learning, and neural networks.

paradigm of data-driven learning. Neural Networks, Support Vector Machines (SVM), Principal Component Analysis (PCA), etc. are just a few examples of these tools. The field of computer science which is concerned with Neural Networks is called *Deep Learning*. It is a strictly data-driven discipline.

2.1.3 Regression vs. Classification Learning

In the domain of supervised learning, we have datasets (X, y), where X is the data and y are the features. The data consists of a set of features, i.e. observable properties, such as the pixels of an image, or numeric measurements of some sort. The labels are scalar values that indicate what the class *means*, i.e. what the image or the numeric measurements represent. We distinguish between *classification* and *regression* learning:

In classification, the targets are integer values, representing class membership. For example, instances in an image dataset may be labelled as either *cat*, *dog* or *horse*. We can represent classes as integers $y \in \mathbb{N}$. Note that there is no sensible distance metric on the labels since *cat* is not more or less similar to *dog* than to *horse*.

Another way to interpret the targets y is *regression learning*: Here, they are real-valued scalars $y \in \mathbb{R}$. For example, a dataset of historical stock market data could be labelled with the current stock price, in an attempt to predict future stock prices.

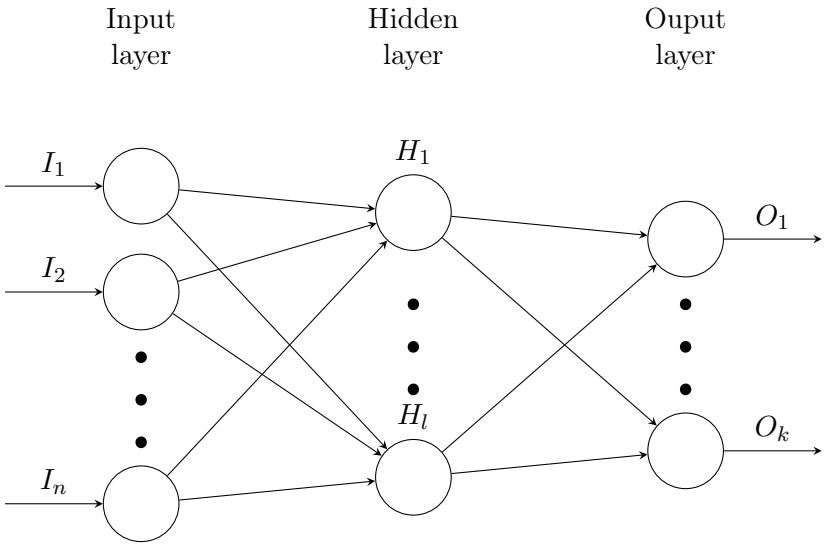


Figure 2.2: A feed forward neural network with two layers (not counting the input layer).

Regression and classification share many architectural and algorithmic properties, such as the model structure and learning algorithm (backpropagation). However, there are several key differences, for example in the domain of loss functions. For more, see [21].

2.1.4 Neural Networks

A very popular machine learning model is the neural network. First outlined by Rosenblatt [9], it gained considerable traction with the advent of big data and massive computing power. Motivated by biological neurons in the human brain, it consists of a sequence of neural layers. Figure 2.2 provides a schematic. It shows a neural network with three layers: One input layer I, one middle layer H, and one output layer O. The layers which are not the input or output are called *hidden* layers. All layers are connected sequentially, which is why this kind of neural network is called a *feed-forward* network. Input x can be processed if it has the dimensionality of the input layer I, i.e. if $dim(x) = n$. The output then has dimensionality $dim(O(H(x))) = k$.

The arrows between the layers depict the *weights* of the neural network. These *weights* denote how input is changed during the transition from one layer to the next. Each arrow corresponds to one real-valued scalar weight, i.e. $m_{ij} \in \mathbb{R}$. The input x is only changed in transition from one layer to the next. Thus, one commonly omits the input layer when counting a model's layers, since it does not process the input, but only reads it in. This is why the network in Figure 2.2 is said to have two layers.

One can describe a neural network formally. Assume the input layer is n-dimensional, and the output layer is l-dimensional. The weights can be expressed as a real-valued matrix of shape $M \in R^{n \times l}$, and input and output are represented as vectors. A feed-forward

pass of input $x \in \mathbb{R}^{1\times n}$ is then expressed by matrix multiplication

$$y = xM \tag{2.1}$$

which is dimensionally correct:

$$dim(y) = dim(xM) = (1, n) \times (n, l) = (1, l) = dim(y) \tag{2.2}$$

Thus, the application of a feed-forward layer is a linear transformation (as defined by its matrix M). The subsequent application of several linear layers is thus also linear, since the matrices $M_1, ..., M_t$ 'collapse' to a single matrix Z, as is shown in Appendix A.1. Thus, to gainfully apply more than one layer, one has to introduce non-linearity between layers to avoid this 'collapse'. This is the task of *activation functions*, which are non-linear, differentiable functions c which operate element-wise on the output of a given layer:

$$y = c(Mx) = [c(Mx)_0, ..., c(Mx)_l]^T \tag{2.3}$$

Popular activation functions include

- Rectified Linear Unit: $ReLU(x) = max(0, x)$,
- Hyperbolic Tangent: $tanh(x) = 2/(1 + e^{-2x}) - 1$,
- and Sigmoid: $\sigma(x) = 1/(1 + e^{-x})$.

Once the network architecture is fixed, suitable entries m_{ij} need to be found for each matrix M or layer in the neural network. For more complex networks, the trainable entries include not only the weights of the matrices but also biases (constant offsets) and other trainable parameters such as in Batch Normalisation [22]. In literature, the entirety of the trainable parameters is commonly referred to as θ.

To find these weights θ, one needs to mathematically define what *suitable* means, i.e. one has to provide some measure to quantify the network's performance. For example, when predicting, say, the stock price for a given value, a suitable measure would be the magnitude of the difference in predicted price and actual price, i.e. $|y_{pred} - y_{true}|$, which we usually abbreviate to $|y' - y|$. Such measures by which we quantify and subsequently optimize the model are called *loss functions*. They denote how 'far off' the prediction y' is from the ground truth y. Thus, one usually aims for minimizing this loss: Lower loss equals a better model. Some of the most common loss functions include

- L1 Loss: $L(y, y') = |y - y'|$
- L2 Loss: $L(y, y') = (y - y')^2$
- Cross-Entropy-Loss:
$$L(y, y') = -\sum_i y_i \log(y'_i) \tag{2.4}$$
where p is a probability density function such as softmax
$$\text{softmax}(x) = \frac{e^{x_i}}{\sum_i e^{x_i}}. \tag{2.5}$$

The weights of a neural network can then be found by stochastic optimization [23], where for each pair (x, y) in the training data, we obtain the model's prediction $f(x) = y'$ and then minimize the loss between the predicted output y' and the true output y. Formally:

$$\theta_* = \arg\min_{\theta} L(f_\theta(x), y) \tag{2.6}$$

where θ are the model weights. Since solving Equation (2.6) is intractable even for shallow neural networks, approximation techniques such as *Gradient Descent* are used. It consists of iteratively optimizing a randomly initialized set of weights θ_0 towards the optimal solution θ_* by computing the gradient of the loss concerning the model's parameters θ_i at timestep i, and then updating the weights accordingly. Formally:

$$\theta_{i+1} = \theta_i - \alpha \nabla_{\theta_i} L(f_{\theta_i}(x), y) \tag{2.7}$$

Here, f is the neural network (parameterized by θ_i at timestep i), L is the loss function, and (x, y) is a training sample consisting of input x and observed output y. The gradient with respect to the parameters θ is represented by the *Nabla* operator ∇_θ. It is a vector-valued operator with dimensionality $T = dim(\theta)$, i.e. with as many entries as there are model parameters. The gradient $\nabla_\theta L$ can be understood as the direction and magnitude of the steepest ascent of L concerning the model parameters θ_i. Put differently, this means that the gradient points toward the set of parameters which *maximize* the loss for the current training sample (x, y). Formally:

$$\nabla_\theta L(f(x)) = \begin{bmatrix} \frac{\partial L}{\partial \theta_0}(f(x)) \\ \vdots \\ \frac{\partial L}{\partial \theta_{T-1}}(f(x)) \end{bmatrix} \tag{2.8}$$

where $\frac{\partial L}{\partial \theta_i}(f(x))$ is the partial derivative of L with respect to θ_i. Note that from now on, we will write f instead of f_θ when there is no ambiguity as to the set of weights that parametrize f. Finally, the model is updated with the newly learned information as encoded in the gradient ∇L, which is multiplied by -1 (because we want to minimize rather than maximize the loss). Thus, $-\nabla L$ is added to the current set of parameters, scaled by a constant α called the *learning rate* (c.f. Equation (2.7)).

Both the loss and each component of the neural architecture must be differentiable since otherwise the gradient cannot be computed because the partial derivates are not defined. Training a model for i steps then consists of applying equation 2.7 sequentially i times. Deep neural networks such as Tacotron [24] can require more than 100,000 steps, which takes days on modern hardware. The number of training steps required varies from model to model and depends on the complexity of the network as well as the number of model parameters. Some extremely large models have over 175 billion trainable parameters, pushing training costs well into the millions of dollars [25].

2.1.5 The Limits of Machine Learning

Although neural networks excel in many fields and even show superhuman performance [26] in some areas, their capabilities are sometimes overestimated. This chapter describes what deep-learning techniques can and cannot do.

Neural networks excel at tasks that are based on *pattern recognition*. Whenever there is a problem where a mapping $f : A \to B$ is to be found (mapping images to animal names; mapping a spoken sentence to its transcribe; mapping a chess position to the next optimal move), deep learning can reliably find this mapping f, provided

- training data of sufficient size and quality and
- a suitable, mathematical way to quantify the quality of the network's output (i.e. a loss function) are available.

Missing training data can usually be acquired (although possibly at a high expense). But often, there is no way to formulate the loss function. For example, how would one formally describe the task of composing a 'good short story', an 'interesting piece of music' or a 'beautiful poem'? Even in more objective domains such as law, how would one mathematically quantify the degree to which, for example, a given statement complies with a given law, or a certain website complies with the General Data Protection Regulation (GDPR)? These problems are very hard for AI to solve because they require more than just pattern recognition skills: They require real, human-like *understanding* of the object, which currently we do not know how to instill into artificial intelligence.

Thus, the AI we have today is called *Weak AI*: Weak AI is artificial intelligence with a narrow field of application, trained for one specific task. It is based on pattern recognition. It can understand *semantics*, i.e. the meaning of data (such as the content of an image, the meaning of the text, etc.) only via statistical and correlation analysis: meaning is inferred not by *understanding*, but, at its core, by co-occurrence of observables (for example, the fact that cows tend to stand on green grass and not on water is understood simply by observing the co-occurrence of cows and grass in the dataset, and not by 'understanding' the inherent properties of grass as a solid surface as opposed to non-supporting water surfaces).

In contrast, *Strong AI* would be artificial intelligence based on *understanding*, with a broad range of applications, mirroring the intelligence of a human. All current AI, however sophisticated, belongs to the first category.

2.2 Adversarial Machine Learning: An Overview

This section provides an overview of Adversarial Machine Learning, which concerns itself with the security of machine learning. We first give an introduction about how attacks on machine learning work technically, then explain why an attacker might want to compromise a model, and finally present current limitations of adversarial machine learning.

2.2.1 Evasion and Poisoning Attacks

This section introduces the two most important attacks on machine learning: evasion and poisoning attacks.

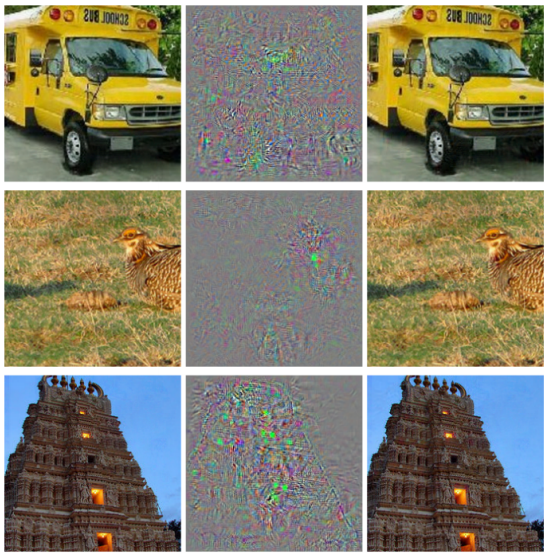


Figure 2.3: Figure from [6], showing three adversarial images $x + \delta$. The leftmost row shows x, and the middle row presents the adversarial noise δ, and the rightmost row shows $x + \delta$. Even though humans do not perceive a difference, a neural network will assign different classes for x and $x + \delta$.

2.2.1.1 Evasion Attacks

Evasion Attacks are attacks on an already trained machine learning model. They do not alter the trained model; instead these attacks create a malicious input $x + \delta$ such that the model f classifies x and $x + \delta$ differently:

$$f(x) \neq f(x + \delta) \tag{2.9}$$

δ is called the adversarial noise. This adversarial noise is what causes the model f to change its prediction of x. It always has the same dimensionality as x, since it is added to x. To be imperceptible for humans, it has a low magnitude, i.e. small vector norm.

For example, assume the input to be an image, i.e. $x \in \mathbb{R}^{h \times w}$. In this scenario, h and w are the height and width of an image, for example $h = w = 32$ for the CIFAR dataset [11]. From this it follows that $\delta \in \mathbb{R}^{h \times w}$. Figure 2.3, taken from Szergedy et al. [6], shows an example of this. Three example images are presented (left column): a bus, a bird, and a temple. A neural network trained for image recognition classifies these correctly and with high confidence. The middle row now presents adversarially crafted noise δ (enlarged to be visible). When added to the original images, we see that sum $x + \delta$ is virtually imperceptible (right column). For humans, x and $x + \delta$ show the same object. A neural network, however, is fooled: It now classifies all three images in the rightmost column as 'ostrich'.

Adversarial noise is found via back-propagation and gradient descent, similarly to Equation (2.7), but instead of optimizing with respect to the model's weights, one optimizes

with respect to the adversarial noise δ:

$$\delta_{i+1} = \delta_i - \alpha \nabla_\delta J(f_\theta, x, \delta_i, y_{target}) \tag{2.10}$$

where again $\alpha \in \mathbb{R}$ is the learning rate, f_θ the trained neural network, x the input image and y_{target} the target class, in the case of Figure 2.3 an 'ostrich'. The loss function J consists of a loss function L similar or identical to the one used to train the neural network, plus weighted regularisation on the noise δ. For example:

$$J(f_\theta, x, \delta_i, y_{target}) = L\big(f_\theta(x + \delta_i), y_{target}\big) + \gamma \|\delta\|_2 \tag{2.11}$$

$$= \big(f_\theta(x + \delta_i) - y_{target}\big)^2 + \gamma \|\delta\|_2 \tag{2.12}$$

where we assume L as $L2$ loss. The scalar value $\gamma \in \mathbb{R}$ adjusts the strength of regularisation. Thus, adversarial examples are found in the following process:

1. Assume a trained neural network f_θ and a 'starting image' with its correct classification (x, y).
2. Define initial $\delta_0 = 0$. For images: $\delta_0 = 0^{h \times w}$. Alternatively, one may initialize δ_0 with random values.
3. Choose a target class $y_{target} \neq y$.
4. Iteratively apply Equation (2.10) until convergence, i.e. $\delta_{i+1} - \delta_i| < \epsilon$ for arbitrary $\epsilon \in \mathbb{R}$.
5. Let $\delta := \delta_{i+1}$ and yield the adversarial example $x + \delta$.

Intuitively, a single step of gradient descent as described in Equation (2.10) finds the noise δ_i which, when added to x, makes the input $x + \delta_i$ seem more like the target class y_{target} for the neural network f_θ. Additionally, the noise δ is kept low in magnitude (due to the regularisation, c.f. Equation (2.11)) and thus remains undetectable for a human. In summary, adversarial evasion attacks are attacks on machine learning models where a trained model is fooled by an adversarial image.

2.2.1.2 Poisoning Attacks

Data poisoning attacks are the second branch of adversarial attacks on machine learning and motivate a significant part of research in this thesis. In contrast to evasion attacks, data poisoning attacks are executed at training time. Malicious instances (x_p, y_p) are inserted into the training data set, which causes the model to either degenerate (denial of service poisoning attack) or allows an attacker to control the model at test time (backdoor poisoning attacks). In contrast to evasion attacks, poisoning attacks change the model. This motivates the name: Similar to how poison damages the human body, adversarial data poison samples damage the machine learning model.

Since poison attacks are less straightforward than evasion attacks, we introduce them in several steps: First, we compare them to evasion attacks. Second, we introduce them formally. Third, we outline how they are created. Fourth, we give insight into how an

	Evasion Attack	Poisoning Attack
When	Attack at test time	Attack at training time
Impact	Does not change the model	Changes the model
Result	Attacker controls model with malicious input	Attacker controls model with benign input, or DoS
Requires	Test time access	Training dataset access

Table 2.1: Key differences between Adversarial Evasion Attacks and Data Poisoning Attacks.

attacker might succeed in introducing adversarial poison samples into a defender's data set.

Comparison to evasion attacks. As stated above, data poisoning attacks are attacks at train time that change a model by introducing malicious instances in the training set, which deteriorate any model that is trained on them. Table 2.1 highlights key differences to evasion attacks. Which attack is easier to perform in practice depends on the capabilities of the attacker: It may be hard to introduce data in the training pool, but easy to submit poisoned samples at test time or the other way around. These considerations motivate the appropriate choice of attack.

Formal description of data poisoning. Given a model f_θ which is to be trained on a dataset $D = \{(x_i, y_i)\}$, a data poisoning attack consists in finding a set of poisoned instances $D_p = \{(x_j^p, y_j^p)\}$ such that a model f_θ trained on $D \cup D_p$ is compromised in one of the following ways:

- **Denial of Service.** In this scenario, the trained model will perform poorly on test data, i.e. the test-loss $L(f_\theta(x), y)$ for $(x, y) \in D_{test}$ will be high. Put differently, the model will not have learned a useful mapping from input to output, and can thus not be used for its designated purpose. However, the defender may not notice the high test loss since the test set may also be poisoned (usually, the test set is just a portion of data taken from the initial training data set). But even if they notice: To make the model usable again, they need to be able to remove the malicious instances from the data set, which usually is not trivial.

- **Backdoor Attack.** In this scenario, the trained model will perform correctly on a validation set D_{val}, i.e. incur small test loss. However, for very specific instances (x, y), which may or may not be present in the defender's dataset, $f_\theta(x) = z \neq y$. Put differently, the data poisoning attack allows the attacker to control the model for a few, specifically chosen instances.

How to create poison samples. The creation of poison samples is technically not as straightforward as the creation of evasion attack samples. This is because one needs to take the training process into consideration, which is completely irrelevant in the context of evasion attacks. Where evasion attacks need to compute the gradient of the model's output with respect to some noise δ, poisoning attacks would need to compute how changes in a given poison sample (x^p, y^p) affect the training process - which itself is a sequence of optimization steps as described in Equation (2.7). As exact computation is intractable, heuristics are used. We defer the presentation of these to Chapter 6 for regression learning and Chapter 7 for classification learning, where we will present different

attacks, implement them and use them in order to have strong baselines against which to evaluate our proposed defences. For now, it suffices to know that there are both white-box and black-box data poisoning attacks, i.e. attacks that either do or do not require access to the model f.

Feasibility of data poisoning attacks. Finally, we examine how an attacker may succeed in introducing poisoning data into the training set. When using public data sets such as MNIST [27] or CIFAR [11], the threat of an attacker compromising the data set may seem unrealistic: After all, the data is vetted by the open-source community and downloaded 'en block' - there is little room for manipulation. However, when data is not as mainstream as MNIST or CIFAR, it cannot be acquired as easily. It has to be bought from third parties, scraped from the web or a variety of sensors, or needs to be collected by a group of individuals who all own a small portion of relevant data. In each of these cases, an adversary can smoothly introduce malicious data. Thus, the threat of data poisoning is very real, especially for entities such as small and medium enterprises (SME) with smaller pools of resources, which have to rely on possibly untrusted third parties to acquire a sufficiently sized data set.

2.2.2 Related Work and State-of-the-Art

In this section, we give a short overview of existing literature on data poisoning attacks. Related work on defences will be presented when we introduce our proposed defences. Some passages in this chapter may be cited verbatim from our seminar papers [5, 4].

2.2.2.1 Data Poisoning Attacks in Classification Learning

Early work on data poisoning attacks against classification learning is presented by Biggio et al. [28] and Xiao et al. [29], which use the Karush-Kuhn-Tucker conditions to find optimal poisoning samples against linear models. Biggio et al. [28] were the first to develop a poisoning attack against SVMs. Additionally, Xiao et al. [13] contribute by evaluating the security of feature selection against poisoning attacks. They adapt the approach to LASSO, Ridge Regression, and Elastic Net. In both scenarios, the attacker attempts to increase the test error and, thus, decrease the overall performance of the classifier.

Munoz-Gonzales et al. [30] are the first to extend data poisoning to the multi-class scenario, which allows for targeted attacks. Instead of the Karush-Kuhn-Tucker conditions, they use back-gradient optimization to generate the first poison samples for neural networks in an end-to-end fashion without the need for a surrogate model.

Shafahi et al. [31] build upon the work of Munoz-Gonzalez et al. [30] and demonstrate reliable clean-label attacks, in which the attacker can control the input data x, but not the corresponding labels y. In transfer learning scenarios, the authors show the effectiveness of a single poison sample. In end-to-end learning settings, i.e. when training the complete model, they develop a watermarking approach to poisoning.

Two data poisoning attacks from related work are presented in Section 7.4, where we implement them and evaluate them against our proposed defence. Thus, we defer a more in-depth analysis for now.

2.2.2.2 Data Poisoning Attacks in Regression Learning

Data poisoning has so far been examined almost exclusively for classification learning. For regression learning, there is work only by Jagielski et al. [7]. They build upon work by Xiao et al. [29], who introduce a gradient-based optimization attack for linear classifiers such as Lasso, Ridge Regression, and Elastic Net for feature selection. Jagielski et al. [7] use the same approach for the same models, but interpret the model's decision surface as a predictor for the continuous target variable, yielding a poisoning attack for linear regression. Additionally, they introduce a non-gradient-based attack, plus a defence called *Trim* and evaluate it on three datasets. Their approach in evaluating the defence is, however, not applicable in practice, since they use an oracle to determine the defence's hyperparameters. More specifically, they assume they know the fraction ϵ of poisoned samples in the dataset of size n, which is generally unknown. We elaborate on this in Chapter 6.

Nonlinear regressors such as Kernel Ridge, Kernel SVM, and Neural Networks have, to the best of our knowledge, not yet been examined in the context of adversarial poisoning. This may be because the attack presented by Xiao et El. [29] is not applicable to nonlinear learners.

Finally, note that there are a few regressors that are inherently designed to resist adversarial data poisoning. One example is the Huber Regressor, which uses a combination of $L1$ and $L2$ loss to mitigate the effects of outliers. While these models are effective in dealing with data poisoning (c.f. Table 6.1), they are limited in their applicability, because they are linear models.

2.2.3 The motivation for attacking Machine Learning

In order to motivate why adversarial attacks are a realistic threat, it is necessary to illustrate that not only the attacks are feasible, but also profitable for the attacker. In this section, we give some examples of real-world use-cases when attacking machine learning.

- **Autonomous Cars.** In 2017, Evtimov et al. [32] showed that image detection for road signs can be adversarially broken under real-world conditions. They crafted a set of stickers, consisting of white and black squares, which cause misclassification when affixed to traffic signs. Figure 2.4 provides an example. Groups that may be motivated to break an autonomous car's traffic sign recognition model include terrorists, activists, or competitors.

- **Face recognition.** Breaking face recognition [33] allows an attacker to gain unauthorized access (to someone's iPhone, Car, or to a facility), or to evade surveillance.

Figure 2.4: An adversarially perturbed stop sign by Evtimov et al. [32], which is classified as 'Speed Limit 45mph'.

- **Stock price manipulation.** Just as conventional attacks with explosives are employed by individuals to manipulate a company's stock [34], adversarial attacks may be employed to damage a company and thus manipulate the price of shares.
- **Medical use cases.** Finlayson et al. [35] provide an interesting use case, where adversarial machine learning may be used by a physician to evade the insurance company's predictive model, allowing access to a greater range of narcotics.
- **Fraud detection.** Additionally, adversarial machine learning may be used to evade fraud detection systems. A malicious adversary can thus modify the features of their fraudulent activity to hide it from a fraud detection system [36].
- **Spam and Malware detection**. Adversarial machine learning may also be used to disguise spam and malware as legitimate by introducing specific phrases or bit patterns, which are designed to fool the detection algorithm [37], [38].

2.2.4 Current Limitations of Adversarial Attacks

There are a few areas where adversarial machine learning struggles. First, these include use cases where the adversarially perturbed sample has to bridge the 'air gap', i.e. where the adversarial sample cannot be supplied to the model directly. One such example is speech recognition, where the user speaks into a microphone, and AI transcribes or executes a given command. Adversarial attacks are challenging in this domain since they have to 'bridge the air gap': The noise δ in the adversarial sample $x + \delta$ must be both small enough to be imperceptible, but robust enough to remain effective in the presence of channel perturbations that come with speaking and recording an utterance (such as microphone bumps, reverberations, delay, ..). While a satisfactory solution to this problem has not yet been found, our research [15] and others [39, 40] explore these limitations. Note that in other fields, such as image recognition, this air gap has already been bridged [41].

Second, active learning has so far remained robust to adversarial attacks. In this domain, there is only little related work, which assumes very high capabilities of the attacker to successfully compromise a model [42].

Third, it remains challenging to find optimally effective data poisoning samples, especially in a black-box scenario. While there is success in this regard for evasion attacks [43], a solution remains elusive for poisoning attacks. We will elaborate on this extensively in Chapter 6 when we introduce the current state-of-the-art.

3 Artefacts in ML-Datasets

A first challenge when creating a machine-learning dataset is the correctness of the dataset w.r.t. 'data artifacts' that can lead to so-called 'learning shortcuts'. Such an artifact is present when features F_i in the data are predictive of the target T but do not carry the desired semantics. Consider the top-left image in figure 3.1, which shows the Pascal VOC 2007 image classification dataset. In this dataset, all images of horses happened to contain the photographer's tag in the bottom left corner. Machine learning models trained on this data would learn that feature $F_i = <Tag\ in\ Image>$ is 100% predictive for target $T = <horse>$. A model which relies on such features does not need to learn the true semantics of a horse and thus will perform poorly in a real-world scenario. A 'learning shortcut' has occurred and made the model unsuited to any practical application.

In this chapter, we analyze this phenomenon further. We present a previously unidentified, real-world learning shortcut we found in arguable the most established audio deepfake detection dataset, show its implications and make suggestions on how to identify such artifacts in the future. We argue that learning shortcuts are not negligible technicalities, but rather fundamental observations, because they lead to systematic over-estimation of model capability. As one reviewer summarizes, 'this work [...] can make us question part of the work presented so far.' Please note that this chapter presents work from and quotes verbatim on our publication where we disclosed our findings to the community [1].

3.1 Introduction to the ASVspoof challenge

Artificial intelligence methods are not only leading to improvements in many areas such as medical technology, object recognition, and computer-human interfaces but are also creating new and very unique problems. Perhaps one of the most outstanding challenges in this regard is deepfakes, i.e. computer-generated video or audio recordings of people. These fake media can be used to put arbitrary words into the mouths of targets [45]. This enables hate campaigns, denial, and fraud [46].

One of the possible countermeasures is the computer-assisted detection of deepfakes. This also operates on a pattern detection basis, i.e. by using AIs that distinguish deepfakes from real media content. This branch of research is also called 'anti-spoofing'. The most important conference in the field of audio anti-spoofing is the ASVspoof conference [47, 48], which has also published several datasets that it uses to organize biennial competitions.

The ASVspoof 2019 Challenge Dataset [47] is the most established dataset for training and benchmarking systems designed for the detection of spoofed audio and audio deepfakes. Overall, the dataset presents two related tasks: logical access (LA) and physical access (PA). All of the data is based on the VCTK corpus [49]. In this chapter, we focus exclusively on the LA task, which is concerned with the detection of spoofed speech (i.e.

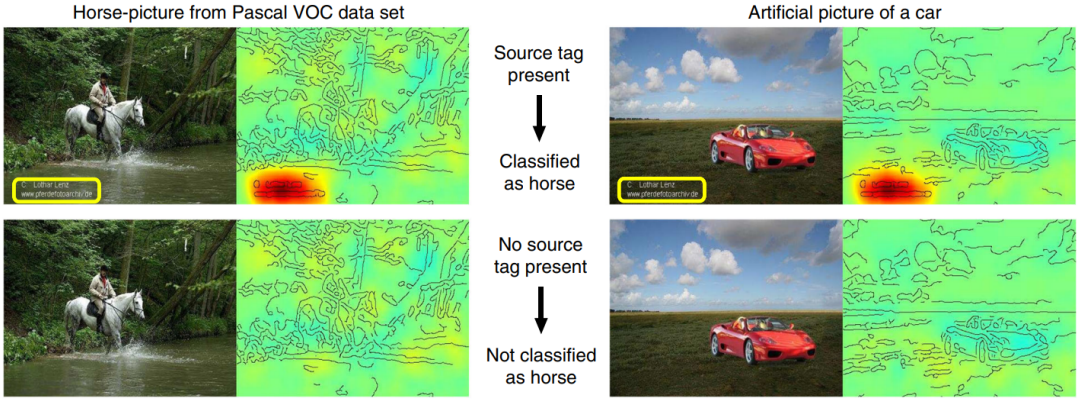


Figure 3.1: Figure from the Pascal VOC 2007 dataset, illustrating the concept of 'shortcut learning'. Here, the photographer's watermark (highlighted by a yellow rectangle) is a strong indicator for the classification of an image as 'horse'. Images with the watermark are classified as a horse (top row), and images without are not classified as a horse (bottom row), irrespective of the actual content of the image. Figure taken from [44].

'deepfakes'). This is because the PA task does not contain the artifact. This is because the PA task concerns itself not with deepfakes, but with lifelines detection (i.e. the detection of whether an audio is spoken live by a human or a playback of a recording).

3.1.1 The ASVspoof 2019 Logical Access (LA) Dataset

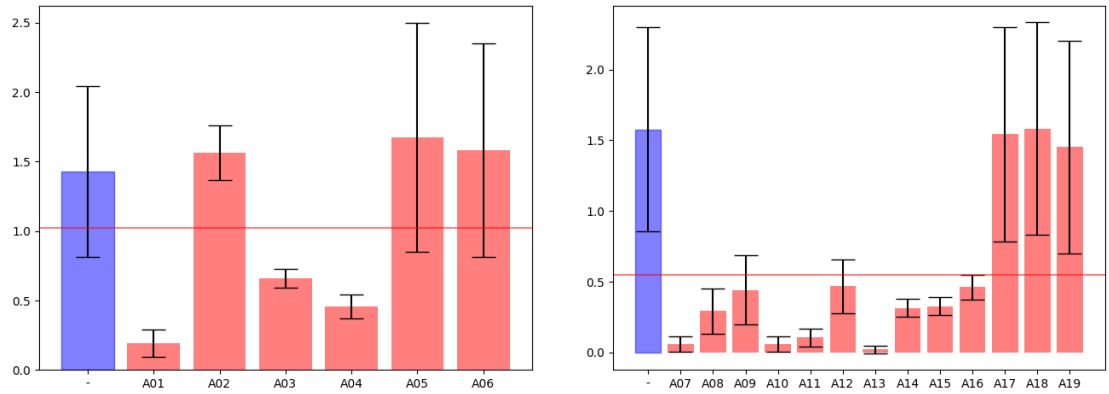
The dataset consists of pairs "speech, label" where speech is a recording of human speech, either synthesized (label=spoof) or authentic (label=bonafide). Samples can be listened to at https://deepfake-demo.aisec.fraunhofer.de/samples. Already, several competitions have been organized, which all base their challenges on this dataset [47, 48], and a significant number of related work has been published which uses ASVspoof as their baseline [50, 51, 52, 53]. More details about the dataset are detailed in [54].

3.2 The artifact: silence as a shortcut to learning

We notice the following irregularity in the ASVspoof 2019/2021 data:

The distribution of the silence in the audio datasets is skewed. Specifically, the *length* of an audio's silence strongly hints at the target label (spoof or benign). Figure 3.2 visualizes this for each of the three data splits 'train', 'dev', 'eval'. The bonafide samples have much longer silences than many of the attacks. This is highly problematic since a model learns to base its decision on the duration of the silence. Any data-driven learner, such as a neural network, will pick up on this uneven distribution of silences in the ASVspoof data. Such dataset artifacts are not unknown in machine learning. Consider the issue found in the Pascal VOC 2007 dataset, where all images of horses also contained a specific watermark [44]. A model could exploit this artifact by learning a 'shortcut',

Figure 3.2: The duration of the leading + trailing silence per attack ID in the LA part of ASVspoof 2019, shown individually for the bonafide data (blue) and the attacks (red). The error bars highlight the standard deviation and the red horizontal line displays the average silence duration over all malicious attacks. Note the average length of silence is much shorter for attacks (red) than for bonafide audios (blue).



(a) The average silence duration in seconds per audio file for the 'train' split.

(b) This plot shows the average silence duration in seconds per audio file for the 'eval' split.

i.e. that watermark equals horse, without actually learning the true features of a horse. Removing the watermark completely broke the model, c.f. figure 3.1 We hypothesize that deepfake-detection AIs may behave similarly in the context of ASVspoof. This raises the question: has related work learned to discriminate audio deepfakes solely, or at least partially, based on the duration of the leading and trailing silences?

3.3 Experimental Analysis

This section presents extensive experiments to answer this question. First, however, we give some technical background on the evaluation of anti-spoof models.

3.3.1 Preliminaries

Anti-spoof models usually are regression models, i.e. they predict a score between 0 and 1, or between $-\infty$ and $+\infty$. This is because spoof detection works in two stages: first, they judge the 'anomalousness' of an audio. Using this score, a threshold needs to be found above which the audio is considered authentic, and below which the audio is considered fake or spoofed. Splitting the problem into these two sequential steps allows for a more fine-grained evaluation than immediate binary classification, and is true to the observation that fakes can have various qualities. It additionally allows to fine-tune the threshold to specific requirements and take into account the trade-off between false-positive and false-negative cost.

Thus, when evaluating anti-spoof methods, we are usually most interested in the first problem, i.e. the score computation. Given good scores, finding an appropriate threshold is feasible. Thus, the metric used is not Accuracy, but Equal Error Rate (EER). The EER is a single-number metric that measures the error rate of an anti-spoofing system, assuming that false-positive and false-negative rates are balanced. The theoretically maximal value of the EER is 1. However, this usually is of small practical importance, since in practice, the lower limit is usually 0.5, which translates to random guessing. A perfect model would achieve an EER of 0. Note that percent and decimal notation are equivalent, i.e. an EER of 0.5 equals 50% EER. In summary, the EER is the error rate at which false-positive and false-negative errors are balanced.

3.3.2 Model and Metrics Descriptions

To evaluate the impact of the observed data artifact further, we first need to set up various deepfake-detection AIs. We then study their behavior, which allows us to determine whether the artifact will result in a learning shortcut. We implement the following models:

3.3.2.1 Random Baseline

First, we implement a random 'dummy' baseline, which has no trainable parameters and scores new samples with a random scalar $\lambda \in [0, ..., 1]$. This is equivalent to random guessing. We expect this model to yield an Equal Error Rate (EER) of 50% on any given dataset or task.

3.3.2.2 Weak Baseline

Additionally, we implement a fully connected neural network (FCNN) with two hidden layers of size 128, ReLU activation, 10% Dropout, and a single output neuron with sigmoid activation. This simple model takes as input an audio file (preprocessed as spectogram) and outputs a scalar value. This model can learn sensible mappings but has very low capacity. It serves as a baseline for more advanced learners.

3.3.2.3 Strong Models from Related Work

We implement four models, a Deep Residual Network, an LSTM, a CNN-GRU model, and RawNet2 as described below. These models are more powerful than the previously employed linear model and achieve an impressive EER of up to 5.87% when trained on the 'train' split and evaluated on the 'eval' split of ASVspoof 2019.

ResNet. This model is a deep residual pre-activation network similar to [51]. It consists of a stack of four residual layers, each of which consists of two stacks of batch-norm, leaky-relu, and a two-dimensional convolutional network. We use a kernel size of $(3, 3)$, stride of $(1, 1)$ and padding of $(1, 1)$. Since these configurations do not shrink the spatial dimensions, each ResNet block employs AveragePooling along the feature dimension, preserving the time dimension (i.e. kernel size of $(3, 1)$). After the last ResNet block, we reshape the output to shape (B, L, F) where B is the batch size, L is the length of the

time dimension, and F is the flattened feature vector. Finally, two dense layers compress this into $(B, L, 1)$, which is aggregated via *mean* along the time axis and yields a single logit $\lambda \in \mathbb{R}$ per element in the mini-batch.

CNN. This model closely follows the architecture presented in [55], and consists of a stack of convolutional layers with (3,3) kernels, each with batch-norm and ReLU activation, followed by a Gated Recurrent Network. The CNN output is reshaped to conserve the time dimension (i.e. aggregating channel and feature dimension) and fed into a three-layer GRU, whose outputs are fed into two linear layers with 50% dropout. The final linear layer has a single output neuron, followed by sigmoid activation. These linear layers classify each feature frame separately, and the result is aggregated via mean over the time dimension and finally activated via the sigmoid function.

LSTM. This model follows [56], and consists of a three-layered, bi-directional LSTM with a 10% percent dropout and 256 hidden neurons. The output of the LSTM is fed into a linear projection layer, which has a single output neuron. We take the mean overall outputs along the time dimension, which yields a single logit $\lambda \in \mathbb{R}$. Finally, we apply sigmoid activation to match the targets $\in [0, 1]$.

RawNet2. This model [57] is a state-of-the-art convolutional/residual neural network using raw features as input. This means that the audio is not transformed into spectogram, but that the waveform (16000 samples per second) is supplied to the model directly, which then employs stacks of convolutional layers to extract higher-level features. This model is an official baseline for the ASVspoof 2021 challenge. We use the source code provided by the authors.

3.3.3 Experimental Description

In this section, we describe the experiments we perform to estimate the impact of the observed data artifact.

3.3.3.1 Experimental Setup

We shortly give a general description of our experiment setup. We train our models on the 'train' split and evaluate their performance both on the 'dev' and 'eval' split. We implement the models in Python 3.8 using PyTorch and run the experiments on an Nvidia A100 GPU. We train the models for 50 epochs each with a learning rate of 0.001, using the Adam Optimizer [58] with a weight-decay of $1e^{-6}$. No early stopping is employed. We standard-normalize our features (subtracting the mean and dividing by the standard deviation) and use binary cross-entropy loss.

3.3.3.2 Experiment: Predictive Power of Length of Silence

To evaluate the impact of the leading silence, we train the FCNN described in Section 3.3.2.1 on the ASVspoof 2019 dataset, but extract only a **single** feature: The duration of the leading silence in seconds. If there is information contained in just the *duration* of the silence, we expect the FCNN to outperform the random baseline, i.e. obtain EER less than 50%. Section 3.3.4.1 presents the results.

3.3.3.3 Experiment: Predictive Power of Silence in Related Work

Additionally, we use three stronger models, a Deep Residual Network (ResNet), an LSTM and a CNN-GRU model (c.f. Section 3.3.2.1) in combination with CQT features [59], and train them with and without access to the audio silence. More specifically, we employ two different pre-processing techniques:

- **Time-wise subselection:** The first pre-processing technique is dubbed 'time-wise subselection'. If enabled ($t > 0$), it returns a random subsection of the audio of duration t. For each epoch, a new random selection is returned. If $t = -1$, this pre-processing technique is disabled and the audio remains unchanged. We test two settings: $t = 2.4s$ and $t = -1$. For $t = 2.4s$, a randomly chosen slice of duration 150 (since $150 * 0.016s = 2.4s$) from the audio is returned (where the hop size is 0.016 seconds). This has, among others, the effect that the model does not have consistent access to the audio silence.

- **Trim Silence**. The second pre-processing technique trims, if enabled, the silence of all audios before returning them to the model. We use the librosa [60] library for this and employ $librosa.effects.trim$ with a ref_db of 40 as a threshold to get the duration of the leading silence.

We train these models with four possible combinations of the two data augmentation techniques (time-wise subselection, and silence trimming). These combinations yielded twelve configurations. We run each of these two times, resulting in training 24 models in total. We expect to see a deterioration in EER if indeed the duration of the silence leaks significant information w.r.t. the target label. Results are presented in Section 3.3.4.2.

In another version of this experiment, we repeat the previous trial but remove the silence only from the evaluation data. More specifically, we train a model on the unmodified 'train' split of ASVspoof 2019 with full access to the original audio, i.e. *with* access to the silences. We then evaluate the model on two versions of the 'eval' split: First, we evaluate it on the unchanged evaluation data. Second, we test it on the evaluation data wherein the leading and trailing silence had been truncated. Results are presented in Section 3.3.4.2.

3.3.3.4 Experiment: Predictive Power of Silence in ASVSpoof 2021

Additionally, we integrated RawNet2 [57], one of the official ASVspoof2021 baselines, into our test setup. Since it uses *feature subselection* by design ($t = 4s$), we skip the configuration $t = -1$, but evaluate only the impact of silence trimming. We train the model for 50 epochs and three times for each configuration and present the results in Section 3.3.4.3.

Table 3.1: The results of training a dense neural network on only the *duration* of leading silence, compared against a random baseline. The network can significantly outperform the random baseline (see test EER, highlighted in blue). Results for the 2021 data (Deepfake and Logical Access) are copied verbatim from the submission platform (*progress* phase).

Model	Dev EER	Eval EER	LA21	DF21
FCNN	31.09±3.8	15.12±0.1	19.93	17.42
Random	50.0±0.0	50.0±0.0	-	-

3.3.4 Results and Analysis

3.3.4.1 Results: Predictive Power of Length of Silence

Table 3.1 shows the results when training the FCNN only on the *duration* of the leading silence. The data is averaged over four individual runs, and shown with standard deviation. Our very simple model substantially outperforms the random baseline, and achieves a respectable 15.12% EER on the test set (vs. 50% EER of the random baseline), using only a single feature as input: the duration of the leading silence (i.e. only a single, scalar input).

To put this in perspective, related work [61, 62, 51] achieves about 4-8% test EER, which is not that much better than our naive baseline.

3.3.4.2 Results: Predictive Power of Silence in Related Work

This section presents the results of the experiment described in 3.3.3.3, where we train models from Section 3.3.2.1 both with and without access to the audio silence. Table 3.2 presents the individual results.

Note that there is a single configuration of pre-processing steps where the model has the chance to consistently exploit the information contained in the duration of the leading silence: Subselection *t = -1* and *Trim Silence = False*. For all models, this configuration considerably outperforms all other configurations: The models net an EER of 5.87%, 7.48%, and 8.33% (blue boxes) when they have reliable access to the leading silence. In the other three scenarios, they never achieve more than 19.05% EER. This is less than what the very basic single-feature dense network in Section 3.3.4.1 achieved. This strongly hints at the fact that the models draw considerable, if not most of their discriminatory power from either the duration of the silence or the silence itself.

3.3.4.3 Results: Predictive Power of Silence in ASVSpoof 2021

Table 3.3 presents the results for RawNet2, one of the ASVspoof 2021 baselines.

We observe the same trend when as in the previous chapters, c.f. Table 3.3: For both the DF and the LA task, trimming the silence during training severely degrades model performance (EER increases from 6.28% to 20.79% for the deepfake and 10.38% to 27.39%

Table 3.2: Three Deep Learning Models are trained on the ASVspoof 2019 'train' split and evaluated on both 'dev' and 'eval', each with four different configurations of data augmentation. For each model, the test EER is lowest (blue box) when the data augmentation allows consistent access to the audio silence.

Model	Trim Sil.	Subsel.	Dev EER	Eval EER
ResNet	False	-1	7.08±1.5	5.87±1.8
ResNet	False	2.4s	1.66±0.4	24.68±2.2
ResNet	True	-1	8.83±7.6	27.23±3.2
ResNet	True	2.4s	1.83±0.1	29.13±1.6
CNN	False	-1	0.15±0.1	7.48±0.6
CNN	False	2.4s	0.61±0.2	20.02±0.9
CNN	True	-1	0.68±0.4	26.27±3.5
CNN	True	2.4s	0.34±0.1	25.47±0.9
LSTM	False	-1	0.84±0.5	8.33±1.2
LSTM	False	2.4s	4.50±0.2	19.05±0.6
LSTM	True	-1	6.21±0.7	27.28±1.4
LSTM	True	2.4s	6.44±1.2	25.62±0.4

Table 3.3: RawNet2 [57], one of the baselines of the ASVspoof 2021 challenge, trained with and without access to the audio silence. Results averaged over three runs. Removing silence from the Training set prevents the model from using the duration of the silence in the eval set as a cue. Consequently, 2019 'eval' EER deteriorates by about 500%. The same phenomenon can be observed for the 2021 data from the Deepfake (DF) and Logical Access (LA) track (EER values copied as-is from the CodaLab submission website during the *progress* phase on July 7th, 2021).

Model	Trim Sil.	EER Eval-19	DF-21	LA-21
RawNet2	False	3.61±1.2	6.28	10.38
RawNet2	True	15.50±5.2	20.79	27.39

for the LA part). The same is true for the ASVspoof 2019 data. Again, we observe that suspiciously much information is contained in the audio silence, without which model performance degrades significantly.

3.4 Mitigation Strategies specific to ASVspoof

In the previous section, we showed that the learning shortcut in the ASVspoof dataset leads to systematic over-estimation of model capabilities. Naturally, the question arises on how to fix this issue. Let us first consider the validity of including silence in spoofing-detection datasets such as ASVspoof. Silence usually is not just 'nothing' or a string of zeros, but has humanly imperceptible properties and characteristics, for example, faint background noises, artifacts from the microphone used, etc. Thus, silence in a digital

recording is represented by a series of non-zero scalar values of low magnitude. It is reasonable to assume that a TTS system will not be able to fake silence perfectly, so silence can be legitimately used to classify audio recordings into spoof/bonafide. Thus, in general, it is permissible and even desirable for recordings in a dataset such as ASVspoof to include leading and trailing silence.

What is problematic, however, is an uneven distribution of silence. As we show in Figure 3.2, the mere *duration* of silence strongly hints at the ground-truth label. In this case, the model learns classification not due to the properties and characteristics of the silence itself, but due to its *duration* (c.f. Table 3.1).

Thus, we suggest including silence in the dataset but have it distributed evenly between the bonafide and spoofed instances. However, it is questionable if the existing ASVspoof datasets can be augmented with sufficient samples such that the distribution of silence is balanced. Thus, this approach should be considered primarily for upcoming datasets. For ASVspoof 2019 and 2021, we suggest publishing a revised version of the dataset, where the leading silence has been removed. This is easy to implement via the Python librosa library or a forced aligner such as the Montreal Forced Aligner (MFA) [63].

3.5 A General Approach to Detecting Learning Shortcuts in ML-Datasets

Learning shortcuts are not just a problem in the ASVSpoof dataset. As described above, they have been found in other datasets, such as Pascal VOC 2007 [44]. Still, is an underappreciated problem that is poorly researched and often ignored. In this section, we lay out why these shortcuts are so difficult to find, and why their impact cannot be measured using traditional metrics. We suggest directions for further research to develop systems and techniques for detecting these shortcuts.

3.5.1 Challenges in Identifying Learning Shortcuts

Usually, the vast majority of learning issues in artificial intelligence are identified via the train/test methodology. This means that the data is split into (at least) two disjoint parts, and the learner is trained on one part and evaluated on the other. Common problems such as overfitting, underfitting, and lack of generalization can be easily detected by comparing success metrics (accuracy, EER, etc.) between the train and evaluation split. At first glance, this approach seems sound: If the model performs well on unseen data, it is assumed to generalize to unseen data. However, train/test generalisation is only a *necessary*, not *sufficient* condition for model generalisation. Necessary because if the model cannot handle unseen test data, it does not generalize. Sufficient because generalization on test data has only limited meaningfulness w.r.t. true model generalization. This is because, as we saw in the example of ASVspoof, artifacts that are present in both splits of the data lead to learning shortcuts. While these shortcuts improve train/test generalization, they work against true model generalization. However, since the established methodology for finding learning errors is the train/test evaluation, these shortcuts are seldom found.

In fact, since they do help train/test performance, researchers tend to embrace them. In summary, learning shortcuts are challenging to detect because the research community lacks the tools to detect them, and the tools available actually favor the presence of learning shortcuts.

3.5.2 How to Find Learning Shortcuts

Thus, in order to identify data artifacts leading to learning shortcuts, a new approach is required. We suggest that the current paradigm of train/test evaluation, which is susceptible to learning shortcuts, should be extended by a qualitative evaluation of the dataset itself. We suggest exploring one of the following strategies:

3.5.2.1 Explainable-AI techniques

. One possibility would be to evaluate the model, not via some numerical metric, but to employ a human expert to asses if the model performs appropriately. This is already common practice in the field of generative models, for example, speech synthesis. Here, the quality of synthesis is determined via a Mean Opinion Score (MOS), where humans rate the audio quality on a scale between 0 and 5. This is done because there are no success metrics that adequately capture the quality of the prediction $\hat{y}$, even in the presence of the ground-truth y. However, such an assessment is difficult for classification- or regression models, which produce only a class score or scalar value as output. Techniques from the domain of explainable AI could help, for example, saliency maps or attention maps, which show how a model derives its output. For example, refer to figure 3.1 on page 38: The heatmap on the right side of the four images clearly shows where the model's attention is focused, and one can see that classification is based only on a single feature, namely the presence of the watermark in the lower-left hand corner. A human assessor would need to check a selection of images' heatmaps, and compare them against the human understanding of the learning goal. While this process is expensive and cannot be automated, it is preferable having one's model broken due to learning shortcuts.

3.5.2.2 Data-only approaches

While techniques from explainable AI as described above certainly work, they come with several downsides. They are hard to automatize, which might lead to little practical usage. Additionally, they do not directly capture the data artifact, but rather the learning shortcut which derives from it. A more immediate, data-only, model-agnostic technique is thus desired. We present an approach in the following paragraph.

We can define shortcuts as features F_i which are

- not independent w.r.t. target T (i.e., the feature is predictive, $p(T|F_i) \neq p(T)$),
- and which are semantically unrelated to target T (i.e., the feature should not be predictive).

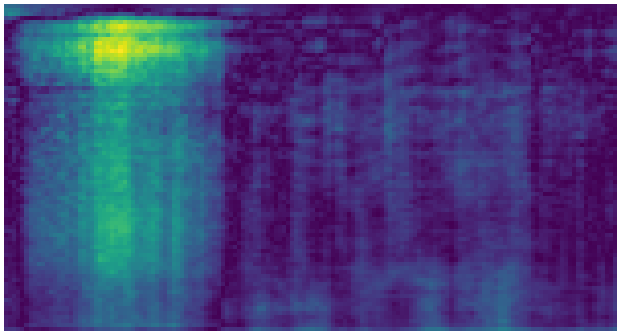


Figure 3.3: This figure shows an aggregation of the ASVspoof 2019 dataset's information as described in equation (3.1). The x axis represents time, while the y axis represents frequency bins. Bright spots represent areas where the pixel x_{ij} holds much information w.r.t. the target class y, while darker regions represent areas where there is less information (per pixel). We see that the left part of the image contains high pixel-wise information, corresponding to the prevalence of silence, depending on the label. Refer to figure 3.4 for visualisation of the individual data instances.

	$p(T\|F_i) \neq p(T)$	$p(T\|F_i) = p(T)$
F_i meaningful	ideal	not ideal
F_i irrelevant	possible learning shortcut	ok

Table 3.4: Interplay between feature predictiveness and semantics. Meaningful features that are predictive are useful for data-driven learners. Irrelevant features that are not predictive are also acceptable. Meaningful features that are not predictive do not pose a problem but are squandered possibilities. Finally, irrelevant features that are predictive are a big problem, since they enable learning shortcuts.

In contrast, how (in)dependence and feature semantics should be related is shown in table 3.4. Only when an irrelevant feature is predictive for T, does the possibility for learning shortcuts arise.

Whether or not a feature *should* be predictive requires a human judge. However, estimating whether a feature is predictive can be solved using the following approach:

$$
\begin{aligned}
& p(T|F_i) \stackrel{?}{=} p(T) \\
\Leftrightarrow\ & p(T, F_i) \stackrel{?}{=} p(T)P(F_i) \\
\Leftrightarrow\ & p(F_i|T) \stackrel{?}{=} p(F_i) \\
\Leftrightarrow\ & \log p(F_i|T) \stackrel{?}{=} \log p(F_i) \\
\Leftrightarrow\ & 0 \stackrel{?}{=} \log p(F_i) - \log p(F_i|T) \\
\Leftrightarrow\ & 0 \stackrel{?}{=} \log \frac{p(F_i)}{p(F_i|T)}
\end{aligned}
$$

Since we have only sample to work with, we evaluate this over our dataset D:

$$\mathbb{E}_{F_i \sim D} \log \frac{p(F_i)}{p(F_i|T)} = \sum_{F_i} p(F_i) \log \frac{p(F_i)}{p(F_i|T)} = KL\left(p(F_i)||p(F_i|T)\right) \tag{3.1}$$

Here, KL is the KL-Divergence. The probabilities $p(F_i)$ and $p(F_i|T)$ can be estimated using, for example, a histogram. Note that $p(F_i|T)$ requires to select a target $T = t$, i.e. we compute $p(F_i|T = t)$. Equation (3.1) can then be aggregated via either mean or max over all target classes t. Thus, in summary, we derive a way to measure an individual feature's predictiveness, given dataset D.

We have implemented this approach as an early prototype, and present the first results in figure 3.3. This figure shows a representation of the ASVspoof 2019 dataset, where the information in all audio files has been aggregated and contrasted between the two classes $y = 0$ and $y = 1$ (authentic and spoof). We can see that the first half of the image is bright, whereas the second part is dark. This corresponds to the fact that soundwaves during the first few timesteps (with time denoted on the x axis) hint at the target label: if the audio starts without a pause, it has a high probability of being spoofed. Figure 3.4 shows random samples for the ASVspoof 2019 dataset, where the top row shows benign and the bottom row malicious samples. Note how the benign samples have long leading silence, corresponding to empty frequency bins in the left half of the images.

While this artifact could be identified by closely looking at the samples, there are other data artifacts imaginable that cannot be spotted by inspecting individual samples. For example, we introduce an artificial artifact into the dataset: we chose three random frequencies, which we attenuate to 95%, but only for instances where the target $y = 0$. This means that, on average, certain frequencies are a bit less pronounced than others for the spoofed instances. The top row of figure 3.4 shows an MFCC-representation of such audio files. The artifact is very difficult to detect visually, especially for the unsuspecting eye. However, applying equation (3.1) yields an information heatmap as shown in the left image in figure 3.5, where the frequency-masking is clearly visible. In this aggregated view, the artifact is clearly visible. In summary, such a tool may be an easy, quick and straightforward way to check the data for artifacts and possible learning shortcuts.

3.5.3 Limitations of Proposed Approach

A key limitation is that we cannot compute this measure for more than one or, at the most, two features F_i simultaneously. Put differently, we cannot compute

$$KL\left(p(F_i, \ldots, F_j)||p(F_i, \ldots, F_j|T)\right)$$

The problem lies in estimating probabilities $p(F_i, \ldots, F_j)$ and $p(F_i, \ldots, F_j|T)$ given the data: If we used a histogram with k bins (as above), estimating the joint (conditional) probabilities would require k^N bins, where N is the number of random variables $F_i, , \ldots, F_j$. The number of bins rises exponentially, while the amount of data remains constant. Thus, for larger N, say $N > 2$, the histogram's bins would be populated only very sparsely, yielding pathological probability estimates. This prevents us from analyzing shortcuts

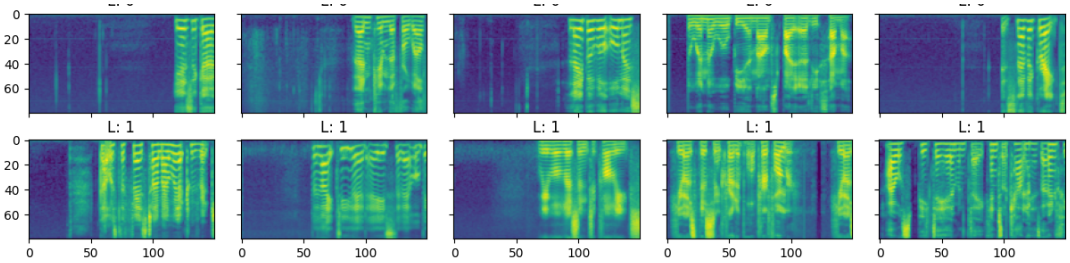


Figure 3.4: Random samples from the ASVspoof 2019 dataset, with the label $y = 0$ in the first row, and $y = 1$ in the second. The images show the first 150 time-slots on the x axis, and the frequency bins on the y axis. The longer silences for $y = 0$ are not visible for every instance.

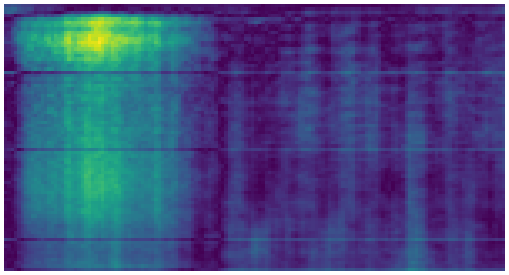


Figure 3.5: To demonstrate how data artifacts may be discovered, we introduced an artificial artifact into the ASVspoof dataset shown in figure 3.4: Three randomly chosen frequencies have been masked by 5% for all instances where the class is $y = 0$. One can observe faint vertical lines for the top-row images of figure 3.4. However, it may be hard to spot this just from the data, when not knowing what to look for. Data aggregation as per equation (3.1) could alleviate this problem: This figure presents such an aggregation, where the artifact is much more clearly visible (distinct vertical lines). Note that we see the original artifact (missing silence) as the bright streaks on the left of the images.

that only occur as a combination of several F_i. An example would be a shortcut such as $F_i = F_j \implies T = t$ and $F_i \neq F_j \implies T = \neg t$.

In summary, there are some promising approaches to identifying data artifacts. We strongly argue for the necessity of such tools, because if such sanity checks are left out, entire research communities may over-estimate the performance of their models, as happened in the case of ASVspoof [1].

4 Integrity of Active Learning

The previous chapter was concerned with detecting errors in the data sampled from the underlying, data-distribution $p(X)$. Now, let us consider deliberate data contamination, i.e. adversarial attacks on the sampling process. In this chapter, we present such an attack that lets the attacker control which instances are included in the final dataset, with a success rate of up to 100%. We analyze implications and design and evaluate appropriate countermeasures. Please note that this research has first been published as a first-authored paper in the 19th Symposium of Intelligent Data Analysis, and passages are quoted verbatim from the corresponding paper [2].

4.1 Active Transfer Learning

Training supervised machine learning algorithms such as neural networks requires large amounts of labelled training data. In order to solve problems for which there is no or little training data, previous work has developed techniques such as Transfer Learning and Active Learning.

- **Transfer Learning** (TL) applies knowledge gained from one problem to a second problem. For example, consider the problem of training a Neural Network on a very small image dataset (say $N = 500$). Training directly on the dataset will yield poor generalization due to the limited number of training examples. A better approach is to use a second, already-existing network trained on a different task to extract high-level semantic features from the training data, and train the first network on these features instead of the raw images themselves. For images, this is commonly employed practice and easily accessible via the *tensorflow* library. For audio data, similar feature extractors are also easily accessible [64].

- **Active Learning** (AL) on the other hand is a process where a learning algorithm can query a user. AL is helpful when creating or expanding labelled datasets. Instead of randomly selecting instances to present to a human annotator, the model can query for specific instances. The model ranks the unlabelled instances by certainty of prediction. Those instances for which it is most uncertain are the ones where a label is queried from the human annotator. Model uncertainty can be understood as the distance to the decision surface (SVM), or entropy of the class predictions (*uncertainty sampling* for Neural Networks). In summary, Active Learning can help in finding the optimal set of instances to label in order to optimally use a given budget [65].

Since these approaches are complementary, they can be combined straightforwardly: One designs a *transfer active learning* system by combining an untrained network with a pre-trained feature extractor and then allows this combined model to query a human expert. Previous work has examined this in detail and finds that it can accelerate the learning process significantly [66, 67, 68]. In this chapter, we show that this combination of AL with TL is highly susceptible to data poisoning.

4.2 Related Work

In this section, we discuss related work on transfer active learning. We first present work on combining transfer and active learning and then discuss related data poisoning attacks for each approach. We observe that there is no prior work on data poisoning for the combined method of transfer active learning.

4.2.1 Active Learning with Transfer Learning.

Kale et Al. [66] present a transfer active learning framework which "leverages pre-existing labelled data from related tasks to improve the performance of an active learner". They take large, well-known datasets such as *Twenty Newsgroup*, and evaluate the number of queries required in order to reach an error of 0.2 or less. They can reduce the number of queries by as much as 50% by combining transfer learning with active learning. The authors of [67] perform similar experiments, and find that "the number of labelled examples required for learning with transfer is often significantly smaller than that required for learning each target independently". They also evaluate combining active learning and transfer learning, and find that the "combined active transfer learning algorithm [...] achieve[s] better prediction performance than alternative methods".

Chan et al. [69] examine the problem of training a word sense disambiguation (WSD) system on one domain and then applying it to another domain, thus 'transferring' knowledge from one domain to another domain. They show that active learning approaches can be used to help in this transfer learning task. The authors of [70] examine how to borrow information from one domain in order to label data in another domain. Their goal is to label samples in the current domain (in-domain) using a model trained on samples from the other domain (out-of-domain). The model then predicts the labels of the in-domain data. Where the prediction confidence of the model is low, a human annotator is asked to provide the label, otherwise, the model's prediction is used to label the data. The authors report that their approach significantly improves test accuracy.

4.2.2 Poisoning Active Learning.

Poisoning active learners requires the attacker to craft samples which, in addition to adversely affecting the model, have to be selected by the active learner for labeling and insertion into the dataset. This attack aims at both increasing overall classification error during test time as well as increasing the cost for labeling. In this sense, poisoning active learning is harder than poisoning conventional learners, since two objectives have to be

satisfied simultaneously. Miller et Al. [42] present such an attack: They poison linear classifiers and manage to satisfy both previously mentioned objectives (albeit with some constraints): Poisoned instances are selected by the active learner with high probability, and the model trained on the poisoned instances induces significant deterioration in prediction accuracy.

4.2.3 Adversarial Collisions

There exists some related work on adversarial collisions, albeit with a different focus from ours: Li et Al. [71] observe that neural networks can be insensitive to adversarial noise of large magnitude. This results in 'two very different examples sharing the same feature activation' and results in a feature collision. However, this constitutes an attack at test time (evasion attack), whereas we present an attack at training time (poison attack).

4.3 Attacking Active Transfer Learning

In this section, we introduce the proposed attack. We first present the threat model and then detail the attack itself. In Section 4.4, we evaluate the effectiveness of our attack empirically.

4.3.1 Threat model

Our thread model is largely identical to the one presented in section 6.4.1. However, in the transfer active learning setup, there are additional components that we have not taken into consideration before: This includes the active learning algorithm itself, and the feature extractor. Additionally, the input data is now unlabeled, in contrast to section 6.4.1, where the data was labeled. We shortly outline how this impacts our threat model.

- The attacker may introduce a small number of adversarially perturbed instances into the active learning pool. These instances are unlabeled. They are screened by the active learning algorithm and may, along with the benign instances, be selected by the Active Learning algorithm for labelling by the human annotator.
- The attacker cannot compromise the output of the human annotator, i.e. they cannot falsify the labels assigned to either the benign or poison instances.
- Additionally, the attacker cannot influence the pool of benign instances (i.e. they cannot remove or alter benign instances).
- The attacker knows the feature extractor used. This could, for example, be a popular open-source model such as *resnet50* [72] or *YAMNet* [73]. These models are readily available and widely used [74].
- The attacker has no knowledge about, or access to the model trained on top of the feature extractor.
- The attacker does not know the active learning algorithm.

This is a realistic scenario and commonly found in related work. For example, Shafahi et Al. [75] present an attack where poisoned data is published on the internet, waiting for unsuspecting crawlers. The authors proceed to show the applicability of their attack in the domain of image classification, where transfer learning is commonly employed.

4.3.2 Feature Collision Attack

Since transfer active learning is designed to use found data, i.e. data from untrusted sources, it is highly susceptible to data poisoning attacks. In this section, we present such an attack and show how it completely breaks the learned model.

Let X, Y be a set of unpoisoned data, where the data X and targets Y consist of N instances. An instance pertains to one of M different classes (e.g. 'dog' or 'cat'). Let f be the pretrained feature extractor, which maps a sample $x_i \in X$ to a d_ζ dimensional feature vector (i.e. $f(x_i) = \zeta_i$). Let g be the dense model head, which maps a feature vector ζ_i to some prediction $y_{pred} \in [0,1]^M$, where y_{pred} is a one-hot vector, i.e. $\sum_{m=0}^{M-1} y_{pred}^m = 1$. Thus, an image is classified by the subsequent application of the feature extractor f and the dense model head g:

$$y = \arg\max_i g(f(x_i)) \tag{4.1}$$

For a set of instances $\{x_i\}$, a set of adversarial examples $\{x_i + \delta_i\}$ can be found by minimizing, for each x_i separately:

$$\delta_i = \arg\min_\delta \left\| f(x_i + \delta) - \mu \right\|_2 + \beta \|\delta\|_2 \tag{4.2}$$

where $\beta \in \mathbb{R}^+$ and μ is a fixed vector of size d_ζ. Solving Equation 4.2 will find adversarial examples that 1) are selected by the active learner 2) break the model, and 3) are imperceptible to the human annotator:

1. **Examples are queried.** A set of thusly found adversarial examples $\{x_i + \delta_i\}$ will all be mapped to the same output μ, i.e. $f(x_i + \delta_i) = \mu$ for all i. This will 'confuse' the active learner, since all adversarial examples share the same feature vector, but have different class labels. Thus, the active learner will incur high uncertainty for instances mapped to this *collision vector* μ, and thus will query almost exclusively these (we verify this experimentally in Section 4.4.2).

2. **Examples are harmful**. These examples will break any model head trained on the extracted features ζ_i, since all adversarial examples share identical features, but different labels. Figure 4.1 gives a visual representation.

3. **Examples are undetected by a human annotator.** Once queried, the adversarial examples $x_i + \delta_i$ will be reviewed by a human annotator and labelled accordingly. The second part of Equation 4.2 ensures that the adversarial noise is small enough in magnitude to remain undetected by human experts. The adversarial example will thus be assigned the label of x_i, but not raise suspicion of the human annotator. The scalar β is a hyperparameter controlling the strength of this regularisation.

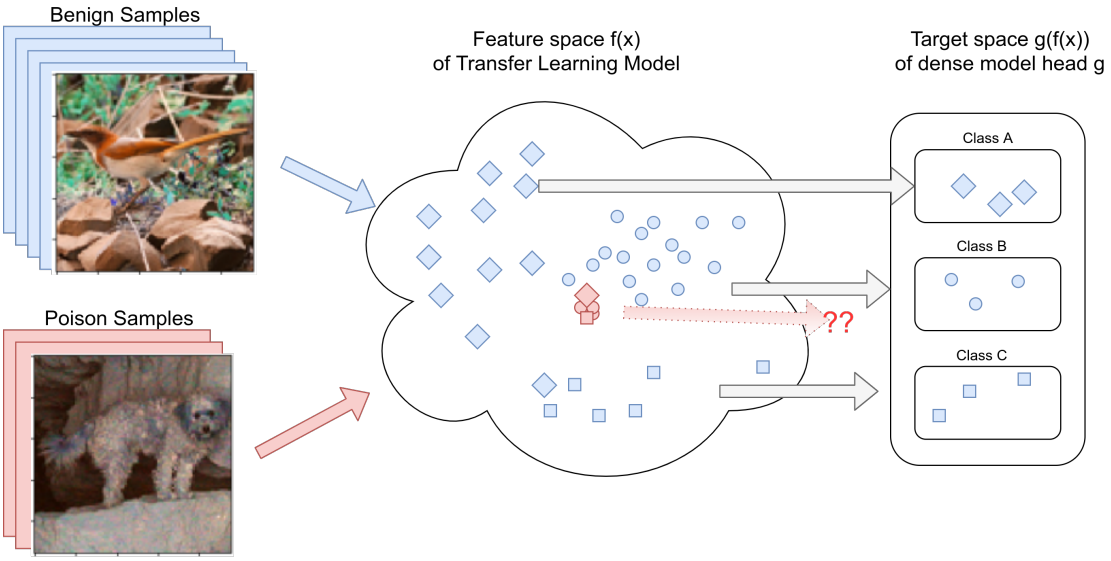


Figure 4.1: This diagram shows the effects of the proposed feature collision attack on Active Learning. On the left hand side, the raw images are shown: A larger stack of benign images (blue), and a smaller stack of images perturbed with our feature collision attack (red). The middle part of the figure represents the feature space of the transfer learner, $f(x)$. Observe how different classes (as indicated by the different shapes, i.e. circles, squares and diamonds) are mapped to different regions in feature space. However, due to the collision attack, malicious samples are all mapped to the same region in feature space, albeit they originate from different classes. The right part of the figure shows the target space of the dense model head, $g(f(x))$, i.e. the different class bins. The dense head can easily match the benign samples by finding a sensible mapping between regions in feature space and the class bins. For example, Class A is represented by the diamond shape, which is found in the upper left region of the feature space. The poison samples, however, cannot be distinguished in feature space, because they all cluster around the collision vector. Thus, they confuse the dense model head g (shown by the red, dangling arrow and the question marks), incurring high uncertainty. Thus, they'll be selected preferentially by the AL algorithm, trumping the benign samples.

4.3.2.1 Choice of Collision Vector

The *Collision Vector* μ is chosen as the zero vector $\mu = 0^{d_\zeta}$ because of two reasons: First, we find that it helps numerical convergence of Equation 4.2 when training with Gradient Descent. Second, a zero-vector of features is highly uncommon with unpoisoned data. Thus, it induces high uncertainty with the active learner, which in turn helps to promote the adversarial poison samples for labeling and inclusion in the training dataset from the beginning. It is possible to chose a different μ, for example the one-vector $\mu = 1^{d_\zeta}$, or the mean of the feature values $\mu = \overline{\zeta_i}$. However, we find that the zero-vector works best, most likely due to the reasons detailed above.

4.3.2.2 Improving attack efficiency

We propose two improvements over the baseline attack. First, when choosing the base instances from the test set to poison and to include in the train set, it is advisable to select those where either the $L1$ vector norm

$$\|f(x) - \mu\|_1 = \sum_j \left| f(x)^j - \mu^j \right| \tag{4.3}$$

or alternatively the $L2$ vector norm

$$\|f(x) - \mu\|_2 = \Big(\sum_j \left(f(x)^j - \mu^j \right)^2 \Big)^{1/2} \tag{4.4}$$

is smallest. $j \in [0, ..., d_\zeta - 1]$ iterates over the vector components of $f(x)$ and μ, which have the dimensionality of the feature extractor's output size. Intuitively, this pre-screens the samples for those where the optimization step (Equation 4.2) requires the least work. In our experiments, we use the $L2$ vector norm as defined by Equation (4.4). Second, maintaining class balance within the poison samples improves effectiveness. This helps in maximally confusing the active learner, since a greater diversity of different labels for the same feature vector μ increases the learner's uncertainty with respect to future samples that map to μ. In all of the following analysis, we evaluate the attack with these improvements in place.

4.4 Implementation and Results

In this section, we first describe our transfer active learning setup and the data sets used for evaluation. We then implement our attack and demonstrate its effectiveness (c.f. Section 4.4.2).

4.4.1 Active Transfer Learner Setup

This section describes the data we use and our choice of transfer learner.

For our experiments, we use image and audio data. This is because Active Learning requires a human oracle to annotate instances, and humans are very good at annotating

both image and audio data, but rather inefficient at processing purely numerical data. This motivates the choice of the active learner, namely a neural network, which in recent times has been shown to provide state-of-the-art performance on image and audio data.

Thus, we create the transfer learner as follows: We use a large, pre-trained model to perform feature extraction on the audio and image data. For image data, we use a pre-trained *resnet50* model [72], which comes with the current release of the python *tensorflow* library. For audio data, we build a feature extractor from the *YAMNet* model, a deep convolutional audio classification network [73]. Both of these feature extractors map the raw input data to a vector of features. For example, *resnet50* maps images with $32 * 32 * 3 = 3072$ input dimensions to a vector of 2048 higher-level features. A dense neural network (*dense head*) is then used to classify these feature vectors. Our Active Learner uses Entropy Sampling [65] to compute the uncertainty

$$\text{uncertainty}(x_i) = H(g(f(x_i))) \tag{4.5}$$

$$= -\sum_{m \in M} g(f(x_i))^m \log g(f(x_i))^m \tag{4.6}$$

for all unlabelled x_i. The scalar value $g(f(x_i))^m$ indicates softmax-probability of the m-th class for of the network's output. The active learner computes the uncertainty for all unlabelled instances x_i and selects the one with the highest uncertainty to be labeled.

4.4.1.1 Prevention of Overfitting.

As detailed in Section 4.1, we use active transfer learning in order to learn from very small datasets. Accordingly, we use at most $N = 500$ instances per dataset in our experiments. In this scenario, overfitting can easily occur. Thus, we take the following countermeasures: First, we keep the number of trainable parameters low and use a dense head with at most two layers and a small number of hidden neurons. Second, we use high Dropout (up to 50%) and employ early stopping during training. Third, we refrain from training the weights of the Transfer Learner (this is commonly referred to as *fine-tuning*). This is motivated by the observation that the *resnet50* architecture has more than 25 Million trainable weights, which makes it prone to overfitting, especially on very small datasets.

4.4.1.2 Datasets

We use three datasets to demonstrate our attack.

- *Google AudioSet* [76], a large classification dataset comprising several hundred different sounds. We use only a subset of ten basic commands (*right*, *yes*, *left*, etc.).
- *Cifar10* [77], a popular image recognition dataset with 32x32 pixel images of cars, ships, airplanes, etc.
- *STL10* [78], a semi-labelled dataset, comprising several thousand labelled 96x96 pixel images in addition to tens of thousand unlabelled images, divided into ten classes. In this supervised scenario, we use only the labelled subset.

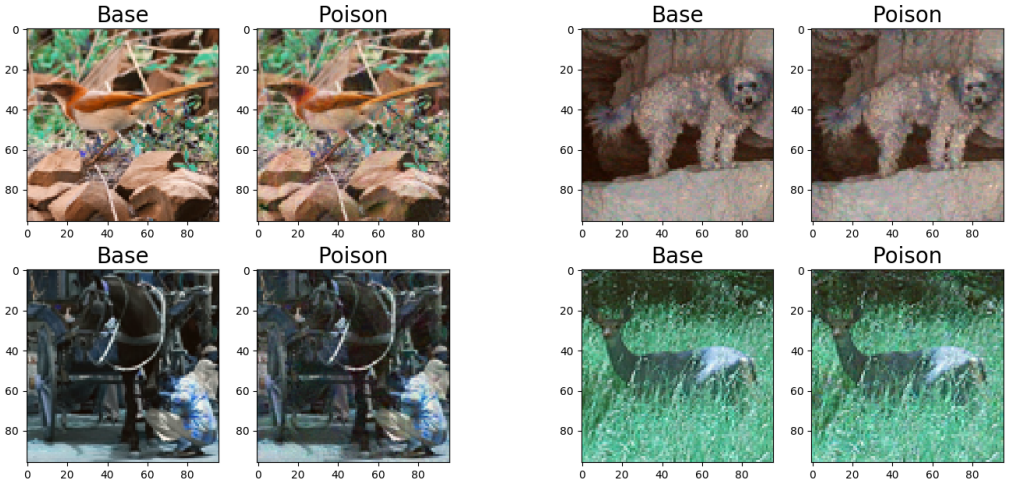


Figure 4.2: Images with index 155 (bird), 308 (horse), 614 (dog) and 3964 (deer) from the STL dataset. The left image of each pair shows the base instance, i.e. the unpoisoned image. The right image shows the poisoned image which causes the collision in the transfer learner's output space.

Each dataset is split into a *train* set and two test sets, *test1* and *test2* using an 80, 10 and 10 percent split. Following previous work [75], we train on the *train* set, use the *test1* set to craft the adversarial examples, and evaluate the test accuracy on the *test2* set. Thus, the adversarial examples base instances originate from the same distribution as the train images, but the two sets remain separate.

4.4.2 Feature Collision Results

We now verify empirically that the created instances look inconspicuous and are hard to distinguish from real, benign instances. Consider Figure 4.2. It shows four randomly selected poison instances of the $STL10$ dataset. Observe that the poisoned images are hardly distinguishable from the originals. We also provide the complete set of audio samples for the Google Audio Set[1], both poisoned and original .wav files.

We now proceed to visually illustrate the results of the feature collision. Consider Figure 4.3: For the poisoned and benign training data in the *AudioSet10* dataset, it shows the corresponding feature vectors after PCA-projection in two dimensions. Thus, it shows what the dense model head 'sees' when looking at the poisoned data set. Note that while the unpoisoned data has a large variety along the first two principal components (as is expected, since the individual instances pertain to different classes), the poison data's features collide in a single point (red dot) - even though they also pertain to different classes. Thus, we observe a 'feature collision' of the poison samples to a single point in feature space, which, in combination with the different labels, will cause maximum 'confusion' in the active learner.

[1] `https://drive.google.com/file/d/1JtXUu6degxnQ84Kggav8rgm9ktMhyAq0/view`

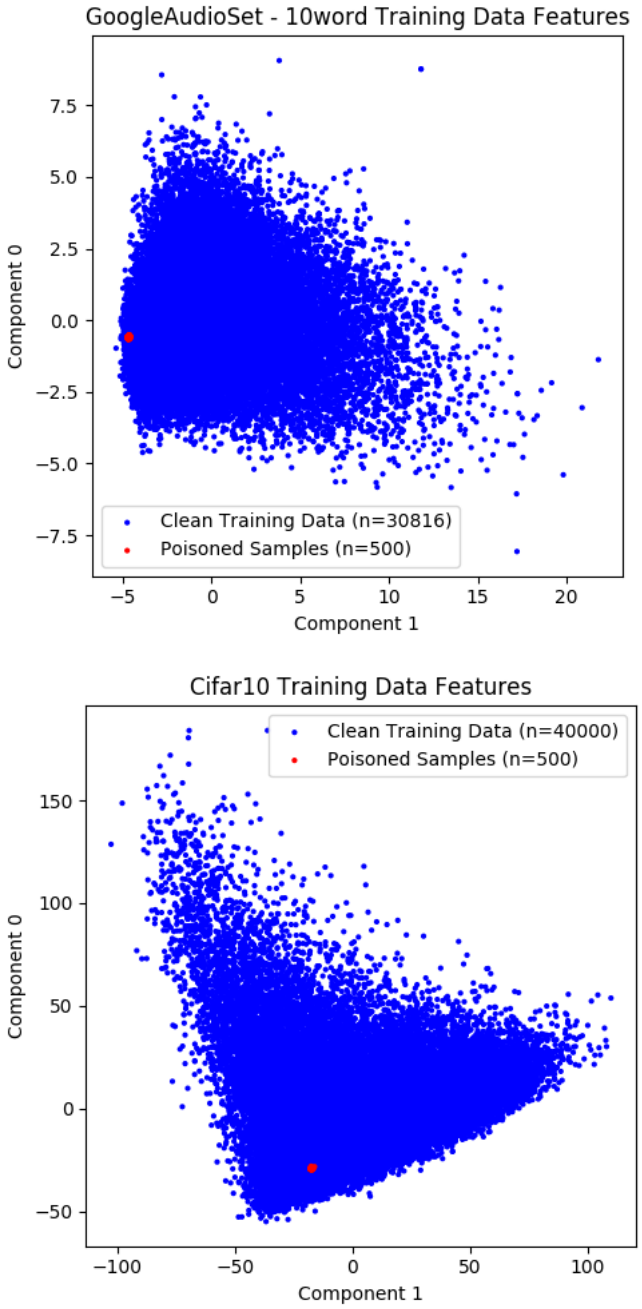


Figure 4.3: Visualisation of the transfer learner's feature space. The top image shows the first two principal components of the *Google AudioSet 10* dataset's features when using the YAMNet feature extractor. The bottom image shows the same visualization of the *Cifar10* dataset, using the *imagenet50* feature extractor. The blue dots represent the unpoisoned training data. The red dots represent poison data found via Equation 4.2, chosen equally per class (50 instances for each of the ten classes). Observe the large diversity of the unpoisoned training data compared to the adversarial poison data, which is projected onto a single point, thus creating a 'feature collision' which massively deteriorates the active learner's performance.

Table 4.1: Results of the feature collision poisoning on three data sets: Cifar10, STL and Google Audio Set. The Head of the Active Learner has one (NN1) or two (NN2) dense layers. The success rate provides the ratio with with poison samples are selected by the active learner.

Dataset	Model	Accuracy (Clean)	Accuracy (Poisoned)	Loss (adv.)	Loss (initital)	N	Success Rate (Poison)	Success Rate (Random)	Time (s)
STL	NN1	0.862	0.347	107.729	66450.624	500	1.0	0.126	113.08
Cifar10	NN1	0.432	0.267	39.746	5016.588	500	1.0	0.013	93.425
Audio Set 10	NN1	0.252	0.143	0.488	24.59	500	1.0	0.016	73.083
STL	NN2	0.844	0.7	107.729	66450.624	500	0.86	0.126	113.08
Cifar10	NN2	0.428	0.341	39.746	5016.588	500	0.832	0.013	93.425
Audio Set 10	NN2	0.263	0.141	0.488	24.59	500	0.998	0.016	73.083

4.4.3 Impact on the Model

In this section, we evaluate the impact on the classification accuracy of transfer active learners when exposed to the poison samples. For each of the three datasets, we create 500 poison samples and include them into the training set. We then create a transfer active learner (a neural network with one / two layers), train it on 20 unpoisoned samples, and simulate human annotation by letting the active learner query for 500 new instances from the training set (which contains a majority of benign data plus the injected poison samples). We find that the active learner

- almost exclusively selects poisoned samples, and
- test performance is degraded severely (by up to 50% absolute).

Table 1 details our results.

For example, consider the first row, which evaluates a one-layer neural network (NN1) on the STL-dataset. In an unpoisoned scenario, after having queried 500 images, the active learner has a test-accuracy of 86 percent. When introducing 500 poison instances, we observe the following: First, even though the poison instances are outnumbered 1:8, the active learner chooses 500 out of 500 possible poison instances for manual annotation - a success rate of 100 percent. In comparison, if the adversarial instances would be queried with the same probability as unpoisoned samples, only 12.6 percent of them would be chosen (random success rate). Secondly, observe that test accuracy is degraded significantly from 86 percent to 34 percent. This is because the model can not learn on data that, due to the feature collision attack, looks identical to one another for the dense head.

4.4.4 Hyper Parameters and Runtime

Table 1 details the run time to find a single adversarial example on an Intel Core i7-6600 CPU (no GPU), which ranges from one to two minutes. We used the following hyperparameters to find the adversarial samples: For the audio samples, we used $\beta = 0.3$ and 2000 iterations with early stopping and adaptive learning rate. For the image samples,

we used $\beta = 1e-5$ and 500 iterations. The difference in β between image and audio data is due to the difference in input feature scale, which ranges from ± 127 for images to $\pm 2^{15}$ for audio data.

4.4.5 Defending Against Adversarial Transfer Poisoning Attacks

How can one defend against such an attack? One possiblity would be to somehow break the adversarial collision, i.e. maniuplate either the feature extractor or the data s.t.

$$g(x') \neq \mu$$

for x' poisoned instances. There are two possibilites to achieves this:

- First, we can alter the feature extractor g. This could theoretically be achieved by unfreezing the model, i.e. re-training of the feature extractor g along with the dense model head. Such training on the poison samples actually serves as 'adversarial retraining', which has been shown to boost model robustness [79]. However, this is not a suitable approach due to the following concerns.
  - **Lack of labelled samples**. In order to unfreeze the weights of the feature extractor f, a large number of labelled samples is required. These are not available at the start of the active learning cycle. Thus, retraining can only occur during later stages. Until then, however, the adversary has free reign to introduce their poison samples into the dataset.
  - **High computational overhead**. Retraining the feature extractor f incurs high computational overhead in comparison to training only the dense head g.
  - **Overfitting**. Training f, especially on a small dataset, may result in overfitting, as described in Section 4.4.1.1.

  Thus, in summary, completely unfreezing the model is not a suitable defense. However, one might change g only slightly, for example by setting a small learning rate (e.g. $1e^{-6}$.). We implement an evaluate this approach and show results in table 4.2.

- Second, in order to break the relationship $g : x' \mapsto \mu$, we can introduce some perturbation $p(x')$ s.t.

  $$g(p(x')) \neq \mu$$

  The following perturbations are natural candidates, which we implement and evaluate, c.f. table 4.2:
  - **Channel swapping**, i.e. randomly permuting the red, green, blue (rgb) channels.
  - **Pixel Shifting**, i.e. shifting the complete image by 1, 2 or 3 pixels to the right. This causes the convolutional filters to misalign to the adversarial noise.
  - **Adding Gaussian Noise**, where we chose $\sigma = 12.75$, i.e. 5% of the feature range (0 to 255).

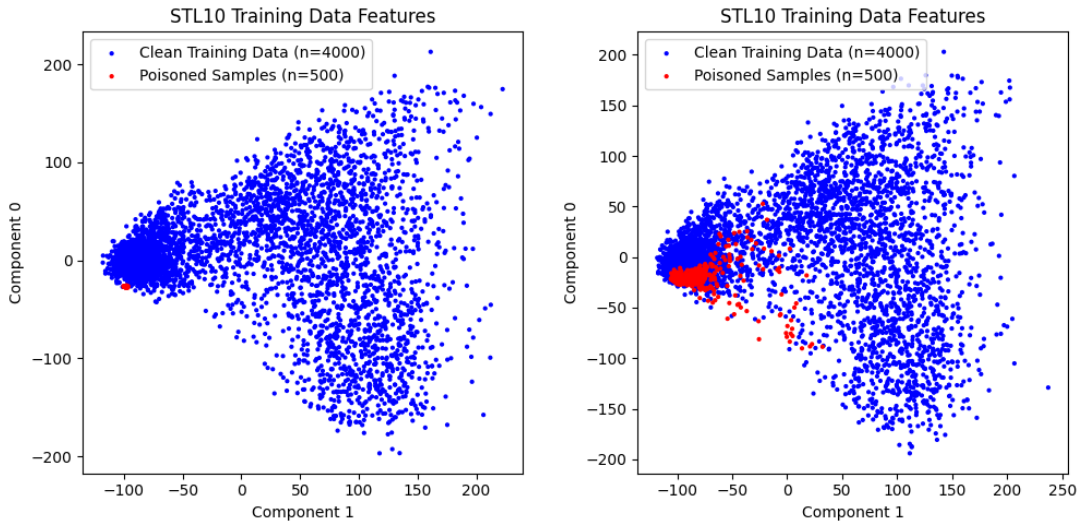


Figure 4.4: The effect of our proposed pixel-shift defense (right) vs. employing no defense (left). Observe how the adversarial samples are distributed much more evenly in the feature space, thereby not coinciding on the collision vector.

- **Image rotation** by 90 degrees. This, again, should misalign the adversarial noise and the convolutional filters of g.
- **Horizontal flipping**, which serves a similar purpose.

Additionally, we implement a naive 'dummy' defense, where we set all pixels to random values. This serves as a sanity check.

4.4.5.1 Evaluation

The following table 4.2 presents the results of our defense on the STL dataset. The image shifting defense is able to almost completely negate the loss in accuracy, whilte at the same time drastically reducing the number of poison samples to be chosen (from 100% to about 45%). The horizontal-mirroring defense performs best w.r.t. this metric (adversarial success rate of only 30%, compared to the baseline probability of 12%). However, it incurs significant loss of model accuracy (about 20%). To summarize, the pixel-shift defense works well, incurring no accuracy-penalty while at the same time largely reducing the adversarial success rate. The effectiveness of this defense is likely due to the fact that convolutional filters are invariant to small shifts in pixel space, since they focus on larger, semantic units, but adversarial noise is not.

Figure 4.4 visualizes the succes of the pixel-shift defense by plotting the feature-space of the training and adversarial data. We see that adversarial data no longer coincides strongly over the collision vector μ.

Model	Defense	Acc. Clean	Acc. Poisoned	Success Rate Pois.
NN1	no_defense	0.852	0.394	1.000
	dummy_defense	0.083	0.089	0.120
	swap_channels	0.799	0.734	0.458
	shift_img_1px	0.829	0.847	0.520
	shift_img_2px	0.829	0.853	0.474
	shift_img_3px	0.841	0.844	0.436
	add_gauss_noise	0.791	0.774	0.462
	rotate_img	0.674	0.625	0.400
	unfreeze_small_lr	0.816	0.356	1.000
	mirror_horizontally	0.687	0.688	0.286
NN2	no_defense	0.824	0.645	0.834
	dummy_defense	0.104	0.105	0.134
	swap_channels	0.751	0.744	0.436
	shift_img_1px	0.818	0.817	0.514
	shift_img_2px	0.832	0.815	0.444
	shift_img_3px	0.812	0.801	0.398
	add_gauss_noise	0.734	0.797	0.428
	rotate_img	0.673	0.652	0.356
	unfreeze_small_lr	0.816	0.678	0.812
	mirror_horizontally	0.678	0.687	0.302

Table 4.2: Performance of the proposed defense strategies on the STL dataset. The best results per column are highlighted. Note that the dummy_defense does achieve low poison success rate, but also breaks the model (see Accuracy score), so it is not a suitable defense, but rather a baseline or sanity check.

4.5 Conclusion

In this chapter, we present an intriguing weakness of the combination of active and transfer learning: By crafting feature collisions, we manage to introduce attacker-chosen adversarial samples into the dataset with a success rate of up to 100 percent. Additionally, we decreased the model's test accuracy by a significant margin (from 86 to 36 percent). The success rate for randomly choosing a poison samples is about 12%, since they poison samples are outnumbered by the benign instances by about $1:7$ (500 poison samples vs. 3500 benign samples). This attack can effectively break a model, wastes resources of the human oracle, and is very hard to detect for a human when reviewing the poisoned instances. Additionally, we show that by using simple input filters such as pixel-shift, we can mitigate the attack to some extent without compromising on model performance. To the best of our knowledge, this particular weakness of transfer active learning has not been observed before.

5 Identifying Mislabeled Data

The second step in the data-creation pipeline, c.f. figure 1.1, is the labeling of the sampled data. These labels involve human annotators, which can easily result in mislabelled instances, i.e. data instances x_i where the corresponding label y_i is incorrect. Since this poses problems for model training and, especially, evaluation, an efficient way to identify such mislabelled instances is required. This chapter presents our research on this subject. Specifically, we design and implement an algorithm that can find mislabelled instances in classification data. This allowed us to find a variety of mislabelled instances in popular, real-world datasets that have not been described before. Passages are quoted verbatim from work submitted to the International Joint Conference of Neural Networks 2019, where this research has been accepted as a full paper [3] and was first presented to the scientific community.

5.1 Motivation

As outlined in Chapter 1, machine learning relies on high-quality training data. A requirement for high-quality data is the correctness and reliability of the data labels. This means that for each instance (x, y), the label y has to match the semantic content of x. For example, an image of a horse x should have the corresponding label *horse* and not *butterfly*. The presence of such incorrect labels can be described as 'label noise' or the presence of data poisoning where the adversary acts randomly. As we will see in Chapter 7, this algorithm can be extended to mitigate the effects of *deliberate* adversaries, i.e. adversarial data poisoning.

Ensuring the correctness of the training data's labels is not trivial, because the human annotators who create these datasets make mistakes. This is a widely recognized fact:

> '...the process that generated the original labels had humans in the loop and errors creep in that way.'
> (Han Xiao [80], Creator of the Fashion-MNIST dataset)

For datasets containing as many as 60,000 instances (e.g., Fashion-MNIST), it is nearly impossible to manually find mislabelled data, such as shown in Figure 5.1. This is why a tool to help identify mislabelled instances is needed.

5.2 Related Work

In this section, we give an overview of related work w.r.t. identifying mislabelled instances in machine learning training data. We observe that there are three ways to deal with noisy

Figure 5.1: Mislabelled instances in the Fashion-MNIST training set. The heading shows the original label and the instance number w.r.t the training set. Note how the semantic content of the image does not fit the label, which is why these instances are considered 'mislabeled'. The correct labels should be 'dress', 'shirt', 'shirt', 'dress'.

	Completely At Random	**At Random**	**Not At Random**
Depends on	-	Class	Class and Features
Example	Dogs & Trees	Leopards & Lions	Yellow Cats & Lions

Table 5.1: Taxonomy of label noise as per Frenay et Al. [81, 82]

datasets. First, one may choose to design robust algorithms that can learn even from noisy data. Second, mislabelled instances can be identified and automatically removed from the dataset before training. Third, mislabelled instances can be identified and then re-evaluated by a domain expert. We now proceed to briefly present related work in each of these categories.

5.2.1 Taxonomy of Label Noise

It turns out the label noise comes in different varieties. A comprehensive overview of class label noise is provided by Frenay et Al. [81, 82], who summarize the literature on label noise and distinguish three sources of noise: completely at random, at random, and not at random. According to their definition, errors occurring *completely at random* are independent of the true class and the feature values. Put differently, everything is confused with everything else. Errors *at random* only depend on the class (e.g., confusing lions and leopards but not lions and trees). Finally, errors *not at random*, depend on the class as well as the feature values (e.g., confusing only big yellowish cats with lions but not black ones). Table 5.1 provides a summary.

5.2.2 Learning With Label Noise

The first step in dealing with mislabelled training data is to simply accept it, and try to design one's model around the label noise.

> If humans did their best reasonable effort when producing the dataset, then algorithms have to live with it. Han Xiao, [80]

Here, the usual approaches for designing robust models can be employed: Using Dropout [83], high learning rates, or regularisation are good starting points.

Additionally, one may employ algorithms specifically designed to be robust against label noise [84, 85]. Finally, there are semi-supervised methods such as [86], which assume the presence of a larger noisy, and a smaller clean dataset, which can be then used to improve robustness to label noise on the larger dataset.

5.2.3 Label Cleansing

A second approach consists of finding and removing mislabelled instances. These methods are usually some sort of anomaly or outlier detection, bootstrapped on a smaller, clean dataset.

For example, Brodely et Al. [87] propose to first train filters on parts of the training data to identify mislabelled examples in the remaining training data. To this end, they use cross-fold validation among decision trees, nearest neighbor classifiers, and linear machines. Mislabelled instances are subsequently removed from the training set. Then, a learning algorithm is trained on the reduced training set. Sabzevari et Al. [88] train random forests on bootstrapped samples to create ensembles. A threshold for the disagreement ratio of the classifiers is introduced to detect mislabelled data. The downside of these approaches is that they diminish the size of the dataset.

5.2.4 Label Noise Identification

A third class of approaches, to which our proposed algorithm belongs, identifies and corrects mislabelled instances. While possibly more costly, such approaches yield a larger training set. Often, these approaches are conceptually similar to the 'label cleansing' methods from the previous subsection and simply omit the re-labeling step.

Ekambaram et Al. [89] present a two-level approach to identifying mislabelled instances in binary classification data: First, a support vector machine (SVM) is trained on the dataset. The support vectors (points close to the decision boundary) are considered noisy, and the rest of the data is considered clean. A second classifier is trained on the clean data and used to re-evaluate the noisy data. Each mismatch is presented to a human annotator for judgment. After reviewing over 15% of the images in ImageNet (about 7500 images), the authors identify 92 mislabelled pictures. In comparison, our proposed method has a much higher yield, c.f. Section 5.4.

Another approach is presented by Alrawi et Al. [90]. They train a convolutional neural network (CNN) ensemble on parts of the dataset to evaluate the remaining dataset via majority voting. This method is applied to different image datasets (CIFAR10, CIFAR100, EMNIST, and SVHN). We compare their method against ours, c.f. Section 5.4.3.

Although most publications have focused on image data, similar techniques can be applied to text documents [91] and e-mail spam filtering [92]. In this context, however, the problem of ambiguity becomes more prominent: Even for a domain expert, it is not always clear what the correct label should be. This is especially true for regression data, where a domain expert is needed to fill the role of the human oracle. Note that in the domain of machine learning, an oracle is some entity with absolute knowledge, usually about the target variable y. It contrasts with the learning model's prediction in that the

oracle is assumed to always be correct. Usually, the oracle is a human who can be queried for the correct label of an instance. Thus, the number of queries is limited and the oracle has to be queried sparingly.

5.3 Methodology

The following section provides our proposed algorithm to identify mislabelled instances in a given classification dataset $\mathcal{D} = (x, y)$.

5.3.1 Problem Statement

Assume a dataset (x, y) of size N and dimensionality D, where each instance pertains to one of C classes. We can encode the targets y as a one-hot matrix of shape (N, C), and the data x as a matrix of shape (N, D)

We define $\mathcal{I} \subseteq \mathcal{D}$ as the set of mislabelled instances. Hence, $\mathcal{I} = \emptyset$ if there are no mislabelled instances in $\mathcal{D}$. The set $\mathcal{I}$ is, of course, unknown to the user but for every index $n \in \{0, \ldots, N-1\}$ a domain expert could manually check whether $(x_n, y_n) \in \mathcal{I}$, i.e. if the instance is mislabelled or not. We refer to some domain expert reviewing every single $d \in \mathcal{D}$ as the *naive approach*. As detailed in the introduction, the naive approach is often prohibitively expensive. We assume that $|\mathcal{I}| \ll N$, because otherwise there is not much room to improve upon the naive approach.

To reduce the number of required reviews by the human annotator, our goal is to construct a mapping

$$f_\alpha : \mathcal{D} \to \mathcal{I}_\alpha$$

for some $0 < \alpha < 1$ (determining the number of instances to be reviewed) such that $|\mathcal{I}_\alpha \cap \mathcal{I}|$ is maximized while $|\mathcal{I}_\alpha|$ is minimized. Here, $\mathcal{I}_\alpha$ where $|\mathcal{I}_\alpha| = \alpha N$ is the set of instances that are to be reviewed by a domain expert. Thus, we aim to construct a system f_α that suggests a set of potentially mislabelled instances $\mathcal{I}_\alpha$. Our goal is to have the set $\mathcal{I}_\alpha$ contain as many truly mislabelled instances as possible.

To make the system user-friendly, we design it to require a minimal set of hyper parameters: Those are α (determining the size of the output set) and the labelled dataset $\mathcal{D}$. As we show below, one may even omit the α parameter, which makes the system entirely free from hyperparameters. As noted above, this makes our proposed system highly applicable in practice.

5.3.2 Proposed Algorithm

To detect mislabelled instances in a given dataset $\mathcal{D}$, we propose the following pipeline.

1. Supply a dataset $\mathcal{D} = (x, y)$ and, optionally, a parameter α. Here, α is the percentage of images the user is willing to manually re-evaluate.
2. Find appropriate preprocessing, network layout, and model hyperparameters automatically as detailed in Section 5.3.4.

3. Preprocess the dataset (x, y) (c.f. Section 5.3.3) and train a model g on it.
4. Use the trained model g to re-classify x, and obtain $g(x) = \tilde{y}$.
5. For all $n \in \{0, \ldots, N-1\}$, compute the inner product

$$\langle y_n, \tilde{y_n} \rangle = \sum_j y_n^j \tilde{y}_n^j. \tag{5.1}$$

   This yields the probability that instance x_n is assigned the original label y_n.

6. Finally, return the set

$$\mathcal{I}_\alpha \subset \{0, ..., N-1\} \text{ where } |\mathcal{I}_\alpha| = \alpha N \tag{5.2}$$

   such that $\sum_{\mathcal{I}_\alpha} \langle y_i, \tilde{y}_i \rangle$ is minimized. This can be implemented efficiently by sorting $\langle y_n, \tilde{y_n} \rangle_{n \in \{0,\ldots,N-1\}}$ by argument in ascending order and then returning the first αN instances. Alternatively, the user may choose $\alpha = 1$ and review the returned, ordered instances until resources such as time or budget are exhausted. Since the probability of being mislabelled decreases with every instance, this will optimally utilize available resources.

In summary, train a classifier g on a given dataset (x, y), and use the same classifier g to obtain class probabilities for x. Then look for instances for which the class probability of the original label is minimal, e.g., instances for which the classifier considers the original label extremely unlikely given the feature distribution learned during training. These are the instances that are most likely mislabeled.

5.3.3 Data Preprocessing

To properly learn from a user-supplied dataset, appropriate data preprocessing is essential. We preprocess numerical, image, and natural language data as follows.

- Numerical data is preprocessed by feature-wise scaling to $[0, 1]$. This process is known as *MinMax scaling*.
- Image data is preprocessed by feature-wise setting the data mean to 0 and feature-wise dividing by the standard deviation (std) of the dataset. This process is known as *standardization*.
- Natural language data is preprocessed using word embeddings [93] as follows: First, we apply very basic textual preprocessing such as splitting on whitespaces, then we map individual words to 300-dimensional word embeddings using a pre-trained embedding [94], and finally, we retrieve the corresponding representation for the whole sequence of words by simply summing up the individual embeddings.

5.3.4 Classification Algorithm

To find mislabelled instances in a dataset as described in Section 5.3.2, a robust classifier g is required. That is to say, g must be able to generalize well and not overfit the

	Numerical	Textual
Depth	{1, 2, 3, 5}	{1, 2, 3, 5}
Units per Layer	{50, 120}	{50, 120}
Dropout per Layer	{0, 0.1, 0.2}	{0, 0.1, 0.2}

Table 5.2: Parameter grid for cross-validation based hyperparameter optimization. For Image data, we use a convolutional network with fixed hyperparameters.

dataset $\mathcal{D}$. This is necessary because since $\mathcal{D} = (x, y)$ is assumed to be noisy (due to mislabelled instances), overfitting on $\mathcal{D}$ results in the classifier simply 'remembering' all instances (x_n, y_n) for $n \in \{0, \ldots, N-1\}$. In this scenario, deviations in individual (x_n, y_n) from the overall distribution would not be found, thus severely diminishing the system's performance.

Hence, g needs to 1) be flexible to correctly learn from a variety of text, image, and numerical datasets and 2) generalize well on noisy datasets. Neural networks, especially with Dropout, are a natural choice in this setting. How we apply them to the individual categories of datasets is detailed in the following subsection.

5.3.5 Automatic Hyperparameter Selection

This chapter describes how we find suitable model architectures and hyperparameters for each type of input data automatically. This relieves the user of this task and makes our pipeline more easily accessible.

- For numerical and textual data, we use a dense feed-forward neural network and optimize the following hyperparameters: Number of hidden layers, number of units per hidden layer, and dropout rate. We use 3-fold cross-validation to search the space of suitable hyperparameters (see Table 5.2) for the best combination of parameters. For preprocessing, we apply data normalization. Additionally, textual data is mapped to a vector representation using word embeddings.

- For image data, we use a convolutional network . It consists of three blocks: Block one consists of two convolutional layers with 48 2x2 kernels, followed by 3x3 max-pooling and 25% dropout. Block two is the same as block one, but with 96 kernels. Block three flattens the convolutional output of block two and applies three dense layers with 50% Dropout and *ReLu* non-linearities. We chose this architecture because convolutions work for all kinds of input images, as convolutional layers can cope with varying image input sizes. Note that we omit the hyperparameter grid search in the interest of computation time. Overfitting is prevented by using high dropout (between 25 and 50% per layer) and early stopping on the validation accuracy. We also experimented with transfer learning, using a ResNet50 as a feature extractor, followed by a dense network. However, we find that this approach is inferior to our convolutional network in terms of precision and recall.

We chose these architectures since they are standard baselines that have been shown to work with the corresponding data types [10, 95, 96]. Since we want the networks

to be able to generalize well and learn even from noisy data, we need to prevent the networks from overfitting, i.e., simply 'remembering' the training data. Thus, we select the hyperparameters via cross-validated grid search. Additionally, we add several levels of Dropout to the parameter grid and use an aggressive learning rate of 1e-2, which also cuts computation time. Also, we use early stopping with *patience*= 15 and *threshold*= 0.005. The final layer feeds into a softmax activation function, which returns class probabilities.

Finally, to deal with imbalanced class labels, we adjust the gradients during training using class weights and employ balanced *F-score* as an evaluation metric.

5.4 Evaluation

We evaluate our system in two ways:

- **Quantitative evaluation**. We create mislabelled instances ourselves by changing the labels in a variety of datasets. Then, we apply our tool and try to identify them.
- **Qualitative evaluation**. We apply our tool to popular, real-world datasets and try to find new mislabelled instances.

In the following section, we describe this evaluation in detail and present the results. We make all experiments accessible by publishing our source code, c.f. `https://github.com/mueller91/labelfix`.

5.4.1 Quantitative Evaluation

To evaluate the effectiveness of our system quantitatively, we proceed as follows.

1. We select 29 datasets from the domain of text, image, and numerical data, for each of which we change μ percent. Here, μ denotes the ratio of mislabelled instances. For example, $\mu = 3\%$ if we assume that 3% of the instances in a dataset have incorrect labels. This yields a new, noisy dataset $\mathcal{D}'$.
2. We apply our algorithm to this noisy dataset $\mathcal{D}'$, c.f. Section 5.3.2.
3. We evaluate the output $\mathcal{I}_\alpha$ of our algorithm by computing the *precision* and *recall* (c.f. Equation (5.3) and Equation (5.4)).

5.4.1.1 Datasets

We use the following datasets (c.f. Table 5.3).

- From `sklearn` [97]: breast cancer, digits, forest cover type, iris, twenty newsgroup, and wine;
- from `kaggle.com`: pulsar-stars, Sloan digital sky survey
- from the `UCI Machine Learning Repository` [98]: adult, credit card default [99], SMS spam [100], and liver;

- from `keras` [101]: CIFAR-10 and CIFAR-100[11], Fashion-MNIST [13], IMDb [102], MNIST [10], and SVHN [103].

Additionally, we create six synthetic datasets with varying feature and target dimensionality (see Table 5.3) using *sklearn*'s *make_ classification* functionality plus one *blob* dataset using *sklearn*'s *blob* functionality.

5.4.1.2 Introducing Artificial Label Noise

How to find a suitable fraction of label noise μ for our quantitative evaluation? Estimating the ratio of mislabelled instances μ for real-world datasets is intractable and may differ significantly between datasets. However, [81] suggests that real-world datasets have around 5% mislabelled data. Since we use mainstream datasets, we chose a smaller value $\mu = 0.03$ (i.e. 3%), because we believe that a 5% error rate may be too large for these well-researched datasets. We find that for other choices of μ, the results of our algorithm are comparable. These may be easily recreated by running the provided script with different parameters.

We simulate both 'noise completely at random', i.e. noise independent of either class and feature values, and 'noise at random', i.e. noise dependent on the class, but not the feature values. We do not evaluate labeling mistakes that depend both on class and feature values (i.e. 'not at random' label noise, c.f. table 5.1). This is because these errors are so specific that they do not apply to the large majority of datasets, where classes are very distinct.

5.4.1.3 Metrics

We report two different metrics: the α-precision and the α-recall:

$$\alpha\text{-precision} = \frac{|\mathcal{I}_\alpha \cap \mathcal{I}|}{|\mathcal{I}_\alpha|}, \tag{5.3}$$

$$\alpha\text{-recall} = \frac{|\mathcal{I}_\alpha \cap \mathcal{I}|}{|\mathcal{I}|}. \tag{5.4}$$

The α-precision (see Equation (5.3)) returns the ratio of the number of flipped labels among the returned αN instances to the output size αN (i.e., how many of the system's suggestions are indeed wrongly labeled?). For example, an α-precision value of 0.8 for $\alpha = 0.01$ and $\mu = 0.03$ for some dataset $\mathcal{D}$ has the following interpretation: Among the set $\mathcal{I}_\alpha$, which has size $0.01N$, 80% flipped labels are found. Hence, this method is more than 26 times better than random guessing, but still returns 20% correctly labelled instances to be re-checked.

The α-recall (see Equation (5.4)) specifies how many of the flipped labels are found by looking only at the first α percent of the data (i.e., how many of the wrongly labelled instances are present in the system's output suggestions?). For example, assume the following: Let $\alpha = 0.03$ and $\mu = 0.03$ for some dataset $\mathcal{D}$. A recall value of 0.8 means that we find 80% of the 3% flipped labels if we review only $0.03N$ instances as suggested by our system.

Dataset	Size	Type	Classes
adult	(32561, 14)	numerical	2
breast_cancer	(569, 30)	numerical	2
cifar10	(50000, 32, 32, 3)	image	10
cifar100	(50000, 32, 32, 3)	image	100
cifar100, at random	(50000, 32, 32, 3)	image	100
cifar100, subset aqua	(2500, 32, 32, 3)	image	5
cifar100, subset flowers	(2500, 32, 32, 3)	image	5
cifar100, subset household	(2500, 32, 32, 3)	image	5
credit card default	(30000, 23)	numerical	2
digits	(1797, 64)	numerical	10
fashion-mnist	(60000, 28, 28, 3)	image	10
forest covertype (10%)	(58101, 54)	numerical	7
imdb	(25000, 100)	textual	2
iris	(150, 4)	numerical	3
mnist	(60000, 28, 28, 3)	image	10
pulsar_stars	(17898, 8)	numerical	2
sloan-digital-sky-survey	(10000, 17)	numerical	3
sms spam	(5572, 300)	textual	2
svhn	(73257, 32, 32, 3)	image	10
synthetic 1	(10000, 9)	numerical	3
synthetic 2	(10000, 9)	numerical	5
synthetic 3	(10000, 45)	numerical	7
synthetic 4	(10000, 45)	numerical	15
synthetic 5	(10000, 85)	numerical	15
synthetic 5	(10000, 85)	numerical	7
synthetic blobs	(4000, 12)	numerical	12
twenty newsgroup	(18846, 300)	textual	20
twitter airline	(14640, 300)	textual	3
wine	(178, 13)	numerical	3

Table 5.3: List of datasets used for evaluating [3]

Dataset	Runtime	α-precision			α-recall		
		$\alpha = 0.01$	$\alpha = 0.02$	$\alpha = 0.03$	$\alpha = 0.01$	$\alpha = 0.02$	$\alpha = 0.03$
adult	2.1 min	0.80	0.63	0.51	0.27	0.42	0.51
breast_cancer	34.0 sec	0.76	0.80	0.74	0.22	0.52	0.74
cifar10	9.47 min	0.98	0.88	0.72	0.33	0.59	0.72
cifar100	13.07 min	0.94	0.82	0.67	0.31	0.54	0.67
cifar100, at random	11.48 min	0.43	0.35	0.31	0.14	0.23	0.31
cifar100, subset aqua	20.2 sec	0.61	0.38	0.32	0.20	0.25	0.32
cifar100, subset flowers	32.8 sec	0.63	0.43	0.34	0.21	0.29	0.34
cifar100, subset household	48.6 sec	0.62	0.46	0.37	0.21	0.30	0.37
credit card default	1.9 min	0.18	0.17	0.18	0.06	0.12	0.18
digits	51.8 sec	0.98	0.95	0.86	0.31	0.63	0.86
fashion-mnist	10.71 min	0.99	0.98	0.90	0.33	0.66	0.90
forest covertype (10%)	4.6 min	1.00	0.95	0.74	0.33	0.63	0.74
imdb	3.71 min	0.70	0.61	0.51	0.23	0.41	0.51
iris	26.9 sec	1.00	0.53	0.55	0.25	0.40	0.55
mnist	3.74 min	1.00	1.00	0.97	0.33	0.67	0.97
pulsar_stars	51.8 sec	0.91	0.86	0.78	0.30	0.57	0.78
sloan-digital-sky-survey	1.5 min	0.80	0.71	0.63	0.27	0.47	0.63
sms spam	1.44 min	0.85	0.86	0.79	0.28	0.57	0.79
svhn	13.6 min	0.92	0.90	0.83	0.31	0.60	0.83
synthetic 1	2.05 min	1.00	0.98	0.89	0.33	0.66	0.89
synthetic 2	2.74 min	1.00	0.99	0.89	0.33	0.66	0.89
synthetic 3	3.79 min	1.00	0.99	0.91	0.33	0.66	0.91
synthetic 4	4.9 min	0.98	0.90	0.74	0.33	0.60	0.74
synthetic 5	3.53 min	0.95	0.84	0.70	0.32	0.56	0.70
synthetic 6	3.58 min	1.00	0.98	0.86	0.33	0.65	0.86
synthetic blobs	37.8 sec	1.00	1.00	0.98	0.33	0.67	0.98
twenty newsgroup	3.2 min	0.79	0.73	0.63	0.26	0.49	0.63
twitter airline	2.39 min	0.66	0.52	0.43	0.22	0.34	0.43
wine	28.1 sec	1.00	1.00	0.88	0.20	0.60	0.88
Average		0.84	0.77	0.68	0.27	0.51	0.68

Table 5.4: Precision and recall values for artificially added 3% noise, averaged over five runs.

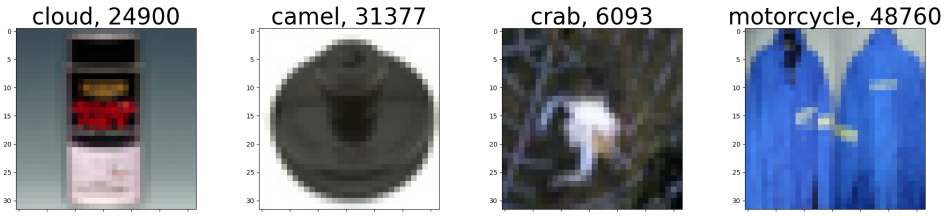


Figure 5.2: Four mislabelled instances in the CIFAR-100 training set, with corresponding label and index of the image in the dataset.

As the returned instances are already sorted by their probability to be mislabeled, we expect the α-precision to become lower for higher values of α as fewer new mislabelled instances are added to $\mathcal{I}_\alpha$. On the other hand, the α-recall increases for higher values of α, as $\alpha = 1$ implies that $\mathcal{I}_\alpha \subseteq \mathcal{I}$, hence α-recall $= 1$. Furthermore, for $\alpha = \mu$, these metrics coincide, i.e. α-precision $= \alpha$-recall (see Table 5.4 for $\alpha = \mu = 0.03$).

5.4.1.4 Results

We run our tool on 29 datasets with $\alpha = 0.03$, and present the results in Table 5.4. We observe the following:

1. **Effectiveness**. Our algorithm works well on all types of input data, i.e. images, natural language, and numerical data. For example, the $\alpha = 0.01$ precision is 0.84, meaning that a user reviewing the first percent of the algorithm's output would find, on average, 84 mislabelled instances for every 100 instances reviewed. This significantly outperforms the random baseline, where a user would find only 3 mislabelled instances for every 100 images examined.
2. **Sensitivity**. The algorithm detects both severe, obvious labeling errors among very different classes (e.g. bridge vs. apple) as well as nuanced errors among similar classes (e.g. rose vs. tulip, both from the CIFAR-100 subset flowers dataset, see 'cifar-100, at random'),
3. **Efficiency**. The algorithm has reasonable computing time, ranging from seconds to minutes, depending on the dataset size and the number of classes. We use an Intel E5-2640-based machine and assigned 20 cores to perform our computations. The convolutional network is trained on an Nvidia Titan X GPU.

In comparison to [90], we are training only one neural network, which saves time while achieving very good results (see Table 5.4). Furthermore, our approach is more general since, for example, natural language can be processed as well. As opposed to [89], we do not require additional information such as classes that are easily confused.

5.4.2 Qualitative Evaluation

For the quantitative evaluation in Section 5.4.1, we have evaluated artificially noisy data where we added $\mu = 3\%$ noise. Now, we perform a qualitative evaluation: We apply our

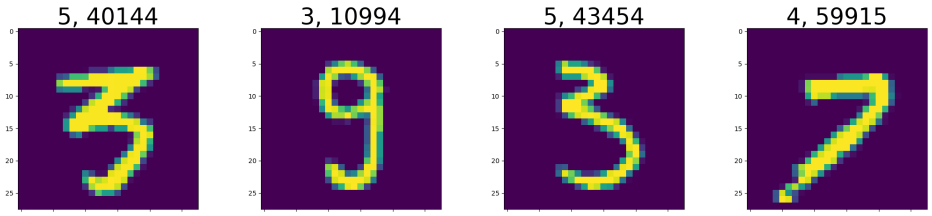


Figure 5.3: Four mislabelled instances in the MNIST training set. The headings indicate the original label and the index of the image.

algorithm to popular real-world datasets without adding extra noise, hoping to find truly mislabelled instances. The datasets we consider in this section contain either images or natural language since we can easily spot mislabelled instances in these domains.

Setting $\alpha = 0.003$ or 0.3%, we manually review 150 images in CIFAR-100 and 180 images in Fashion-MNIST and inspect for mislabelled instances. Whine in doubt, we stick to the assigned label. Following this procedure, we detect labeling errors that, to the best of our knowledge, have not been reported yet. Our findings include the following mislabelled instances.

- **CIFAR-100**. We identify seven mislabelled instances. Four examples are shown in Figure 5.2; the full results are presented in Table A.1 in the Appendix;
- **Fashion-MNIST**. We identify over 60 mislabelled instances within the first 180 instances. Some examples are shown in Figure 5.1; for a complete list, refer to Table A.2 in the Appendix.
- **MNIST**. We also present four mislabelled instances from the MNIST dataset, c.f. Figure 5.3.

To show that our tool can also process natural language, we apply it to the *twenty newsgroup dataset*, a collection of e-mails, each of which is assigned a group label. Our algorithm finds instances for which the class label and the e-mail content do not match, see Table 5.5 and Table 5.6.

Note that the list of instances we present for each dataset is not supposed to be complete, but rather intends to illustrate that even well-researched datasets contain mislabelled instances. Hence, a tool to simplify the process of re-checking is of great importance.

5.4.3 Improvement Over Related Work

With our algorithm, we overcome a fundamental restriction imposed by Ekembaram et Al. [89], who require the user to supply a set of classes likely to be confused. Another restriction is that their approach, at its core, is limited to binary classification problems. Application to multi-class data is possible but does not scale. In contrast, our algorithm applies naturally to multi-class data and does not require auxiliary input whatsoever.

In comparison to [90], we are more rigorous in considering something as 'mislabeled'. Where Alrawi et Al. accept instances with ambiguous labels as mislabeled [90], we only

Table 5.5: An example of an instance from the twenty newsgroup dataset where the content does not meet the label.

Index Nr. 13622
Category: sci.med
From: turpin@cs.utexas.edu (Russell Turpin) Subject: Meaning of atheism, agnosticism (was: Krillean Photography) Sci.med removed from followups. (And I do not read any of the other newsgroups.) >As a self-proclaimed atheist my position is that I _believe_ that >there is no god. I don't claim to have any proof. I interpret >the agnostic position as having no beliefs about god's existence. As a self-proclaimed atheist, I believe that *some* conceptions of god are inconsistent or in conflict with fact, and I lack belief in other conceptions of god merely because there is no reason for me to believe in these. I usually use the word agnostic to mean someone who believes that the existence of a god is unknown inherently unknowable. Note that this is a positive belief that is quite different from not believing in a god; I do not believe in a god, but I also do not believe the agnostic claim.

Table 5.6: A second example of an instance from the twenty newsgroup dataset where the content does not meet the label.

Index Nr. 13203
Category: talk.politics.mideast
From: kevin@cursor.demon.co.uk (Kevin Walsh) Subject: Re: To All My Friends on T.P.M., I send Greetings Reply-To: Kevin Walsh <kevin@cursor.demon.co.uk> Organization: Cursor Software Limited, London, England Lines: 17 In article OAF.93May11231227@klosters.ai.mit.edu oaf@zurich.ai.mit.edu writes: >In message: C6MnAD.MxD@ucdavis.edu Some nameless geek szljubi@chip.ucdavis.edu writes: »To Oded Feingold: » »Call off the dogs, babe. It's me, in the flesh. And no, I'm not »Wayne either, so you might just want to tuck your quivering erection »back into your M.I.T. slacks and catch up on your Woody Allen. » >This is an outrage! I don't even own a dog. > Of course you do. You married it a while ago, remember?

report those which are clearly mislabeled. Furthermore, our approach is more efficient since we use only a single model instead of an ensemble. Additionally, we require no test-train split and can therefore process smaller datasets more accurately. The biggest improvement however is that our tool is not limited to image data, but can be applied to numerical and natural language data as well.

5.5 Summary

In this chapter, we presented an automated system to help find potentially mislabelled instances in classification datasets. This has the potential to improve classifiers trained on these datasets, especially when the datasets have, unlike CIFAR, not been exposed to a large audience and can only be reviewed by a few domain experts. Since we cannot assume these domain experts to be proficient in machine learning techniques, we designed our system such that no hyperparameters have to be supplied – the system works out of the box for numerical, image, and text datasets. The only input data required is the dataset to be analyzed and a parameter that identifies the size of the output set. Hyperparameters are inferred automatically to improve the tool's usability.

We evaluate our system on over 29 datasets on which we add a small fraction of label noise, and find mislabelled instances with an averaged precision of 0.84 when reviewing our system's top 1% recommendation. Applying our system to real-world datasets, we find mislabelled instances in CIFAR-100, Fashion-MNIST, and others (see Appendix). Finally, please note that passages in this chapter have been quoted verbatim from our IEEE paper, where this research has been first published [3]. As we will see in chapter 7, this algorithm can be extended to detect *adversarial data poisoning*.

A limitation of this approach is that it will not detect mislabelled data that is consistently mislabeled. For example, if for a given dataset of animals, all horses are labelled *cat*, our proposed method cannot detect this. This, however, is a principled limitation of the pattern-recognition approach of machine learning. Without external feedback, no algorithm will be able to detect this. We note that such extreme cases are very rare, even in a *Data Poisoning* scenario, because an attacker would need to mislabel almost every single *horse* instance in the dataset. Focusing on the much more common case of *erroneous* labels, our algorithm can reliably identify those instances that do not correspond to the features of the majority of the class instances.

6 Identifying Adversarially Poisoned Data in Regression Learning

The last chapter introduced an algorithm to find mislabelled instances due to human error or a *randomly acting adversary*. This chapter introduces algorithms to find mislabelled instances originating from a different process: A *deliberate adversary*, who aims at maximally degrading the model. In literature, this is called an *adversarial attack*.

We study adversarial attacks in the context of regression learning (this chapter) and classification learning (c.f. Chapter 7). Please note that this chapter quotes verbatim from our conference paper [4], which has been published at the 25th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC 2020), where the work presented in this chapter has first been presented to the scientific community.

This chapter is structured as follows: Section 6.1 motivates the threat of adversarial attacks in the domain of regression learning and outlines our contribution. Section 6.3 presents a medical use case where data poisoning is detrimental, and Section 6.4.2 presents related work. We introduce a new data poisoning attack in Section 6.4.3, against which we evaluate our newly proposed defense in Section 6.5. The evaluation is presented in Section 6.6.

6.1 Motivation

In this chapter, we argue for the importance of secure and reliable regression learning and motivate the problem of data poisoning in this domain.

Regression learning is increasingly used in mission-critical systems: In medicine for the development of pharmaceuticals [104, 105], in the financial sector for predictive analysis such as managing hedge funds [106, 107] and cash forecasting [108], as well as for predictive maintenance [109] and quality control [110].

As we rely more and more on these systems, researchers find that they are vulnerable to malicious data poisoning attacks, which can result in significant real-world casualties. As an example, consider the following scenario. A widely used blood thinner is the *Warfarin* drug. It has a very small therapeutic window, meaning that too high dosages lead to bleeding, while too low dosages lead to blood clotting [111]. Machine learning can help in estimating the correct dosage, and pharmaceutic companies provide appropriate datasets [112] on which regression learning has been successfully applied [113, 114, 115].

However, such an estimation is susceptible to data poisoning attacks: An adversary may introduce a very small percentage of 'poisoned samples' into the dataset. Motives for such an attack can be manifold: Personal motives (a malignant doctor, underpaid caregiver or simply a psychopath nurse [116]), financial motives (one company damaging

another company's reputation, or an individual betting on the crash of some company's stock value, similar to [117]), or even political or terrorist motives.

Such poisoning attacks are not just theoretical threats, but can be executed even with moderate resources: Because of the narrow therapeutic window of the *Warfarin* drug, and (as we will show) the large impact of even only a few malicious samples, data poisoning can be realized even by an individual with a significant impact on the learner. Related work confirms this, showing that data poisoning is feasible and has already been observed in real-life scenarios [31, 118].

6.2 Contribution

To address this issue, in this chapter, we present work from our seminar paper [4], where me and my co-authors contribute as follows:

- **Use Case**. We show the harmfulness of data poisoning in regression learning by analyzing the Warfarin dosage prediction use-case.
- **New Attack**. We present a new black-box attack that exceeds previous state-of-the-art, and for the first time evaluate poisoning attacks on nonlinear regression learners. We present this attack to have a strong baseline to estimate attacker-capability under *real-world* conditions.
- **New Defense**. We present an improvement over previously suggested defenses that consistently outperforms SOTA.
- **Evaluation**. We thoroughly evaluate our attack and defense on 26 datasets and several state-of-the-art regression learners, i.e. Neural Networks, Kernel SVR, and Kernel Regression.
- **Tooling and Reproducibility**. We publish all source code and experiments to enable full reproducibility.

6.3 Case Study: Warfarin Dosage Estimation

In order to further motivate our research, this section demonstrates the feasibility and practicality of adversarial data poisoning on mission-critical systems: We examine the medical use case of Warfarin dosage prediction.

The International Warfarin Pharmacogenetics Consortium, a group of pharmacogenetic research centers, have created the *IWPC* or *Warfarin dataset* [112]. It is the joint effort of 59 contributors, resulting in an average contribution of 1.7% of the data per member. Based on this dataset, models have been developed which predict the therapeutic dosage of Warfarin for a patient [113, 114, 115].

We use a new black-box poisoning attack (to be detailed in Section 6.4.3) and add 2% poison data to the Warfarin dataset, which is about as much as the average IWPC contributor did. Table 6.1 shows the Mean Absolute Error (MAE) of different models after training on this dataset. In the absence of poison samples, the MAE of models

Model	Clean Error	Poisoned Error	Increase in MAE	Decrease in acceptable Dosages
Elastic Net	8.52	11.07	1.30	21.25%
Huber Reg.	8.40	8.46	1.01	-1.09%
Kernel Ridge	8.41	10.98	1.31	21.07%
Lasso	8.49	11.20	1.32	22.31%
MLP	10.05	12.33	1.23	11.41%
Ridge Reg.	8.49	10.99	1.29	22.34%
SVR	10.85	12.52	1.15	14.13%
Median	8.49	11.07	1.29	21.07%

Table 6.1: Mean absolute error (MAE) of different regression models when poisoning the *Warfarin* dataset. The first column (Clean Error) shows the MAE when no data poisoning is present. The second column (Pois. Error) shows the MAE when 2% poison samples are introduced, with the relative change indicated by the third column. The poisoning strongly affects the number of patients who receive an *acceptable dosage* of Warfarin (fourth column). For example, when using the Elastic Net model, 21.25% percent of the patients no longer receive an acceptable dosage due to 2% data poisoning. Note that the only regressor not impacted by this is the *Huber Regressor*, which is specialized in dealing with poisoned data.

like Lasso, Elastic Net, and Ridge is around 8.50, which is comparable to state of the art [114]. When adding just 2% of data poisoning, the median error increases to 11.07 (a 29 percent increase): In comparison to its size, the poison data has an influence 14 times as high on the training error. This has tangible effect on the patients: The rate of acceptable dosages decreases by 21%, due to just 2% of poison data. Here, we follow Ma et Al. [114]: we examine the percentage of patients whose predicted dosage of Warfarin is within 20% of the true therapeutic dosage. This is referred to as an *acceptable dosage*. We will revisit this use case in more detail in Section 6.7, where we show how to defend against such attacks.

6.4 Data Poisoning in Regression Learning

In this section, we present our threat model and examine previously suggested attacks. Additionally, we introduce a new and improved attack, which is thoroughly evaluated on 26 datasets in Section 6.6. Our motivation for presenting this new attack is to have a stronger baseline against which to evaluate our newly proposed defense (c.f. Section 6.5). Existing attacks are either computationally infeasible, or require (black-box) access to to trained model, which may not be realistic (c.f. section 6.4.2.2). Our attack is completely independent of the model, while at the same-time outperforming related attack, and thereby suited for realistic estimation of attacker capability.

6.4.1 Threat Model

We consider a realistic attack scenario where the attacker has only limited capabilities, such as a malicious individual could have. Specifically, we consider black-box attacks where

- the attacker knows nothing about the model (not even what kind of regressor is used),
- the attacker does not have access to the training dataset (X, y), but only to a smaller substitute dataset (X^{sub}, y^{sub}), and
- where the attacker is capable of fully controlling the ϵn data samples he contributes to the dataset. He is not able to manipulate the rest of the data.

As indicated in the introduction and Section 6.3, the possibility of introducing small amounts of poison data into the dataset is highly realistic. If the dataset is crawled and collected automatically, malicious instances just need to be placed where the crawler can find them [31, 119]. If it is collected manually, the ability to poison a dataset is proportional to an individual's contribution to the dataset. As detailed in Section Section 6.3, the Warfarin dataset is collected by 59 individuals; thus, an average contribution constitutes about 2% of the dataset. We show that this amount of poisoning is sufficient to effectively poison the dataset (c.f. Section 6.3, Section 6.6.4, and Section 6.7).

6.4.2 Related Poisoning Attacks in Regression

Jagielski et Al. [7] present both a white-box and a black-box attack on regression learning. In this section, we present these attacks and their limitations.

6.4.2.1 Related White-Box Attacks

Deriving from Xiao et Al. [29], Jagielski [7] present a white-box attack on linear regressors. The attacker's objective is formulated as a bi-level optimization problem:

$$D_p^* = \arg\max_{D_p} \mathcal{W}(D_{test}, \theta_p^*) \tag{6.1}$$

$$s.t. \quad \theta_p^* = \arg\min_{\theta \in \Theta} \mathcal{L}(D_{tr} \cup D_p, \theta) \tag{6.2}$$

Equation (6.2) represents the usual minimization of the model loss $\mathcal{L}$ during the fitting of a model on both the clean training dataset D_{tr} and the poisoned dataset D_p. This yields a trained model with weights θ_p^*, i.e. a poisoned model.

A short note on the notation used: The set of poison datasets to be optimized over is called D_p, while the optimal poison dataset as per Equation (7.1) is called D_p^*; the asterisk denoting optimality. Similarly, the set of model parameters to be optimized over are called $\theta \in \Theta$, while the best set of parameters θ_p^* results from training the model up to convergence on the train and poisoned data (c.f. Equation (7.2)). This step is called the 'inner optimization'.

Equation (6.1) refers to maximising the attacker's objective $\mathcal{W}$ with respect to some test set D_{test}, using the model's weights as determined by Equation (6.2). Solving the first equation depends on the solution of the second equation, which is why it is considered a bilevel optimization problem. This is a hard problem: The attacker has to determine how the points they introduce in the dataset will change the model weights during training. The authors [29, 7] solve this using the Karush-Kuhn-Tucker (KKT) conditions. These are

a set of conditions that are assumed to remain satisfied when a given poison sample X_c is introduced. The authors can then proceed to solve a linear system, and thus derive the gradients. While effective and even applicable to classification learning (c.f. Section 7.4.3), this approach has two drawbacks:

- **Feasibility**. This approach is not feasible for deep neural networks, since the time required for solving this linear system is in $O(p^3)$, where p is the number of parameters in the model, and DNNs easily have a few million parameters or more [120]. For a more detailed analysis, see Munoz-Gonzalez et Al. [30].
- **White-Box Threat Model.** Additionally, this is a white-box attack and thus requires access to the gradients of the model. Our threat model assumes a more realistic black-box scenario where the attacker has no knowledge of the model and cannot access the gradients.

6.4.2.2 Related Black-Box Attacks

Jagielski et Al. [7] also present a black-box attack called *StatP*. This attack samples ϵn points from a multivariate Gaussian distribution, where the corresponding mean μ and covariance matrix Σ are estimated as the mean and co-variance of the true dataset D_{tr}. Then, *StatP* rounds the feature variables to the corners, queries the model, and rounds the target variable to the opposite corner. The corners are defined as the minimum and maximum of the feasibility domain γ of each variable. Both features and target are scaled to $[0, 1]$, thus the feasibility domain is a high dimensional cube $[0, 1]^{d+1}$ where d is the number of features. In summary, this attack creates a few isolated clusters of adversarial data, where both features and target take only extreme values of either $\gamma_{min} = 0$ or $\gamma_{max} = 1$.

This attack, however, still requires access to the trained black-box model, which may be unrealistic in a real-world scenario. Additionally, we find that while this attack is successful on linear models, it is unsuccessful when applied to non-linear models. We show this empirically in Appendix A.3.1, but give a brief explanation here: Nonlinear learners (such as Neural Networks, Kernel SVR, and Kernel Regression) can accommodate both the poison points created by *StatP* and the true data simultaneously. This is because the thusly created poison data does not contradict the true data points, since true data points rarely have features in the corners of the feasibility domain. In summary, this attack works only on linear models. This insight will motivate our proposed *Flip* attack on nonlinear learners, which we present in the next section.

6.4.3 Flip: A Black-Box Attack on Nonlinear Regressors

Algorithm 1 presents our proposed black-box attack called *Flip*. This algorithm computes a set of adversarial poison points for any poisoning rate $0 < \epsilon < 1$. The attack is completely independent of the regressor model and only requires

1. a substitute dataset (X^{sub}, y^{sub}) from the same domain as the training dataset D_{tr},
2. and the feasibility domain of the target variables $[\gamma_{min}, \gamma_{max}]$.

Algorithm 1 Flip attack

Require:

Substitute data X^{sub}, y^{sub} of size m
Number of poison points $\lceil \epsilon n \rceil$ to compute
Feasibility domain $[\gamma_{min}, \gamma_{max}]$ of the target values
Function Flip
$\Delta \leftarrow \emptyset$
for $i \in [1, ..., m]$ **do**
    $\Delta \leftarrow \Delta \cup \max(y_i^{sub} - \gamma_{min}, \gamma_{max} - y_i^{sub})$
end for
$T_\epsilon \leftarrow t \in \mathbb{R}$ s.t. t is the $\lceil \epsilon n \rceil$-th highest value of Δ
$I_\epsilon \leftarrow \{i \in [1, ..., m]$ s.t. $d_i >= T_\epsilon$ where $d_i \in \Delta\}$
$X_p \leftarrow \emptyset, \quad y_p \leftarrow \emptyset$
for $i \in I_\epsilon$ **do**
    if $y_i > \frac{1}{2}(\gamma_{max} - \gamma_{min})$ **then**
        $y_{p,i} \leftarrow \gamma_{min}$
    else
        $y_{p,i} \leftarrow \gamma_{max}$
    end if
    $y_p \leftarrow y_p \cup y_{p,i}$
    $X_p \leftarrow X_p \cup X_i^{sub}$
end for
return X_p, y_p

The feasibility domain is necessary because we usually assume that only certain target variables are valid. Extreme values are bound to raise suspicion, such as a room temperature of -400 degrees Celsius, or medical dosages that are unreasonably high or low. Note that in contrast to the *StatP* algorithm, our proposed attack does not modify the data's feature values when creating the poisoning instances.

We now describe our attack (c.f. Algorithm 1). After having initialized an empty set Δ in line 5, we populate it in the following *for loop* (line 6-9). For each instance in the substitute dataset, we find the maximum of the distance to the lower or upper end of the feasibility domain and save the results to Δ. Then, in line 10 we find the $\lceil \epsilon n \rceil$-highest value $t \in \mathbb{R}$ in Δ. This is used in line 11 to compute the indices of those points for which there is most potential to disturb. Thus, the rationale of line 10-11 is to find those points for which the target value is closest to either γ_{min} or γ_{max}. These values are the ones that can be maximally disturbed by shifting the target variable to the other side of the feasibility domain. This is implemented in line 13-22, where we compute the poison set by retaining the feature values and 'flipping' the target value to the other side of the feasibility domain for the appropriate candidates as specified by I_ϵ. Finally, in line 23, we return the found poison data.

The intuition behind this algorithm is as follows:

- First, we observe that we do not want to change the feature values. If we change the feature values, a non-linear regressor can still accommodate both benign and poisoned data. We have observed this for the *StatP* attack from related work, c.f. section 6.4.2.2, which works well for linear models, but not for non-linear models.
- Second, we want to change the target labels, but only so much as to not arise suspicion (thus the feasibility domain). This is because higher (but still insusceptible) feature values will impact the learner more.

Before presenting an empirical evaluation concerning its effectiveness, we review related data poisoning defenses, evaluate shortcomings, and propose our new defense algorithm. Section 6.6 will then evaluate both our proposed attack and defense, where we show that the proposed *Flip* attack can reliably poison datasets, irrespective and without access to the defender's learning model.

6.5 Data Poisoning Defenses

In this section, we present defenses for adversarial data poisoning in regression. First, we list a set of requirements to make a defense applicable in the real world. Second, we evaluate existing defenses with respect to these requirements. Finally, we present our improvement over the baseline. A quantitative evaluation is given in Section 6.6, while a qualitative evaluation is presented in Section 6.7.

6.5.1 Requirements for Applicable Data Poisoning Defenses

When creating a dataset such as the Warfarin Dataset [112], the defender or creator does not know the degree of data poisoning ϵ. It is also entirely possible that no poisoning has occured, i.e. that $\epsilon = 0$. Thus, while ϵ may be known to the attacker, it is unknown to the defender. Thus, the quality of defense should not depend too much on a correct guess of ϵ, and should not deteriorate the quality of an unpoisoned dataset. With this requirement in mind, we proceed to present and evaluate existing defenses.

6.5.2 Related Defenses

Since there exists so little work on data poisoning in regression, existing defenses are also few. Two defenses from the domain of classification learning are presented by Steinhardt et Al [121]. The *Sphere* defense first computes centroids in the poisoned data, and then removes points outside a spherical radius around the centroids. The *Slab* defense 'projects points onto the line between the centroids and then discards points that are too far away' [121]. However, the authors themselves note that these defenses may leave datasets vulnerable, and present an example based on the International Movie Database (IMDB) classification dataset where both *Sphere* and *Slab* fail [121]. For this reason, we do not consider these defenses to be viable.

Algorithm 2 Trim defense, as proposed in [7]

Require:
Poisoned dataset $D_{tr} \cup D_p$ of combined size n
Some model loss $\mathcal{L}$
Estimated fraction of poison points $\hat{\epsilon}$
Function Trim
$\mathcal{I}^{(0)} \leftarrow$ a random subset of indices with size $n\frac{1}{1+\hat{\epsilon}}$
$\theta^{(0)} \leftarrow \arg\min_\theta \mathcal{L}(D^{\mathcal{I}^{(0)}}, \theta)$
$i \leftarrow 0$
while True **do**
    $i \leftarrow i + 1$
    $\mathcal{I}^{(i)} \leftarrow$ subset of size $n\frac{1}{1+\hat{\epsilon}}$ with minimal loss $\mathcal{L}(D^{\mathcal{I}^{(i)}}, \theta^{i-1})$
    $\theta^{(i)} \leftarrow \arg\min_\theta \mathcal{L}(D^{\mathcal{I}^{(i)}}, \theta)$
    break if some convergence condition is met
end while
return $D^{\mathcal{I}^{(i)}}$
End

A state-of-the-art defense is *Trim* [7], for which we give pseudo-code in Algorithm 2. It is an iterative algorithm, which first fits a regressor to a subset of the poisoned data, and then iteratively calculates the error between the regressor's prediction on the *train* set and the *train* targets. It refits the regressor on those points with the smallest error and repeats until a convergence criterion is met. Finally, it returns the points with the

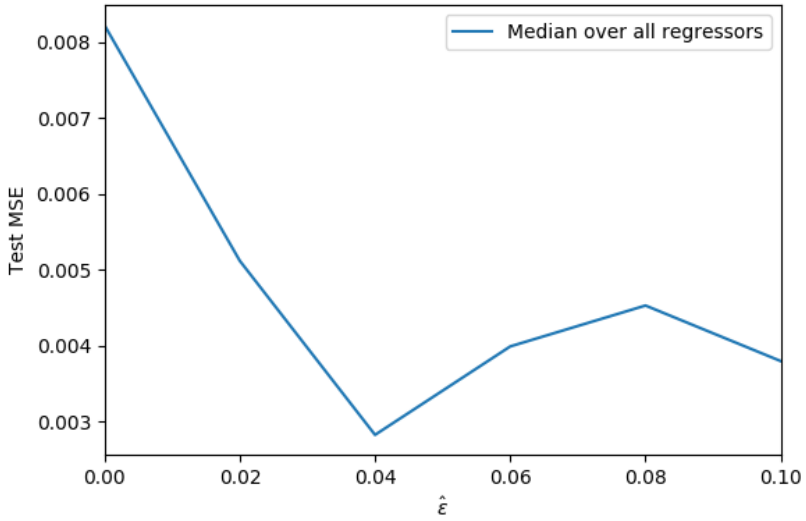


Figure 6.1: Three real-world datasets (*Warfarin*, *Loan* and *Housing* [7]) are poisoned with the *Flip* attack, where $\epsilon = 0.04$. We then apply the *Trim* defense with different values of $\hat{\epsilon}$, and train a regressor on the resulting dataset. Finally, we plot the test MSE against $\hat{\epsilon}$, averaged over all datasets and regressors. Observe that the *Trim* defense is highly dependent on the correct choice of $\hat{\epsilon}$ (in this case 0.04): If $\hat{\epsilon}$ is estimated too low, poison points remain in the training data, skewing the regressor and causing high test loss. If $\hat{\epsilon}$ is estimated too high, legitimate points are removed with the poisoned points. This loss in training data causes the regressor to learn a distribution different from the test distribution, which also incurs higher test loss. Thus, it is important to accurately estimate the degree of poisoning $\hat{\epsilon}$.

smallest error as a 'cleaned' dataset. The number of points to fit on, and conversely, the number of points to discard, is determined by a supplied parameter $\hat{\epsilon}$, the assumed degree of poisoning. If $\hat{\epsilon} = \epsilon$, the defense has been shown to work very well [7]. However, there is a caveat: In a realistic scenario, the defender does not know ϵ.

Consider Figure 6.1, where we poison three real-world datasets, including the *Warfarin* dataset, with a fraction of data poisoning $\epsilon = 0.04$. Then, for each dataset, we clean it using the *Trim* defense and supply $\hat{\epsilon} \in [0.00, 0.02, 0.04, 0.06, 0.08, 0.10]$ (i.e. we clean the poisoned dataset with different estimates $\hat{\epsilon}$ to quantify the effect of $\hat{\epsilon}$ on *Trim*). On the resulting data (which is partially or fully free from poison samples, depending on $\hat{\epsilon}$), we train a regressor and calculate the MSE on a separate test set. Then, we average the MSE over all datasets and plot the median of the regressors against $\hat{\epsilon}$.

We make one key observation: The effectiveness of *Trim* highly depends on the correct choice of $\hat{\epsilon}$. Selecting $\hat{\epsilon}$ below the actual degree of poisoning results in not all poison samples being removed and, thus, in an increase of the test error of a regressor. Selecting $\hat{\epsilon}$ above the actual degree of poisoning results in benign data being removed, which also removes relevant structure/information contained in the dataset and, as a result, also

increase test error. Therefore, a better selection strategy than blind overestimation of $\hat{\epsilon}$ is required.

In summary, the state-of-the-art poisoning defense *Trim* is not applicable in practice, since it requires the defender to precisely know the true poison rate ϵ.

6.5.3 The Iterative Trim Defense

Algorithm 3 iTrim defense

Require:
Poisoned dataset $D_{tr} \cup D_p$
Some model loss $\mathcal{L}$
Maximum estimated poisoning rate ϵ_{max}
Number of runs r
Threshold t
Function iTrim
$I \leftarrow \left\{ \epsilon_{max} \frac{j}{r-1} \text{ s.t. } j \in \{0, ..., r-1\} \right\}$
for $i \in I$ **do**
  $D^{(i)} \leftarrow trim(D_{tr} \cup D_p, \mathcal{L}, \hat{\epsilon} = i)$
  $L^{(i)} \leftarrow \min_\theta \mathcal{L}(D^{(i)}, \theta)$
end for
$\epsilon_{opt} \leftarrow \min\{i \in I \text{ s.t. } |L^{(i)} - L^{(i-1)}| < t\}$
return $trim(D_{tr} \cup D_p, \mathcal{L}, \hat{\epsilon} = \epsilon_{opt})$
End

As shown in the last subsection, the *Trim* defense has the potential to accurately remove poison samples from a given dataset, provided that $\hat{\epsilon}$ is chosen correctly, but over- or underestimating $\hat{\epsilon}$ significantly decreases test performance. This insight motivates our proposed *Iterative Trim* defense (*iTrim*), which enhances *Trim* by an iterative search for the best $\hat{\epsilon}$. In this section, we present this algorithm and show how to select suitable values for $\hat{\epsilon}$. In Section 6.6, we show empirically, on 26 datasets, that *iTrim* can be applied to poisoned data under realistic conditions, and reliably identifies and removes the poisoned data.

6.5.3.1 Algorithm Description

Algorithm 3 details the *iTrim* defense. It takes as arguments the poisoned dataset $D_{tr} \cup D_p$, a loss $\mathcal{L}$, and three scalar hyperparameters. The first, ϵ_{max}, is an estimate of the maximum possible poisoning rate. This hyperparameter can be chosen arbitrarily large without impacting the defense's result, but if chosen adequately will improve run time. The second hyperparameter specifies the number of runs r. This hyperparameter does not have too much influence on the algorithm's performance; it influences together with ϵ_{max} which values of $\hat{\epsilon}$ will be tried. The final hyperparameter, the threshold t, does

have an impact on the algorithm's performance, and we will discuss how to chose it later on.

iTrim starts by calculating a set I of possible candidates $\hat{\epsilon}$ (line 7). The hyperparameters ϵ_{max} and r define the right bound and the number of points, respectively. Then, for each candidate, calculate the cleaned dataset $D^{(i)}$ using Trim, train the regressor and obtain the corresponding train loss $L^{(i)}$ (lines 8 - 11). Finally, the optimal value for $\hat{\epsilon}$ is found when the error in train loss between two consecutive losses $L^{(i)} - L^{(i-1)}$ first undercuts some threshold t (line 12). Finally, the dataset is cleaned using *Trim* with this estimate, and the result is returned (line 13).

6.5.3.2 Poison Rate Selection

Before we give an intuition for our algorithm, we shortly address validation approaches to finding $\hat{\epsilon}$. As already mentioned, $\hat{\epsilon}$ is a hyperparameter of *Trim*. In machine learning, a common approach to finding hyperparameters are validation schemes such as cross-validation. But for this approach to work, we require a clean validation dataset. Since we only have a single dataset, we have to assume that any validation split will contain poisoned instances, rendering conventional validation approaches invalid for finding hyperparameters in this setting.

Thus, we now proceed to explain our iterative approach to finding $\hat{\epsilon}$: Consider Figure 6.2, where we apply *Trim* to the *Warfarin* dataset poisoned with $\epsilon = 0.04$. The orange dashed line shows the train loss for different candidate values $\hat{\epsilon} \in [0.00, 0.02, 0.04, 0.06, 0.08, 0.10]$. Note that for the correct estimation of poisoning degree $\hat{\epsilon} = 0.04$, the train loss becomes almost zero, decreasing several orders of magnitude compared to $\hat{\epsilon} = 0.00$. Further increasing $\hat{\epsilon}$ still decreases the train MSE, but only insignificantly. Thus, the training loss can be approximated by two straight lines, joined at a distinctive kink where $\hat{\epsilon} = \epsilon$. Figure A.1 in the Supplementary Material shows this for other real-world datasets. We can understand this kink as the point where the dataset ceases to contain data that incur extremely high train loss - in other words, where all adversarial poison data have been removed. This assumption is supported by the blue line in Figure 6.2, which shows the test MSE for the same *Kernel Ridge* regressor trained on the thusly cleaned datasets. For $\hat{\epsilon} = 0.04$, the test loss is minimal. For $\hat{\epsilon} > 0.04$, *Trim* starts to remove legitimate data (since all poison data have been removed), which is why test performance deteriorates. Section 6.6 will verify this empirically.

Based on the insight that the training loss can be approximated by two straight lines that intersect at ϵ, we develop our selection criterion for $\hat{\epsilon}$. We define t as the maximum absolute gradient of the straight line where $\hat{\epsilon} > \epsilon$ (i.e. the slope of the orange dashed line on the 'right' side of the graph, where all poison data have been removed). We will refer to this straight as the *normal straight*. Then $|L^{(i)} - L^{(i-1)}|$ is used to approximate the gradient of the straight for each subinterval. The division by the length of the interval is omitted since all intervals are equidistant. We choose $\hat{\epsilon}$ as the first candidate so the estimated straight is normal (i.e. $|L^{(i)} - L^{(i-1)}| < t$).

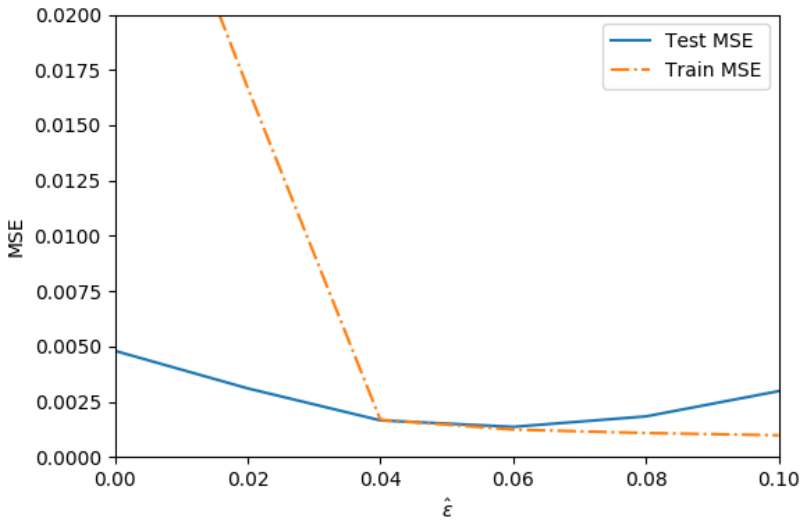


Figure 6.2: Applying *Trim* to the *Warfarin* dataset, poisoned with $\epsilon = 0.04$. The orange dashed line shows the averaged train loss for different estimations of poisoning $\hat{\epsilon}$, using *KernelRidge* regression. There is high train loss for $\hat{\epsilon} = 0$ and 0.02, because in these cases, not all poison points in the dataset can be removed. Once all adversarial poison data is removed ($\hat{\epsilon} = 0.04$, the delta in train loss decreases by several orders of magnitude, approaching zero. The line in blue shows the test error for the same setup. Note that the best $\hat{\epsilon}$ is indeed characterized by a sudden change of train loss, as indicated in the left figure. Also note that for $\hat{\epsilon} > 0.04$, *Trim* starts to remove legitimate data, which deteriorates test performance.

6.5.3.3 Threshold Selection

iTrim is dependent on an appropriate choice of the threshold t. If t is vastly too large, poison points are left in the dataset. If t is too small, *iTrim* deteriorates to *Trim*, and starts removing non-poison points. However, we find that there is rather a large window of appropriate values of t. This is because 1) we apply feature/target scaling to $[0, 1]$, and 2) the difference in train loss we observe once all poisoned points are removed is dramatic (c.f. Figure 6.2). Based on our evaluation on 26 datasets (c.f. Section 6.6.2), we find empirically that choosing values between 0.05 and 0.0001 performs comparably, and thus decide on a threshold $t = 0.001$. This threshold is then used on all datasets, which mimics real-world circumstances where fine-tuning to individual datasets is not feasible (since this would require evaluation data).

In summary, we find that:

- *Trim* lacks a mechanism to find a good estimate for the percentage $\hat{\epsilon}$ of poisoned points in the dataset.
- Over- and underestimation of $\hat{\epsilon}$ deteriorate the dataset to be cleaned.
- Appropriate values for $\hat{\epsilon}$ can be found via *iTrim*.

6.6 Empirical Evaluation

In this section, we evaluate our attack and defense algorithms against 26 datasets. We show that we 1) can reliably poison nonlinear and linear models while assuming a realistic black-box threat model, and 2) defend against this attack better than previously suggested defenses.

6.6.1 Experimental Setup

We try to make our experiment as general and realistic as possible. First, we split each of the 26 datasets into a randomly drawn *substitute set* of size 0.25, a *train set* of size $0.75 * 0.8$ and a *test set* of size $0.75 * 0.2$. For each combination of the 26 *substitute* datasets and $\epsilon \in [0.00, 0.02, 0.04, 0.06, 0.08, 0.10]$, we create a poisoned dataset using the respective attack, which we append to the corresponding *train set* and shuffle. This results in $6 * 26 = 156$ combinations of *train* dataset and poisoning rate. This step does not depend on the regressors. Then, for each regressor and each of the 156 *poisoned train datasets*, we perform a cross-validated grid search to find suitable hyperparameters. Finally, for all 156 *poisoned train datasets* and both defenses (*Trim* and *iTrim*), we apply the defense to each of the 156 *poisoned train datasets*. We then train a regressor and measure test error on the corresponding *test data* sets and report below. Thus, in total, we run $156 * 7 * 2 = 2184$ experiments (7 being the number of different regressors evaluated). Each experiment mirrors a data set being poisoned and then cleaned with either of the defenses. The experiments and source code are published to enable reproducibility, c.f. `https://github.com/Fraunhofer-AISEC/regression_data_poisoning`.

6.6.2 Datasets and Regressors

For our experiments, we use 26 datasets: Three datasets introduced in [7], eight datasets from the GitHub repository *imbalanced dataset* [122], and 15 datasets from the *KEEL* regression repository [123]. Each dataset contains at least 1000 data points. For datasets where $n > 10000$, we randomly sample a subset $n = 10000$. In keeping with [7], we scale features and targets to $[0, 1]$. See Appendix A.3.1 for a detailed summary.

We evaluate four linear models (HuberRegressor, Lasso, Ridge, Elastic Net) and three non-linear models (Neural Networks, Kernel Ridge with RBF kernel, and Support Vector Regressor with RBF kernel). To the best of our knowledge, we are the first to evaluate poisoning attacks against non-linear regressors.

6.6.3 Evaluation of *StatP*

In this section, we very briefly report the effectiveness of *StatP* on non-linear regressors. As detailed in [7], the attack is effective for linear regressors. We find, however, that it is not effective when applied to non-linear learners. For example, a Neural Network's MSE remains nearly unchanged (from 0.051 to 0.055) when poisoned with ten percent of poison samples created by *StatP*. In Appendix A.3, we elaborate this in more detail and evaluate additional non-linear learners such as Kernel SVM and Kernel Ridge, which we find to behave similarly.

6.6.4 Evaluation of *Flip*

In this section, we present the results when evaluating our proposed *Flip* attack against 26 datasets and seven regressors. Figure 6.3 shows the performance of the *Flip* attack, averaged over all datasets. Figure A.3 in the Supplementary Material shows results per dataset.

We observe that the attack is highly effective: When adding only 4% of poison data, the MSE of most regressors doubles compared to the non-poisoned case. We observe that all models seem equally susceptible to our attack, except for the Huber Regressor and Support Vector Regressor, which are designed to be outlier-resistant.

We shortly discuss why these models are more resilient:

- **Huber Regressor**. The Huber Regressor is design to be outlier resistant. This is achieved by using a specialized *Huber Loss* function [124], which is a mixture between L_1 and L_2 loss:

$$L_\delta(x) = \begin{cases} \frac{1}{2}x^2 & \text{for } |x| \leq \delta, \\ \delta\left(|x| - \frac{1}{2}\delta\right) & \text{otherwise.} \end{cases}$$

  Thus, the impact of outliers (defined as those points with large magnitude) on the loss function is limited by replacing the L_2 loss and its quadratic penalty with the linear L_1 loss. However, thus Huber loss has a major downside: It requires an additional hyper-parameter δ, which defines the magnitude above which feature

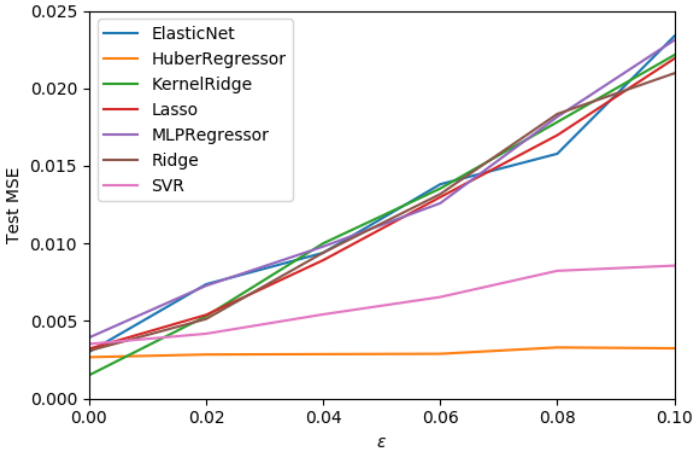


Figure 6.3: Evaluation of our proposed *Flip* attack. This plot shows the MSE for different poison rates per regressor, averaged over all 26 datasets. Most regressors obtain an MSE of around 0.003 when $\epsilon = 0$, and all but two deteriorate linearly as ϵ increases. Figure A.3 in the Supplementary Material shows the same results, but for each dataset individually. Note the effectiveness of the attack: For only four percent of poison data, the test error increases to well over 200% for most models.

values are considered as outlier. This, obviously, is data-dependant, and one either has to guess δ or find it via expensive hyper-parameter optimization such as grid-search. In this vase, however, it is not clear which evaluation metric to use, since the poisoned data would also be present in the test data. Thus, while using the Huber loss can work well, its dependence on the parameter δ makes real-world application difficult.

- **Support Vector Regressor**. This model proves to be more robust to the poisoned data than other models, such as the neural network (i.e. *MLPRegressor*). However, it still is affected: Test MSE increases by about 250% when 10% poisoned data is introduced.

In summary, while some models are more robust to poisoned data than others, the benefit of employing these has to be carefully weighted against potential downsides. The Huber Loss can also be used in the context of neural networks, and can prove a suitable alternative to the MSE-Loss, provided one has a way to estimate δ. For problems that are not too complex and where the full flexibility of neural networks may not be needed, the *rbf*-kernel Support Vector Regressor may be a useful algorithm.

6.6.5 Evaluation of *Trim* and *iTrim*

In this section, we report the results when defending against the *Flip* attack. We report both the performance of the *Trim* defense and our proposed *iTrim* defense and compare efficiency. In order to mimic the behaviour of a defender in a realistic scenario, we

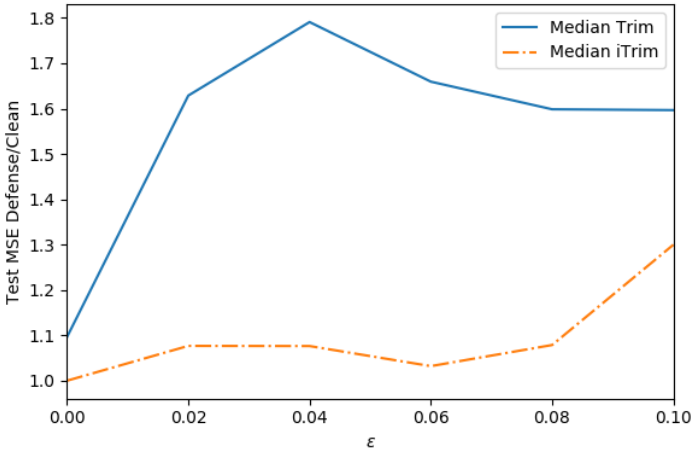


Figure 6.4: Evaluation of the *Trim* and *iTrim* defense (lower error is better). First, we poison the 26 datasets using *Flip*. Then we apply the *Trim* or *iTrim* defense and calculate the test MSE. Finally, we normalize by the *baseline* MSE - the error obtained by a model trained on unpoisoned data. The resulting quotient represents the degree to which the defenses can negate the *Flip* attack. We observe that *iTrim* consistently outperforms *Trim* by a large margin. More detailed, non-averaged results are presented in Figure A.4 and Figure A.5 in the Supplementary Material.

set both $\hat{\epsilon} = 0.14$ (for *Trim*) and $\epsilon_{max} = 0.14$ (for *iTrim*). A defender would have to guess the percentage of poisoned data ϵ, with a preference for overestimation rather than underestimation (as explained in Section 6.5.3).

We proceed as follows: With varying degrees of poisoning, we poison all 26 datasets using the *Flip* attack. Then, for each regressor, we clean (i.e. 'defend') the datasets using the *Trim* as well as the *iTrim* defense (separately). We fit the regressor on the thusly obtained dataset and compare the test error against a regressor trained on the clean data. Figure 6.4 shows the median of all regressors for *Trim* (blue line) and *iTrim* (orange dashed line). The Supplementary Material provides more details: Figure A.4 presents the results for each regressor individually, while Figure A.5 depicts the results for all 26 datasets.

We observe that both defenses are effective. However, *iTrim* achieves higher performance than *Trim*, especially when there is a large discrepancy between ϵ and $\hat{\epsilon}$. This is due to *iTrim*'s capability of more accurately estimating the degree of poisoning. Especially for $\hat{\epsilon} > \epsilon$, we see considerable improvement due to *iTrim*s more advanced estimate of ϵ.

6.6.6 Runtime

In this section, we detail the runtime of the *iTrim* defense algorithm. *iTrim* calls the *Trim* defense r times, which in turn performs j fit operations of the regressor - until either a convergence criterion is met, or the number of runs j is exhausted. In our experiments, we choose $r = 6, j = 20$. Running the complete experiment (attacking all 26 datasets,

Table 6.2: Mean absolute error (MAE) of different regression models when poisoned using the *Flip* attack.

Model	MAE C	MAE P	MAE D	MAE P/C	MAE D/C	Accbl. P/C	Accbl. D/C
Elastic Net	8.52	11.07	8.51	1.30	1.00	21.25	-0.82
Huber Reg.	8.40	8.46	8.41	1.01	1.00	-1.09	-0.82
Kernel Ridge	8.41	10.98	8.41	1.31	1.00	21.07	0.53
Lasso	8.49	11.20	8.49	1.32	1.00	22.31	0.00
MLP	10.05	12.33	9.86	1.23	0.98	11.41	-2.80
Ridge Reg.	8.49	10.99	8.51	1.29	1.00	22.34	1.06
SVR	10.85	12.52	11.19	1.15	1.03	14.13	2.90
Median	8.49	11.07	8.51	1.29	1.00	21.07	0.00

for seven regressors, and six poisoning rates ϵ) results in $26 * 6 * 7 = 1092$ calls to the *iTrim* defense. On a Intel(R) Xeon(R) CPU E7-4860 v2 @ 2.60GHz with 96 cores, this takes about 120 minutes when parallelizing into 15 separate processes. Thus, running a single *iTrim* defense takes, on average, $120/1092 * 15 = 1.6$ minutes per 6-core process. Obviously, this is highly dependent on the regressor's complexity, the size of the dataset, the number of features, and the parallelism capabilities of the program code. Still, this indicates the feasibility of applying the *iTrim* defense in a real-world scenario, where after weeks, months, or even years of data gathering, running the *iTrim* incurs negligible additional time overhead.

6.7 Warfarin Revisited

In Section 6.3, we presented the medical use case of predicting the therapeutic Warfarin dosage and showed that data poisoning can significantly impact the performance of regressors applied to this problem. In this section, we will illustrate the empirical results of Section 6.6.5 on the use case of Warfarin dosage prediction, and show that by using our defense, we can effectively defend the dataset against data poisoning in this medical use case.

We structure the different stages of data poisoning into three different scenarios: First, the (C)lean case. In this scenario, no data poisoning occurs. This case will be used as a baseline for measuring the effects of data poisoning and defense. Second, the (P)oison case. In this scenario, the attacker introduces 2% poison samples using the *Flip* attack proposed in Section 6.4.3. No countermeasures are taken. Third, the (D)efended case. In this scenario, the data are poisoned with 2% poison samples like in (P), but *iTrim* is used as a countermeasure. The results for these three scenarios are summarized in Table 6.2.

To recapitulate: Warfarin is a blood thinner with a narrow therapeutic window resulting in high medical significance for the correct prediction of the therapeutic Warfarin dosage. The scenarios (C) and (P) have already been presented in Section 6.3. On clean data, the models used in our evaluation perform comparable to state-of-the-art models, and 2% poison samples are sufficient to considerably increase MAE and decrease the number of patients receiving an acceptable dosage of Warfarin by up to 22%.

In Table 6.2, the column 'MAE D/C' provides the factor by which the MAE of a regressor increases when the dataset is poisoned with 2% poison samples and then defended using *iTrim*. As we can see, the median is 1.00, indicating that the damage is completely mitigated. The individual values range from 0.98 to 1.03, which indicates that where previously *Flip* incurred an increase in MAE of up to 31%, the *iTrim* defense reduces this error increase to less than a tenth. In summary, the MAE of the tested models in the (D) scenario is approximately the same as in the (C) scenario, meaning the defense successfully eliminates (most) of the negative impact of the poison samples.

The column *Acceptable D/C* gives the percentage by which the number of patients receiving an acceptable Warfarin dosage decreases in the (D) scenario compared to the (C) scenario. The median reduces from 21.07 in scenario (P) to a median of close to 0 in scenario (D). This shows that the number of patients receiving an unacceptable Warfarin dosage due to data poisoning is significantly reduced when the *iTrim* defense is employed. In summary, we observe that the *iTrim* defense decreases the influence of the poison samples significantly, almost restoring the dataset to unpoisoned quality. Thus, it results in more patients receiving adequate predictions for their therapeutic Warfarin dosage.

6.8 Conclusion

In this chapter, we introduced a novel data poisoning attack on regression learning as well as a corresponding defense mechanism. We show the effectiveness of our proposed attack and defense algorithm in a large empirical evaluation over seven regressors and 26 datasets. Both attack and defense assume realistic constraints: The attack is black-box and doesn't assume access to the true dataset, but only a substitute dataset. The defense, on the other hand, does not assume any knowledge of the poisoning rate ϵ, but estimates it using an iterative approach. The content of this chapter has been published and peer-reviewed as a scientific paper at 2020 IEEE 25th Pacific Rim International Symposium on Dependable Computing [4]. Passages in this chapter have been quoted verbatim from this work.

7 Identifying Adversarially Poisoned Data in Classification Learning

The previous chapter presented a defence against adversarial data poisoning in regression data. This chapter examines a related problem: Identifying adversarial data poisoning in classification learning. This chapter is based on research first presented in the conference paper 'Defending against adversarial denial-of-service data poisoning attack' [5], published at the DYNAMICS workshop of the Annual Computer Security Applications Conference 2020 (ACSAC'20) conference. Thus, passages are cited verbatim from this publication.

Also, please note that while the threat model and parts of the defender's model architecture are largely identical between a regression and classification data poisoning scenario, the algorithms to create and find adversarial instances differ significantly. This difference is a key factor in motivating this research, instead of simply applying the results of the last chapter to the domain of classification. Section 7.8 elaborates these differences further.

7.1 Motivation

The significance of machine learning for the prediction of categorical targets (i.e. classification learning) is unwavering: In recent years, it has led to a continuous stream of breakthroughs in countless fields, for example, image classification [72, 125] or speech recognition [126, 127]. Important classification problems also include security-sensitive applications such as face recognition [128], fingerprint identification [129], autonomous driving [130] and the detection of spam [131], malware [132] and network intrusion [133].

Similar to the scenario of regression learning, data poisoning attacks are a serious threat to these systems. For example, Nelson et al. [131] use DoS data poisoning to attack a spam filter. By sending attack emails that contain many words likely to occur in legitimate emails, they increase the likelihood of future benign emails to be marked as spam by the learning algorithm. As spam filters rely on very low false-positive rates, they are effectively shut down by this attack. A similar attack is presented by Biggio et Al. [134], who poison a machine-learning based malware clustering system. Further poisoning attacks are detailed in [135, 28, 136, 30, 29].

To mitigate the threat of DoS poisoning, previous work has developed defenses based on measuring the influence of samples on a model [136, 131] or clustering and anomaly detection [137, 138]. However, most defenses are computationally inefficient or susceptible to the curse of dimensionality [139]. Also, anomaly detection based approaches often suffer from the arbitrary selection of detection thresholds.

7.2 Contribution

This chapter contributes by presenting a new defense against DoS data poisoning attacks in the domain of classification. It improves upon related work as follows:

- **Robustness**: The proposed defense is not based on anomaly detection or metric learning. Thus, it is not susceptible to the curse of dimensionality [140], as opposed to [136, 131, 137, 138]. This allows for application on very high-dimensional data.
- **Broad and realistic applicability**: Our proposed defense does not come with restrictions (such as exclusive applicability on linear or binary models). Additionally, it does not depend on the defender knowing the true fraction of outliers (as opposed to [141]); but even provides an estimate of the poisoning rate.
- **Performance**: We outperform state-of-the-art: False positive / false negative rates are improved by at least 50%, often more (c.f. Section 7.6).

7.3 Related Defense Algorithms

To mitigate the threat of DoS poisoning attacks against spam filters, Nelson et al. [131] propose to measure the impact of each training sample on the classifier's performance. This defense can effectively determine samples that decrease the accuracy and should be discarded. However, it is computationally expensive and can suffer from overfitting when the dataset is small compared to the number of features. A similar approach based on influence functions is presented in [136].

The authors of [141] propose a two-step defense strategy for logistic regression and classification. After applying outlier detection, the algorithm is trained in solving an optimization problem based on the sorted correlation between the classifier and the remained samples. However, this approach is not feasible in practice as the fraction of poison samples is required for the algorithm to perform well.

There are also different approaches to outlier detection. The authors of [138] use the k-nearest neighbors algorithm to find instances whose labels differ from that of their neighbors. These instances are considered to be adversarial. In contrast, the authors of [137] first split the instances in the poisoned dataset into their respective classes and then use different metrics for outlier detection to identify poison instances with high outlierness scores. Note that this defense requires a training set of *trusted* data. Both approaches highly depend on the selection of appropriate thresholds and suffer from the curse of dimensionality.

7.4 Poisoning Attack

In this section, we present two DoS poisoning attacks from related work. We implement them in order to obtain a strong attack against which to evaluate our proposed defense.

7.4.1 Threat Model

Our thread model is the same as the one presented in section 6.4.1. To reiterate: We assume an attacker who can introduce some percentage of poisoned data into the dataset. We evaluate our proposed defense against both a black-box attack (label flipping, c.f. Section 7.4.2) and a stronger white-box attack (back-gradient optimization, c.f. Section 7.4.3).

7.4.2 DoS Poisoning via Label Flipping Attack

As a baseline attack, we consider the *Random Label Flipping* attack, where the attacker randomly selects a percentage ϵ of instances from the training dataset and changes their labels to a class different from the original one [138]. This simple concept of label flipping describes a black-box attack, as it only requires the attacker to have access to the labels of the training dataset D_{tr}.

7.4.3 DoS Poisoning via Back-gradient Optimization Attack

In this section, we briefly describe the current state-of-the-art attack for DoS data poisoning by Munoz et Al. [30], which is derived from Xiao et Al. [29]. Conceptually, it is very similar to the white-box attack on regression learning examined in Section 6.4.2.1. It is a white-box attack that iteratively introduces poison samples into the training dataset. An attacker objective function $\mathcal{W}$ maps a model with poisoned parameters θ_p^* and a test set D_{test} to a real number, i.e. $\mathcal{W}(D_{test}, \theta_p^*) \in \mathbb{R}$. The attacker aims at finding a poisoned dataset D_p^* such that their objective $\mathcal{W}$ is maximized on D_{test} for a model trained on $D_p^* \cup D_{tr}$. Formally:

$$D_p^* = \arg\max_{D_p} \mathcal{W}(D_{test}, \theta_p^*) \tag{7.1}$$

$$s.t. \quad \theta_p^* = \arg\min_{\theta \in \Theta} \mathcal{L}(D_{tr} \cup D_p, \theta) \tag{7.2}$$

Equation (7.1) is called 'outer problem', since it depends on the model parameters θ_p^*, which are found during model training ('inner problem', Equation (7.2)). Thus, finding a set of poison data D_p^* which maximizes the attacker reward is a bi-level optimization problem. Note that for DoS poisoning attacks, maximizing the attacker's objective function $\mathcal{W}$ is equivalent to maximizing the model loss $\mathcal{L}$.

Since the bi-level optimization problem is intractable, it is not solved directly, but an iterative approach is employed: The attacker initializes an empty set $D_p^* = \varnothing$ and adds the poison samples one by one, each of which is created as follows. First, the poison sample is initialized either by sampling a random instance from a train or substitute set. Then, the label of the instance is flipped, i.e. changed to a different class. Finally, the attack iteratively applies poisoning steps which modify the features of the poison sample in order to increase its impact. Instead of solving Equation (7.1), which is intractable, each step applies back-gradient optimization to derive $\nabla_{x_p} \mathcal{W}$. This gradient describes

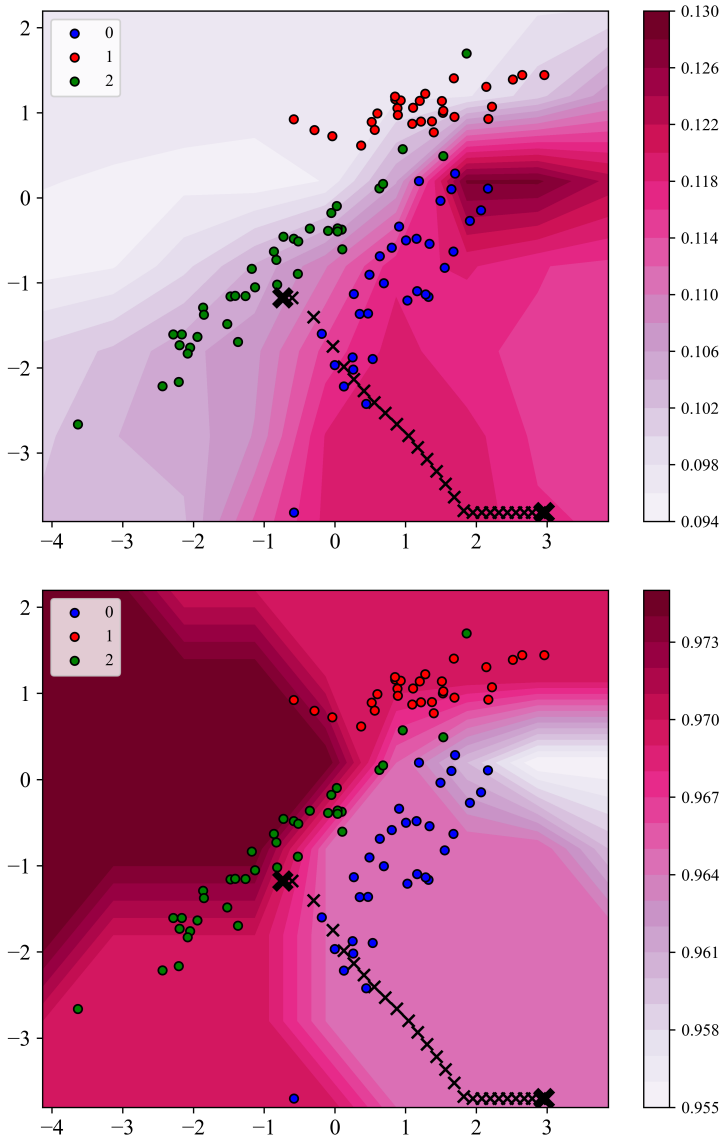


Figure 7.1: *Top image:* The trajectory of a malicious sample during a poisoning attack against the two-dimensional *points* dataset. After initially switching the label from 2 to 1, the instance is moved from $x_p = [-0.74, -1.18]$ (region of smaller validation loss) to $x_p = [2.96, -3.7]$ (region of higher validation loss) by iteratively applying back-gradients. *Bottom image:* The same plot is shown for a validation accuracy colormap.

how the attacker's objective function changes w.r.t. a single poison instance x_p's feature values, which can then be modified accordingly. This gradient is computed as follows:

$$\nabla_{x_p} \mathcal{W} = \nabla_{x_p} \mathcal{L} + \frac{\partial \theta}{\partial x_p} \nabla_{\theta} \mathcal{L} \tag{7.3}$$

It is difficult to compute $\frac{\partial \theta}{\partial x_p}$, which indicates how the model changes with respect to the poison sample. Early poisoning attacks have replaced the inner problem with its stationary KKT conditions, but this approach can be prohibitively expensive in time and memory, especially for larger neural networks (the attack has cubic complexity in the number of model parameters). The approach presented in [30] replaces the inner optimization with a sequence of learning steps to smoothly update the parameters θ. After an incomplete optimization of the inner problem within T iterations, the parameters θ_T can be used to compute the desired gradients of the outer problem. While current state-of-the-art approaches based on KKT conditions require solving one linear system per parameter and computationally demanding matrix operations that scale in time as $O(|\theta|^3)$ and in memory as $O(|\theta|^2)$, the time complexity of back-gradient descent is in $O(T)$. This significantly reduces the complexity of computing the outer gradient and enables to poison large neural networks and deep learning algorithms. To avoid storing the whole training trajectory $\theta_1, ..., \theta_T$ along with the forward derivatives for back-propagation, Munoz et Al. [30] propose to compute these parameters during the backward pass based on θ_T by reversing the steps of the learning algorithm. Pseudo-code algorithms in [30] summarize how to derive $\nabla_{x_p} \mathcal{L}$ by reversing a gradient-descent procedure with a fixed number of epochs and a fixed step size. We refer to this attack as a back-gradient optimization attack. We implement it both in Python and Matlab and use it to evaluate our defense against a strong, white-box adversary.

Intuition. We give a visual example of this attack in Figure 7.1. This figure shows the *points* dataset (c.f. Section 7.6.1) with three classes (shown as blue, red, and green dots) over a two-dimensional feature space. The black crosses represent a single poisoned instance as it evolves during $T = 26$ iterations of the attack. This poison sample is created by taking a legitimate data instance and flipping its label to a poison label, in this case from 2 to 1. Then, over the course of 26 iterations, the attack gradually shifts the poisoned instance to a region where it will induce higher validation loss and lower validation accuracy. The effects of training a model on the normal data D_{tr} plus a single poisoned instance x_p are shown by the background color map, whose x_1, x_2 coordinates represent the feature values of the poisoned instance x_p. To create the background plot, we partition the feature space into a grid, train a model for each poison instance on the grid, and plot validation loss and accuracy. This gives us the 'ground truth' with respect to how the poison sample should be changed in order to be maximally effective. Observe how the attack modifies the point such that it is moved to regions of higher validation loss and lower validation accuracy - i.e. to regions that are associated with bad model performance.

7.5 Poisoning Defence

While the previous section detailed strong baseline attacks from related work, this section presents our contribution: A new defense to detect DoS data poisoning, which we later evaluate against the previously presented attacks, and which improves over previously presented defenses.

7.5.1 Requirements

Our defense is designed to identify poison samples in a dataset without knowledge of the poisoning rate ϵ. It requires only the model and the poisoned dataset, both of which are available to the defender. This is the most unrestricted input conceivable and shows the applicability of our defense in any real-world setting.

7.5.2 Our proposed algorithm

We present an algorithm that identifies poison instances and removes them from the dataset, while legitimate data remains. This approach is inspired by our previous work [3], where we detect mislabeled instances in classification datasets.

We define the following notation: Let D be a dataset $D = (x, y)$, consisting of N instances out of C target classes. Put differently, input x represents N data instances while target y represents N data labels as one hot-encoded vectors of length C. Both x and y can be conveniently expressed in matrix notation. Finally, let $D_p^* \subseteq D$ for $D = (x, y)$ be the set of poison instances, which is unknown to the defender.

Informal description. To identify these poison samples, we train a classifier M on the given dataset D and use the same classifier to obtain new class probabilities for x. We then discard instances for which the class probability is small, which means that the classifier considers the original label extremely unlikely based on the feature distribution learned during training. This works because there is still a significant majority of unpoisoned data in the training dataset. This is a reasonable assumption to make for crowd-sourced datasets, where a malicious individual can only introduce a small portion of malicious data. Nevertheless, we show in our experiments that our system works with up to 10% of poison data in the training dataset.

Formal description. Formally, our proposed defence corresponds to an indicator function $\iota \rightarrow \{0, 1\}$ where $\iota(d) = 1$ indicates that $d \in D$ is malicious. We aim at maximizing the intersection between the true poisoned samples D_p^* and $\{d \in D | \iota(d) = 1\}$ while minimizing the intersection of D_p^* and $\{d \in D | \iota(d) = 0\}$. Our proposed algorithm is detailed in algorithm 4. Additionally, we provide a textual description in the following paragraph.

1. Prerequisites: Given a dataset $D = (x, y)$, which may contain malicious poison instances, i.e. $D = D_{tr} \cup D_p^*$. We allow D_p^* to be empty (no attack is present).
2. Train model M on the dataset D.

Algorithm 4 Our proposed defense

Require:

Dataset $D_{orig} = D_{tr} \cup D_p^*$, where D_{tr} is the training data and D_p^* the poisoned data
A defender training loss L
An upper window size s to approximate the discrete derivative of l_n
Function
$D \longleftarrow D_{orig}$
while True **do**
  $\theta^* \longleftarrow \arg\min_\theta \mathbb{E}_{(x,y)\sim D} L(f_\theta(x), y)$
  $\Delta \longleftarrow \varnothing$
  for $i \in [1, ..., |D_{orig}|]$ **do**
    $y_i^{pred} \longleftarrow f_{\theta^*}(x_i)$ where $x_i \in D_{orig}$
    $l_i \longleftarrow \langle y_i, y_i^{pred} \rangle$
    $\Delta \longleftarrow \Delta \cup \{l_i\}$
  end for
  $\Delta' \longleftarrow sort\_ascending(\Delta)$
  $k = \arg\max_i \sum_{i=1}^{|D|-s} \frac{1}{s} \sum_{k=1}^{s} |l_{i+k/2} - l_{i-k/2}|$ where $l_i \in \Delta'$
  $D' \longleftarrow \{x_i \in \Delta' | i \geq k\}$
  if $D' = D$ **then**
    break
  end if
  $D \longleftarrow D'$
end while
return D

3. Reclassify training input x using the trained model M and obtain the new class probabilities (or logits) $M(x) = y'$.
4. Calculate for all $n \in \{0, ..., N-1\}$ the label likelihood l_n that instance x_n is assigned the original label y_n. Because y_n is a one-hot vector, i.e. $y_n = [0, 0, \ldots, 1, \ldots, 0]^T$, this computation is trivial:

$$\begin{aligned} l_n &:= \langle y_n, {y'}_n \rangle \\ &= 0{y'}_n^0 + 0{y'}_n^1 + \cdots + 1{y'}_n^c + \cdots + 0{y'}_n^{N-1} \\ &= {y'}_n^c. \end{aligned}$$

   Here, c is the non-zero valued index in the one-hot vector y_n and ${y'}_n^c$ indicates the c-th vector component of ${y'}_n$.
5. Sort all training instances according to l_n in ascending order. This yields a graph as presented in Figure 7.3, where instances with small likelihood w.r.t their original label are on the left side of the plot.
6. Determine a threshold $k \in \{0, ..., N-1\}$ to distinguish between wrongly labeled poison data and correctly labeled benign data. We find this threshold by approximating the first derivative of l_n over a sliding window, and simply choosing k as the argmax. We motivate this in Section 7.6.3.
7. Remove instances D'_p where $l_i \leq l_k$ (i.e. left of the cutoff point) from the training set. Retrain the model on $D - D'_p$.
8. Iteratively apply steps 2 to 7 until convergence.

Intuition. The intuition behind this algorithm is as follows: By computing the likelihood l_n for each instance, we estimate the probability that the target label is correct. This is because a high label likelihood l_n indicates that the target label comes from the same distribution as the majority of the dataset $D = D^*_p \cup D_{tr}$. Note that we assume that the majority of the data points as being benign. This is a very reasonable assumption, which is usually made in literature. To reiterate: One generally assumes that $D_{tr} \gg D^*_p$ because the attack can introduce only a small amount of data (c.f. Section 6.3). A high value of l_n thus correlates with the sample in question being unpoisoned or benign.

Extracting information about the unpoisoned data distribution from a poisoned dataset requires a model M which can generalize well even in the presence of adversarial data. Such a model may not be optimal in an unpoisoned scenario, as it may discard a more complicated structure in the benign data. However, it is suitable in a poisoned scenario, because we want to discard everything except the most rudimentary structure in the data. The goal is to learn from the benign data what the poisoned instances' labels should be.

We then sort the instances by l_n and remove a portion that has very small l_n (and thus likely has a wrong label, which indicates adversarial data). We retrain our model on the remaining data and iteratively re-evaluate the whole dataset. In our empirical evaluation in Section 7.6, we find that this approach reliably identifies poisoned instances with low false positive and false negative rates.

Name	Size	Classes	$\lvert D_{tr}\rvert$	$\lvert D_{val}\rvert$	$\lvert D_{test}\rvert$
breast_cancer	(540, 30)	2	100	220	220
fashion_mnist_156	(6300, 784)	3	300	3000	3000
fashion_mnist_17	(2100, 784)	2	100	1000	1000
mnist_156	(6300, 784)	3	300	3000	3000
mnist_17	(2100, 784)	2	100	1000	1000
points	(300, 2)	3	100	100	100
spambase	(4200, 57)	2	200	2000	2000

Table 7.1: Datasets used when evaluating DoS data poisoning in the context of classification

Concerning the model M, we note that it must be able to generalize well on the dataset. It crucial that the model does not overfit, because if it does, it will not learn to approximate the underlying distribution of the samples D_{tr}, but simply remember all instances of D, including the poisoned instances in D_p^*. Such a model yields high label likelihood l_n for all instances, which renders the defense strategy invalid. Neural networks can satisfy these requirements. To prevent overfitting, we use an aggressive learning rate and early stopping.

Conceptual similarity to *LabelFix* algorithm. Note the conceptual similarity of our approach to the *LabelFix* algorithm in chapter 5. In both cases, we identify mislabeled or poisoned samples by training a strongly generalizing model on the noisy or poisoned data set. The model is able to ignore outliers, focusing on the majority of the clean data. Using the label likelihood l_n, we can identify the likelihood of data given the model, i.e. find data that does not conform to the model, and thus does not originate from the majority, i.e. the benign class.

7.6 Evaluation

In this section, we evaluate our proposed defense against the attacks presented in Section 7.4. We find that these attacks, especially the back-gradient attack, considerably deteriorate the dataset, and that our defense can effectively mitigate the attack. Finally, we compare our defense against previously published data poisoning defenses.

7.6.1 Experimental Setup

We evaluate our model on seven numerical and image datasets from Keras [10, 13] and the UCI Machine Learning Repository [142], as presented in Table 7.1. Poison instances D_p^* are sampled from the validation set D_{val} at run time. We use a neural network with one inner layer consisting of 10 neurons, as is appropriate for the small datasets we use. This is in line with related work [30]. For the image datasets, we also perform these experiments with a two-layer convolutional network.

dataset	test loss			test accuracy		
	clean	flip	back-grad	clean	flip	back-grad
breast_cancer	0.14	0.21	0.30	0.96	0.93	0.89
f_mnist_156	0.09	0.24	0.30	0.98	0.92	0.92
f_mnist_17	0.00	0.19	0.19	1.00	0.92	0.92
mnist_156	0.14	0.44	0.50	0.96	0.88	0.88
mnist_17	0.04	0.31	0.56	0.99	0.91	0.86
points	0.21	0.26	0.26	0.94	0.93	0.93
spambase	0.47	0.61	0.87	0.88	0.82	0.79
mean	**0.16**	**0.32**	**0.42**	**0.96**	**0.90**	**0.88**

Table 7.2: Results of applying the label flipping and back-gradient attack against a neural network with one inner layer, averaged over 5 runs. Poisoning 10% of the training data considerably degrades the model's performance with respect to the test loss and test accuracy.

7.6.2 Attack Results

In this section, we present the results of the aforementioned data poisoning attacks on our datasets. We inject up to 10% poison data, and average all results of five individual runs.

The results for attacking an exemplary dataset (*spambase*) are shown in Figure 7.2 and Table 7.2. It can be seen that both the black-box and white-box attacks are effective and that the 'back-gradient' attack consistently outperforms the more naive black-box 'flipping' attack.

Additionally, we evaluate the effectiveness of the 'back-gradient' attack by plotting the model's decision surface. Figure A.6 in the appendix shows a *points* classification task with no adversarial data poisoning, as well as data poisoning induced by the flipping and back-gradient attack. Observe how the attack dramatically alters the decision surface. Table A.5 shows the individual and averaged results for all datasets for five and ten percent of data poisoning. We compare the effectiveness of this attack against corresponding attacks in the domain of regression learning in Section 7.8.

Finally, we apply the 'back-gradient' optimization attack to a convolutional neural network consisting of two convolutional layers ($kernel\_size = (3, 3)$, $pool\_size = (2, 2)$). The authors of [30] report that convolutional neural networks are more resilient against optimal poisoning attacks. We can confirm this result empirically, see Table A.4 in the Appendix.

7.6.3 An Example of Applying our Defence

In this section, we apply our defense against a single, exemplary dataset to further illustrate how the defense works. The complete, empirical evaluation on all datasets is presented in the following section.

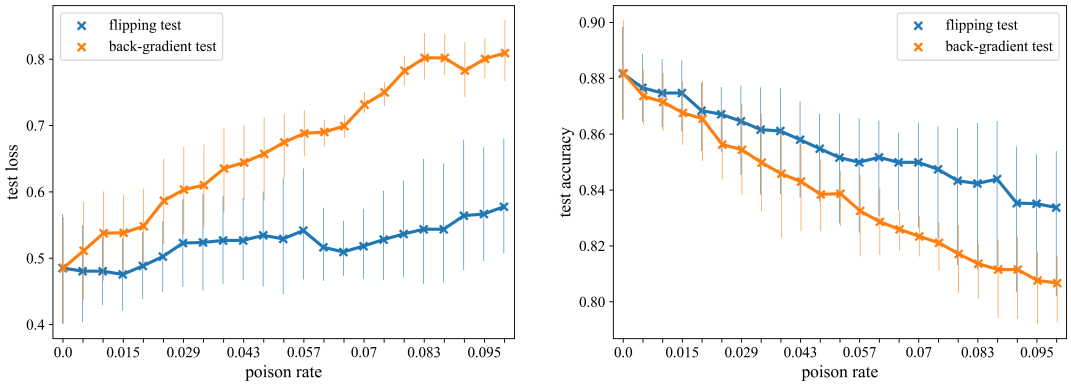


Figure 7.2: Degrading a model's performance by increasing the fraction of poison samples in the *spambase* training dataset. The test loss and accuracy are averaged over 5 runs. Error bars represent the standard deviation. The figure on the left shows the increase of test loss (attacker reward) while the figure on the right shows the decrease of test accuracy when training a model on increasingly poisoned data. Both attacks can be considered effective, however the *back-gradient attack* outperforms *flip* significantly.

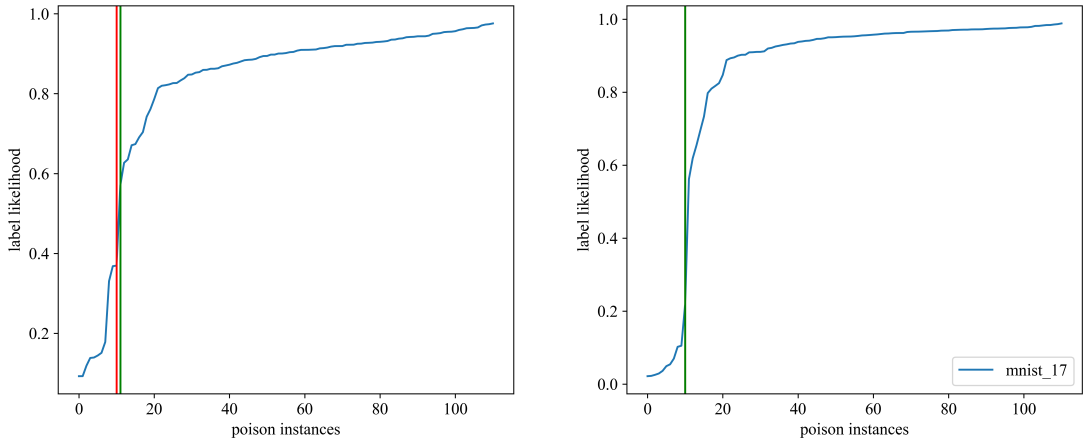


Figure 7.3: The result of applying our defense until convergence (shown: iteration one (left) and two (right)) on the *mnist_17* training dataset. Each image shows a 'knee plots' of label likelihood values l_n (blue), sorted in ascending order. The x-axis lists the l_n-sorted instances, while the y-axis shows the corresponding label likelihood-values l_n. The red line represents the true fraction of poison samples (10%), while the green line represents the fraction estimated by our defence algorithm. Our defence is accurate in estimating the poison rate (c.f. right image, where the red and green line overlap). Additionally, it identifies individual instances with a low $FPR = 0.014$ and a low $FNR = 0.013$. This results in the poisoned test loss being restored from 0.58 to 0.10, and the test accuracy being increased from 0.86 to 0.97.

dataset	$\hat{\nu}$	FP		FN		test loss		test accuracy	
		random	defence	random	defence	before	after	before	after
breast_cancer	0.088	0.09	0.023	0.09	0.034	0.32	0.31	0.89	0.92
fashion_mnist_156	0.107	0.09	0.016	0.09	0.008	0.30	0.16	0.92	0.97
fashion_mnist_17	0.095	0.09	0.000	0.09	0.004	0.18	0.00	0.93	1.00
mnist_156	0.113	0.09	0.022	0.09	0.008	0.51	0.17	0.88	0.96
mnist_17	0.101	0.09	0.014	0.09	0.013	0.58	0.10	0.86	0.97
points	0.070	0.09	0.005	0.09	0.034	0.26	0.24	0.92	0.94
spambase	0.072	0.09	0.028	0.09	0.055	0.80	0.79	0.81	0.84
mean	**0.092**	**0.09**	**0.016**	**0.09**	**0.022**	**0.42**	**0.25**	**0.89**	**0.94**
std	**0.015**	**0.00**	**0.009**	**0.00**	**0.018**	**0.20**	**0.24**	**0.04**	**0.05**

Table 7.3: Results of our defence against the back-gradient optimization attack on a neural network with one inner layer, averaged over 5 runs. The datasets contain 10% poison samples each. The column $\hat{\nu}$ shows the poison rate estimated by the defence algorithm while the next two columns compare FPR and FNR of our defence against a random baseline. Test loss and accuracy improve considerably when using the defence on the poisoned datasets.

Applying our defense. Assume a poisoned dataset D. After applying steps 2-3 from the defense (c.f. Section 7.5.2), we obtain a model M trained on D. Step 4 yields $l_n = \langle y_n, {y'}_n \rangle$ for all instances in the dataset D. Remember that this label likelihood l_n denotes the likelihood of instance n being assigned the original label.

Step 5 consists of sorting all training instances according to l_n. This results in a so-called knee plot. Figure 7.3 presents such a plot for the *mnist_17* dataset. The red line shows the true rate of poison samples in our training dataset ($\epsilon = 10\%$), and the green line is the model's estimate of the poisoning rate. This estimate $\hat{\epsilon}$ is selected as the highest increase in l_n over n, or put differently, $\hat{\epsilon}$ is the arg max of the first derivative of l_n. Since obviously l_n is a discrete function, we approximate its derivative using a sliding window. Finally, in step 7 and 8, we chose as new training data where l_n falls above the threshold, and retrain the model on this data. This process is iterated until convergence, in the example Figure 7.3 for two iterations.

The intuition behind this approach is as follows: The poison estimate $\hat{\epsilon}$ separates values of low and high label likelihood l_n. Low values indicate a mismatch between feature and target values, which may indicate poisoned data. An oracle would assign benign data a label likelihood of $l_n = 1$ and poisoned data a likelihood of $l_n = 0$. Thus, the obvious choice of threshold would be the point of discontinuity in l_n. Since we need to work with an imperfect classifier instead of an oracle, the discontinuity is not as pronounced. However, the largest derivative is a solid approximation which we find to work well.

7.6.4 Defense Results

We apply our defense on data poisoned with the label *back-gradient optimization* attack, the strongest baseline attack in related work, c.f. Section 2.2.2. Table 7.3 presents the

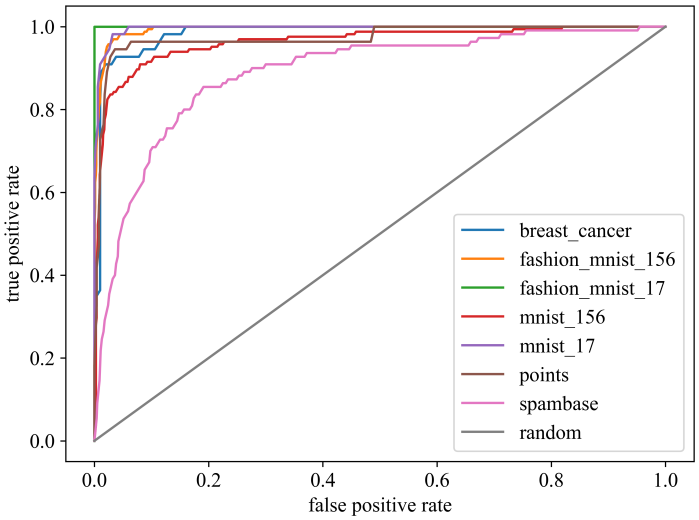


Figure 7.4: ROC curves averaged over 5 runs, obtained by applying our defence on seven datasets poisoned with the back-gradient optimization attack. The x-axis presents the fraction of false positive samples (benign data) while the y-axis presents the fraction of true positive samples (poison data) at varying thresholds.

results. We identify poisoned instances with an averaged $FPR = 0.02$ and $FNR = 0.02$, significantly outperforming a random baseline where both $FPR, FNR = 0.09$. The false-positive rate corresponds to the non-poison samples *left* of the threshold that our classifier wrongly assumes to be poisoned and will be discarded from the training dataset. The false-negative rate corresponds to the poison samples *right* of the threshold that our classifier wrongly assumes to be benign and will remain in the training dataset. Note that our defense does not know the poisoning rate.

For additional illustration, we also present ROC curves, c.f. Figure 7.4, which plots the false positive rate FPR against the true positive rate TPR. An optimal model achieves a $TPR = 1$ with no false-positives (i.e. $FPR = 0$). For the $fashion\_mnist\_17$ dataset, we can present such a perfect model (green line). The random baseline (diagonal black line) is significantly outperformed for all experiments.

Table A.5 in the appendix provides an evaluation of non-poisoned models to demonstrate the utility of our defence. In this case, our method only discards a few samples of low label likelihood with a minimum effect on the test loss and accuracy of the model.

To show that our approach is not limited to a specific learning model, we also apply it on a convolutional neural network (see Table A.4 in the appendix).

7.6.5 Evaluation Against Related Defences

We implement two popular approaches for data-poisoning detection from related work and compare their results to our defense, c.f. Table 7.4. The defense in [138] finds the k nearest neighbors for each sample in the training set using the euclidean distance. If the

dataset	FP				FN			
	[138] (kNN)	[137] (L2)	[137] (LOF)	our defence	[138] (kNN)	[137] (L2)	[137] (LOF)	our defence
breast_cancer	0.034	0.009	0.031	0.023	0.011	0.085	0.050	0.034
fashion_mnist_156	0.017	0.059	0.035	0.016	0.009	0.014	0.032	0.008
fashion_mnist_17	0.000	0.038	0.040	0.000	0.000	0.000	0.000	0.004
mnist_156	0.019	0.125	0.046	0.022	0.023	0.012	0.016	0.008
mnist_17	0.025	0.061	0.034	0.014	0.020	0.032	0.034	0.013
points	0.020	0.025	0.029	0.005	0.013	0.011	0.013	0.034
spambase	0.098	0.012	0.042	0.028	0.051	0.097	0.075	0.055
mean	**0.031**	**0.047**	**0.037**	**0.016**	**0.018**	**0.036**	**0.031**	**0.022**
std	**0.029**	**0.037**	**0.006**	**0.009**	**0.015**	**0.036**	**0.023**	**0.018**

Table 7.4: Comparing our defence to outlier detection used in previous poisoning defences (see [137, 138]), averaged over 5 runs.

fraction of data points with the most common label among these k nearest neighbors is above a given threshold t and if this most common label is different from the actual class of the sample, the sample is considered to be poisoned and will be rejected. Based on the experiments in [138], we use $k = 10$ and $t = 0.6$ for all datasets.

The outlier detection mechanism in [137] first splits a small fraction of trusted data into the different classes and then trains an outlier detector for each class. Every sample is assigned an outlierness score, which is either the Euclidean distance (L2 norm) or the local outlier factor (LOF) with respect to its k nearest neighbors. The threshold to detect outliers is set by using the Empirical Cumulative Distribution Function of the benign training data and identifying the score at a certain α-percentile. Samples with an outlierness score above that threshold are discarded. The parameters $k = 5$ and $\alpha = 0.99$, respectively $\alpha = 0.95$, are chosen according to the authors' suggestions.

Summary. We significantly outperform related defenses, c.f. Table 7.4, both in terms of FPR and FNR. Additionally, we observe that all of these methods suffer from the curse of dimensionality since they are based on a notion of metric distance. Experiments were conducted in real-world conditions: We average over all datasets and supply neither trusted reference data nor access to the ground-truth poisoning rate to our defense.

7.7 Conclusion

In this chapter, we present a new approach to defending against DoS poisoning attacks in classification learning. Our proposed algorithm can detect and discard malicious instances, either by selecting a threshold automatically or by sorting the instances for efficient triage by a human expert. While previous work has mostly used clustering and outlier detection against poisoning attacks, we re-evaluate the training samples by the poisoned classifier. This approach outperforms related state-of-the-art.

The success of our proposed approach shows that learning models themselves can identify adversarial instances, even under the influence of severe data poisoning. Strong regularisation and a focus on generalisation capability allows for detecting data poisoning instances as those points that are *different* from the majority of the remaining data.

7.8 Data Poisoning in Regression and Classification: A Comparison

We now proceed to summarize the key differences and similarities of data poisoning attacks between regression and classification learning. These differences motivate our decision to separate the topic into two parts, Chapter 6 and Chapter 7. This section does not present new technical contributions but provides additional in-depth detail on data-poisoning on regression and classification learning.

7.8.1 Similarities

We will start by examining the similarities.

- **Threat Model**. For both classification and regression, the threat model is identical. A defender trains a model (regressor or classifier) on their dataset. The attacker can introduce a small percentage of poison samples into this dataset. The goal of the attacker, their knowledge, and capabilities may vary but are not dependent on whether the problem is a regression or classification task. Thus, the threat model is identical.
- **Architecture of the Defender's Model**. Also, the architecture of the defender's model and the training process are identical for both classification and regression, or at least very similar. Often, neural networks are employed, which are mostly identical between the two domains. There are only a few differences: a) the loss function (L_p loss for regression, Cross-Entropy-Loss for classification) and b) the final layer (1 neuron with linear activation vs. n neurons with linear or softmax layer). Additionally, data preprocessing and success metrics may vary. However, this is independent of targets being continuous or categorical. These models are always trained with stochastic gradient descent or variants thereof. Also, traditional machine learning techniques such as Support Vector Machines or Decision Trees exist for both regression and classification problems. In summary, the marginal differences in the defender's model do not justify splitting research on data poisoning between regression and classification learning.
- **Unbounded Feature Values**. Also, in both classification and regression, the feature values of the data are bounded/unbounded not because of the target variable, but because of the problem domain. Put differently, whether or not the feature values are constrained to a given interval (i.e. image pixels are constrained to $[0, 1] \in \mathbb{R}$) does not depend on the task being a classification or regression problem, but on the nature of the data. In both a regression and classification scenario, the attacker has to be careful not to exceed the limits of the feature values to avoid a) creating invalid data, and b) creating suspicious data. For example, an adversarial poisoning sample that holds a temperature value of -300 degree Celsius is very easy to spot for any outlier-based defense since absolute zero is -273.15 degree Celsius. In summary, for both classification and regression, the defender has to be

mindful of how to change the feature values. Often, the solution is to introduce a 'feasibility domain', c.f. Section 6.4.2.2.

7.8.2 Differences

However, there are also considerable differences between classification and regression learning in the field of adversarial poisoning attacks. These motivate different algorithms for both creating and defending against adversarial data poisoning.

- **Datasets**. The most obvious difference lies in the available datasets: Classification learning requires data that is labelled categorically, i.e. with classed $0, ..., n-1$. Regression learning requires data with continuous targets, i.e. with unbounded targets $\in \mathbb{R}$ or at least $\in \mathbb{N}$. The meaning of these targets is very different: In classification learning, the metric distance between two targets has *no* meaning. For example, class 0 is not more similar to class 1 than to class 9. This becomes intuitively clear when looking at the CIFAR-10 image dataset: Class 0 represents airplanes, class 1 represents cars, and class 9 represents trucks. It is nonsensical to assume that an airplane is 9 times more similar to a car than a truck. However, in regression learning, the converse is true: The targets usually represent some measurable quantity like weight, speed, gram, dollar, bits per second, etc., which allow for comparability.
- **Manipulation of Target Values**. The difference in target values between regression and classification problems also affects adversarial data poisoning: In regression learning, an attacker usually shifts the targets towards extreme values at the border of the feasibility domain (c.f. the *StatP* or *Flip* attack, Section 6.4.2.2). In contrast, for classification learning, there are no such 'extreme' target values the poisoned instance may assume. This is because the targets are categorical, and the notion of distance is not defined as explained above. It is not clear how an attacker can choose optimal targets for their poisoned instances. In fact, related work relies heavily on choosing the targets randomly (c.f. *Backgradient attack* or *Random Flip attack*, Section 7.4.3).
- **Effectiveness of Attack.** When comparing the strongest data-poisoning attacks in literature in the domain of regression learning vs. classification learning, we observe a discrepancy in effectiveness. Attacks on regression learning are far more effective: With about two to four percent of poison data, the test error is increased to more than 200 percent (c.f. Figure 6.3). In a corresponding classification task, the accuracy drops from an averaged 0.95 to 0.92 for five percent of data poisoning. This means that the model's 'rate of failure' increases from 5% to 8%, which is equivalent to an increase of test error to 160% (c.f. Table A.5). In summary, when poisoning datasets, we observed that SOTA data poisoning is less effective in classification learning than in regression learning. This is because regression learning allows for modifying the feature values to the extremes, whereas in classification learning, the attacker is largely confined to changing the feature values (c.f. Section 7.4.3).

- **Applicability of Defense**. The difference in target values motivates different defense strategies because the meaning and interpretability of the targets differ between the domains of classification and regression. Consider for example our proposed defense in Section 7.5.2. To compute the label likelihood $l_n = \langle y_i, y_i' \rangle$, a categorical target vector y_i is required which can be interpreted as a probability distribution. While classification targets are naturally interpretable as a probability distribution, regression targets are not, since they represent the value of some quantity. Conversely, the *Trim* and *iTrim* defense identify data poisoning by sorting all instances by the magnitude of the training loss they incur. While this motivates our proposed data poisoning defense for classification (c.f. Chapter 7), migrating this defense to the domain of classification incurs a change in loss function and target values, which in turn requires that this defense substitute the training loss with the label likelihood l_n.
- **Related Work**. Finally, the amount of related work also differs between poisoning classification and regression data. We note that there is significantly more research on 'classification' than on 'regression' learning: Google Scholar lists more than five times more related work for the search term 'data poisoning classification' vs 'data poisoning regression': 981k vs 190k on Dec. 17th, 2020. These numbers are obviously inflated, but show that classification is studied more than regression learning. We can confirm that there is significantly more related work for classification poisoning, especially in the domain of defenses against data poisoning. This may be because humans are more interested and better at interpreting classification data (e.g. images) than regression (numbers & measurements), thus it attracts more research.

In summary, while many aspects of data poisoning are consistent throughout regression and classification learning, there are considerable differences: Most importantly, regression targets are interpreted differently from classification targets, from which follows that attack and defense strategies need to be adapted.

8 Conclusion

This chapter concludes the thesis and reflects on its contributions. It summarizes our answer to the proposed research question, discusses current research trends in this field, and finally concludes with suggestions for future research.

8.1 Answering the Research Question

The main research question of this thesis is: What influences (i.e., both attacks and data errors) compromise the integrity and correctness of ML datasets, and how can this be counteracted?

Our answer is as follows: The integrity and correctness of ML datasets can be compromised by either *random* or *deliberate* adversaries; i.e. errors in the data, or adversarial data poisoning attacks. Thus, the integrity and correctness of ML datasets can be maintained by employing data-sanitation techniques that address both of these scenarios:

- **Random Adversary / Data Errors**. To assert the correctness of the data labels against labeling mistakes, we propose to use our tool *LabelFix* as presented in [3], which has also been published on GitHub, see `www.github.com/mueller91/labelfix`. We show that it reliably detects labeling mistakes and thus contributes to the correctness of ML data. To guard against data errors that lead to learning shortcuts, we suggest careful exploratory analysis (as in [1]) and information-aggregation schemes such as presented in equation (3.1).
- **Deliberate Adversary**. If possibly untrusted data has been incorporated into the training set, one should use a sanitation algorithm such as we present in [4] or [5], depending on the target domain. We show that these algorithms can reliably estimate the presence and degree of data poisoning, so it is always a good idea to apply them to learn more about one's dataset and maintain integrity. When using active learning, one has to take the possibility of data poisoning into account [2] and should consider adversarial retraining.

Our algorithms beat state-of-the-art in adversarial settings and find previously unpublished errors in some of the most popular machine-learning datasets. We show that the algorithms can be applied in realistic scenarios and are viable when the degree or presence of data poisoning is unknown. They run efficiently even when only limited computing power is available. In summary, we argue that it is always a good idea to apply our proposed algorithms to a newly created dataset. Compared to the effort of collecting, processing, and especially labeling the data, the overhead induced by our algorithms is negligible, but there is a tangible benefit in catching errors and malicious data before the dataset is used in downstream tasks or even published.

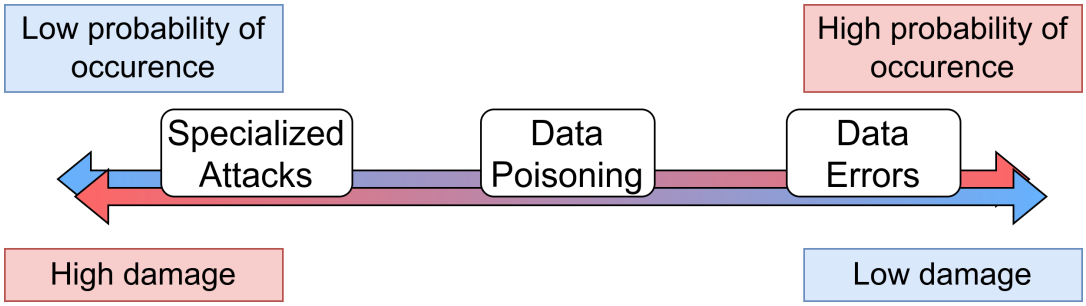


Figure 8.1: Work presented in this thesis on a spectrum 'probability of occurrence' and 'damage'. We identify three categories: First, specialized attacks such as 'adversarial active transfer learning', e.g. work presented in chapter 4. These attacks have a lower probability of occurrence since they require a rather specialized setup. The damage inflicted however is high, since the active learning algorithm is completely broken, thus human annotation effort is wasted. Second, work on data poisoning, c.f. chapter 6 and chapter 7. These threats have a medium probability of occurrence and inflict medium damage (the dataset has to be cleaned, but no manual labeling effort is lost). Third, data errors (c.f. chapter 5 and chapter 3), which have a high probability of occurrence, but are often thought of as not too important and/or impactful.

8.2 Evaluating Current Research Trends

A natural way to structure the work presented in this thesis is to place them on a spectrum between two poles. Let us consider two spectra: First, the *damage* or harm inflicted by the attacks and errors presented in the previous chapters. We define damage as the amount of resources (human effort and computation time) required to undo the negative results of the attacks or dataset errors. Second, we structure our work by the *probability of occurrence*, i.e. sort from most likely to least likely. When aligning our work on these two spectra, three subgroups emerge. We visualize this in figure 8.1.

- **Specialized Attacks.** First, there are specialized attacks such as 'adversarial active transfer learning', e.g. work presented in chapter 4. These attacks inflict high damage if successful since the active learning algorithm is completely broken. This is because the human annotators are provided with attacker-chosen instances instead of those which would benefit the model the most. Thus, the advantageous properties of the active-learning setup are completely negated. This is especially harmful since human annotation is very time-consuming, and repetition is costly. However, these attacks are rather specialized and thus have a lower probability of occurrence than other attacks.
- **Data Poisoning**. The second group is work on data poisoning, c.f. chapter 6 and chapter 7. These threats have a medium probability of occurrence: They require some attacker capabilities but are rather straightforward to execute and do not target specialized machine-learning pipelines, but common classification and

regression models. They inflict medium damage: the dataset has to be cleaned and the model retrained, but no manual labeling effort is lost.

- **Data Errors**. Third, there are data and labelling errors, c.f. chapter 5 and chapter 3. These have a high probability of occurrence, as we have shown: We find labeling errors in many of the state-of-the-art datasets, c.f.chapter 5. The same is true for data errors, which we identify in the most-established audio-deepfake detection datasetchapter 3. The impact of these attacks ranges from limited to severe. There is a tendency to underestimate the impacts.

Having defined *probability of occurrence* and *damage*, it is natural to conceptualize the *riskiness* as the product of both:

$$\text{riskiness} = \text{damage} \times \text{probability}$$

Thus, *riskiness* is the expected damage. We observe that the *riskiness* is approximately constant across all three groups described above. This is because where one factor is high, the other one is low, and vice-versa. Interestingly enough, this equal *riskiness* between the three groups does not lead to equal research effort by the scientific community. We select three representative queries for each of the groups and run them against Google Scholar and the Digital Bibliography & Library Project (DBLP), two of the major search engines for scientific publication. The results confirm that 'adversarial data poisoning' is much more researched than data errors and specialized attacks. Table 8.1 lists our queries and the number of hits. While this table is only a rough approximation of the research landscape, it shows that there is about 4-10 times more research for the popular 'adversarial data poisoning' topic. This corresponds to our perception of the scientific landscape, and also explains why the data artifact in ASVspoof 2019 has not been identified before: other topics seem to attract more interest and thus, more research effort - even though the data artifact has massive impact on all work performed.

In summary, even though all three groups should warrant equal attention, the scientific community has picked its favorite and somewhat ignores both high-damage, low-probability attacks, as well as (perceived) low-damage, high-probability data errors. We hypothesize that this is because the former seems 'too unlikely' to worry about, while the latter is ignored because it seems not harmful enough. However, we argue against this trend: Since all groups share the same riskiness, they should receive equal research attention.

8.3 Future Work

From the analysis in the previous chapter flows our suggestions for future work.

Most importantly, we suggest allocating equal research effort for equal riskiness. This means that both specialized attacks and dataset errors should not be neglected. For example, we need to find new ways to identify data artifacts that can lead to learning shortcuts [1]. We make some concrete suggestions in section 3.5.2. Solutions to this problem will not only prevent dataset errors but also help models generalize beyond the training dataset. They will also improve our understanding of the learning models,

Query	Google Scholar	DBLP
active transfer data poison	174k	0
adversarial active learning	171k	63
transfer learning data poisoning	66k	0
data poisoning	2600k	180
adversarial data	674k	993
data poison attack	403k	127
shortcut learning	129k	29
data artefact machine learning	97k	0
mislabelled data	48k	41

Table 8.1: This table shows the number of hits when querying Google Scholar (in thousands) and DBLP for various keywords from the field of ML data correctness. While Google Scholar is much less restrictive than DBLP, we observe a similar trend: Firstly, there is a large body of work on 'data poisoning' (middle). Specialized attacks such as 'active transfer data poisoning' (top) and dataset-related issues (bottom) are much less researched. This favoritism seems unjustified since all three groups share similar riskiness. All results as per 02.03.2022.

making their predictions explainable, which can lead to greater trust and thus a more widespread use.

Apart from this, it would be interesting to transfer our technical approach to other fields of IT security: the idea to identify errors or attacks by re-evaluating contaminated data could also be employed in the domain of anomaly detection. For example, one might train a classification model with heavy regularisation on a dataset with outliers. As presented in chapter 5, the label probability l_n could identify which instances are outliers. It would be interesting to apply this approach to other domains, such as predictive maintenance or health-related use-cases. Forgoing traditional anomaly detection techniques comes with some advantages (no threshold detection, avoiding the curse of dimensionality), so this technique may prove fruitful beyond the domain of adversarial machine learning.

Bibliography

[1] N. M. Müller, F. Dieckmann, P. Czempin, R. Canals, K. Böttinger, and J. Williams. Speech is silver, silence is golden: What do asvspoof-trained models really learn? *ASVSpoof*, 2021.

[2] N. M. Müller and K. Böttinger. Adversarial vulnerability of active transfer learning. In *19th Symposium on Intelligent Data Analysis (IDA)*, 2021.

[3] N. M. Müller and K. Markert. Identifying mislabeled instances in classification datasets. In *2019 International Joint Conference on Neural Networks (IJCNN)*, 2019.

[4] N. M. Müller, D. Kowatsch, and K. Böttinger. Data poisoning attacks on regression learning and corresponding defenses. In *25th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC)*, 2020.

[5] N. M. Müller, S. Roschmann, and K. Böttinger. Defending against adversarial denial-of-service data poisoning attacks. In *DYNAMICS Workshop, Annual Computer Security Applications Conference (ACSAC)*, 2020.

[6] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[7] M. Jagielski, A. Oprea, B. Biggio, C. Liu, C. Nita-Rotaru, and B. Li. Manipulating Machine Learning: Poisoning Attacks and Countermeasures for Regression Learning. In *Proceedings - IEEE Symposium on Security and Privacy*, volume 2018-May, pages 19–35. IEEE, may 2018. URL: `https://ieeexplore.ieee.org/document/8418594/`, `doi:10.1109/SP.2018.00057`.

[8] R. Bellman. Dynamic programming treatment of the travelling salesman problem. *Journal of the ACM (JACM)*, 9(1):61–63, 1962.

[9] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[10] Y. LeCun, C. Cortes, and C. Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[11] A. Krizhevsky, V. Nair, and G. Hinton. Cifar-100 (canadian institute for advanced research). URL: `http://www.cs.toronto.edu/~kriz/cifar.html`.

[12] N. M. Müller, D. Kowatsch, P. Debus, D. Mirdita, and K. Böttinger. On gdpr compliance of companies' privacy policies. In *International Conference on Text, Speech, and Dialogue*, pages 151–159. Springer, 2019.

[13] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[14] N. M. Müller, P. Debus, and K. Böttinger. Distributed anomaly detection of single mote attacks in rpl networks. In *Proceedings of the 16th International Joint Conference on e-Business and Telecommunications*, volume 2, 2019.

[15] T. Dörr, K. Markert, N. M. Müller, and K. Böttinger. Towards resistant audio adversarial examples. In *Proceedings of the 1st ACM Workshop on Security and Privacy on Artificial Intelligence*, pages 3–10, 2020.

[16] G. Atkinson and V. Metsis. Identifying label noise in time-series datasets. In *Adjunct Proceedings of the 2020 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2020 ACM International Symposium on Wearable Computers*, pages 238–243, 2020.

[17] Mnist on benchmarks.ai. `https://benchmarks.ai/mnist`. (Accessed on 05/18/2022).

[18] J. Y. et Al. Asvspoof 2021 worshop slides: Accelerating progress in spoofed and deepfake speech detection. *ASVSpoof*, 16th September 2021.

[19] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[20] P. Foret, A. Kleiner, H. Mobahi, and B. Neyshabur. Sharpness-aware minimization for efficiently improving generalization. *arXiv preprint arXiv:2010.01412*, 2020.

[21] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[22] S. Ioffe. Batch renormalization: Towards reducing minibatch dependence in batch-normalized models. In *Advances in neural information processing systems*, pages 1945–1953, 2017.

[23] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1608.04747*, 2015.

[24] Y. Wang, R. Skerry-Ryan, D. Stanton, Y. Wu, R. J. Weiss, N. Jaitly, Z. Yang, Y. Xiao, Z. Chen, S. Bengio, et al. Tacotron: Towards end-to-end speech synthesis. *arXiv preprint arXiv:1703.10135*, 2017.

[25] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[26] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[27] L. Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[28] B. Biggio, B. Nelson, and P. Laskov. Poisoning attacks against support vector machines. *arXiv preprint arXiv:1206.6389*, 2012.

[29] H. Xiao, B. Biggio, G. Brown, G. Fumera, C. Eckert, and F. Roli. Is feature selection secure against training data poisoning? In *International Conference on Machine Learning*, pages 1689–1698, 2015.

[30] L. Muñoz-González, B. Biggio, A. Demontis, A. Paudice, V. Wongrassamee, E. C. Lupu, and F. Roli. Towards poisoning of deep learning algorithms with back-gradient optimization. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 27–38, 2017.

[31] A. Shafahi, W. Ronny Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein. Poison frogs! Targeted clean-label poisoning attacks on neural networks. In *Advances in Neural Information Processing Systems*, volume 2018-Decem, pages 6103–6113, 2018. URL: `http://papers.nips.cc/paper/7849-poison-frogs-targeted-clean-label-poisoning-attacks-on-neural-networks.pdf`, `arXiv:1804.00792`.

[32] I. Evtimov, K. Eykholt, E. Fernandes, T. Kohno, B. Li, A. Prakash, A. Rahmati, and D. Song. Robust physical-world attacks on machine learning models. *arXiv preprint arXiv:1707.08945*, 2(3):4, 2017.

[33] Y. Dong, H. Su, B. Wu, Z. Li, W. Liu, T. Zhang, and J. Zhu. Efficient decision-based black-box adversarial attacks on face recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7714–7722, 2019.

[34] Inside the plot to bomb borussia dortmund soccer team to make money. `https://money.cnn.com/2017/04/21/investing/dortmund-bombing-stock-plot/`. (Accessed on 12/02/2020).

[35] S. G. Finlayson, J. D. Bowers, J. Ito, J. L. Zittrain, A. L. Beam, and I. S. Kohane. Adversarial attacks on medical machine learning. *Science*, 363(6433):1287–1289, 2019.

[36] M. Carminati, L. Santini, M. Polino, and S. Zanero. Evasion attacks against banking fraud detection systems. In *23rd International Symposium on Research in Attacks, Intrusions and Defenses ({RAID} 2020)*, pages 285–300, 2020.

[37] N. H. Imam and V. G. Vassilakis. A survey of attacks against twitter spam detectors in an adversarial environment. *Robotics*, 8(3):50, 2019.

[38] H. S. Anderson, A. Kharkar, B. Filar, and P. Roth. Evading machine learning malware detection. *Black Hat*, 2017.

[39] N. Carlini and D. Wagner. Audio adversarial examples: Targeted attacks on speech-to-text. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 1–7. IEEE, 2018.

[40] H. Yakura and J. Sakuma. Robust audio adversarial example for a physical attack. *arXiv preprint arXiv:1810.11793*, 2018.

[41] A. Athalye, L. Engstrom, A. Ilyas, and K. Kwok. Synthesizing robust adversarial examples. In *International conference on machine learning*, pages 284–293. PMLR, 2018.

[42] B. Miller, A. Kantchelian, S. Afroz, R. Bachwani, E. Dauber, L. Huang, M. C. Tschantz, A. D. Joseph, and J. D. Tygar. Adversarial active learning. In *Proceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop*, pages 3–14, 2014.

[43] C. Guo, J. R. Gardner, Y. You, A. G. Wilson, and K. Q. Weinberger. Simple black-box adversarial attacks. *arXiv preprint arXiv:1905.07121*, 2019.

[44] S. Lapuschkin, S. Wäldchen, A. Binder, G. Montavon, W. Samek, and K.-R. Müller. Unmasking clever hans predictors and assessing what machines really learn. *Nature communications*, 10(1):1–8, 2019.

[45] Audio deep fake: Demonstrator entwickelt am fraunhofer aisec - youtube. `https://www.youtube.com/watch?v=MZTF0eAALmE`. (Accessed on 02/22/2022).

[46] A voice deepfake was used to scam a ceo out of $243,000. `https://www.forbes.com/sites/jessedamiani/2019/09/03/a-voice-deepfake-was-used-to-scam-a-ceo-out-of-243000`. (Accessed on 08/16/2021).

[47] J. Yamagishi, M. Todisco, M. Sahidullah, H. Delgado, X. Wang, N. Evans, T. Kinnunen, K. A. Lee, V. Vestman, and A. Nautsch. Asvspoof 2019: The 3rd automatic speaker verification spoofing and countermeasures challenge database. 2019.

[48] Asvspoof 2021: Automatic speaker verification spoofing and countermeasures challenge. https://www.asvspoof.org/index2019.html. Accessed: 2021-06-11.

[49] C. Veaux, J. Yamagishi, K. MacDonald, et al. Superseded-cstr vctk corpus: English multi-speaker corpus for cstr voice cloning toolkit. 2017.

[50] A. Chintha, B. Thai, S. J. Sohrawardi, K. Bhatt, A. Hickerson, M. Wright, and R. Ptucha. Recurrent convolutional structures for audio spoof and video deepfake detection. *IEEE Journal of Selected Topics in Signal Processing*, 14, 2020.

[51] M. Alzantot, Z. Wang, and M. B. Srivastava. Deep residual neural networks for audio spoofing detection. *arXiv preprint arXiv:1907.00501*, 2019.

[52] B. Chettri, D. Stoller, V. Morfi, M. A. M. Ramírez, E. Benetos, and B. L. Sturm. Ensemble models for spoofing detection in automatic speaker verification. *arXiv preprint arXiv:1904.04589*, 2019.

[53] Z. Wang, S. Cui, X. Kang, W. Sun, and Z. Li. Densely connected convolutional network for audio spoofing detection. In *2020 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, pages 1352–1360. IEEE, 2020.

[54] X. Wang, J. Yamagishi, M. Todisco, H. Delgado, A. Nautsch, N. Evans, M. Sahidullah, V. Vestman, T. Kinnunen, K. A. Lee, et al. Asvspoof 2019: A large-scale public database of synthesized, converted and replayed speech. *Computer Speech & Language*, 64:101114, 2020.

[55] Y. Wang, D. Stanton, Y. Zhang, R.-S. Ryan, E. Battenberg, J. Shor, Y. Xiao, Y. Jia, F. Ren, and R. A. Saurous. Style tokens: Unsupervised style modeling, control and transfer in end-to-end speech synthesis. In *International Conference on Machine Learning*, pages 5180–5189. PMLR, 2018.

[56] Y. Jia, Y. Zhang, R. J. Weiss, Q. Wang, J. Shen, F. Ren, Z. Chen, P. Nguyen, R. Pang, I. L. Moreno, et al. Transfer learning from speaker verification to multispeaker text-to-speech synthesis. *arXiv preprint arXiv:1806.04558*, 2018.

[57] H. Tak, J. Patino, M. Todisco, A. Nautsch, N. Evans, and A. Larcher. End-to-end anti-spoofing with rawnet2. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6369–6373. IEEE, 2021.

[58] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[59] J. C. Brown. Calculation of a constant q spectral transform. *The Journal of the Acoustical Society of America*, 89(1):425–434, 1991.

[60] B. M. et Al. librosa/librosa: 0.8.1rc2, May 2021. URL: `https://doi.org/10.5281/zenodo.4792298`, `doi:10.5281/zenodo.4792298`.

[61] R. Kubichek. Mel-cepstral distance measure for objective speech quality assessment. In *Proceedings of IEEE Pacific Rim Conference on Communications Computers and Signal Processing*, volume 1, pages 125–128. IEEE, 1993.

[62] S. Davis and P. Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE transactions on acoustics, speech, and signal processing*, 28(4):357–366, 1980.

[63] M. McAuliffe, M. Socolof, S. Mihuc, M. Wagner, and M. Sonderegger. Montreal forced aligner: Trainable text-speech alignment using kaldi. In *Interspeech*, volume 2017, pages 498–502, 2017.

[64] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[65] B. Settles. Active learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2009.

[66] D. Kale and Y. Liu. Accelerating active learning with transfer learning. In *2013 IEEE 13th International Conference on Data Mining*, pages 1085–1090, Dec 2013. doi:10.1109/ICDM.2013.160.

[67] L. Yang, S. Hanneke, and J. Carbonell. A theory of transfer learning with applications to active learning. *Machine Learning*, 90, 02 2013. doi:10.1007/s10994-012-5310-y.

[68] X. Wang, T.-K. Huang, and J. Schneider. Active transfer learning under model shift. In E. P. Xing and T. Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 1305–1313, Bejing, China, 22–24 Jun 2014. PMLR. URL: http://proceedings.mlr.press/v32/wangi14.html.

[69] Y. S. Chan and H. T. Ng. Domain adaptation with active learning for word sense disambiguation. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 49–56, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL: https://www.aclweb.org/anthology/P07-1007.

[70] X. Shi, W. Fan, and J. Ren. Actively transfer domain knowledge. In *Proceedings of the 2008th European Conference on Machine Learning and Knowledge Discovery in Databases - Volume Part II*, ECMLPKDD'08, page 342–357, Berlin, Heidelberg, 2008. Springer-Verlag.

[71] K. Li, T. Zhang, and J. Malik. Approximate feature collisions in neural nets. In *Advances in Neural Information Processing Systems*, pages 15842–15850, 2019.

[72] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[73] M. Plakal and D. Ellis. Yamnet. Jan 2020. github.com/tensorflow/models/tree/master/research/audioset/yamnet.

[74] Resnet and resnetv2. https://keras.io/api/applications/resnet/. (Accessed on 11/26/2020).

[75] A. Shafahi, W. R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein. Poison frogs! targeted clean-label poisoning attacks on neural networks. In *Advances in Neural Information Processing Systems*, pages 6103–6113, 2018.

[76] Google audio set. `https://research.google.com/audioset/`, 2017. (Accessed on 11/26/2020).

[77] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. Citeseer 2009, 2009.

[78] Stl-10 dataset. `http://ai.stanford.edu/~acoates/stl10/`, 2011. (Accessed on 11/26/2020).

[79] B. Li, Y. Vorobeychik, and X. Chen. A general retraining framework for scalable adversarial classification. *arXiv preprint arXiv:1604.02606*, 2016.

[80] Mislabeled instances found · issue #166 · zalandoresearch/fashion-mnist. `https://github.com/zalandoresearch/fashion-mnist/issues/166`. (Accessed on 11/19/2020).

[81] B. Frénay and A. Kabán. A comprehensive introduction to label noise. In *ESANN*, 2014.

[82] B. Frénay and M. Verleysen. Classification in the presence of label noise: a survey. *IEEE transactions on neural networks and learning systems*, 25(5):845–869, 2014.

[83] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[84] Y. Li, J. Yang, Y. Song, L. Cao, J. Luo, and L.-J. Li. Learning from noisy labels with distillation. In *ICCV*, pages 1928–1936, 2017.

[85] J. Bootkrajang. A generalised label noise model for classification in the presence of annotation errors. *Neurocomputing*, 192:61–71, 2016.

[86] D. Hendrycks, M. Mazeika, D. Wilson, and K. Gimpel. Using trusted data to train deep networks on labels corrupted by severe noise. *arXiv preprint arXiv:1802.05300*, 2018.

[87] C. E. Brodley and M. A. Friedl. Identifying mislabeled training data. *Journal of artificial intelligence research*, 11:131–167, 1999.

[88] M. Sabzevari, G. Martínez-Muñoz, and A. Suárez. A two-stage ensemble method for the detection of class-label noise. *Neurocomputing*, 275:2374–2383, 2018.

[89] R. Ekambaram, D. B. Goldgof, and L. O. Hall. Finding label noise examples in large scale datasets. In *Systems, Man, and Cybernetics (SMC), 2017 IEEE International Conference on*, pages 2420–2424. IEEE, 2017.

[90] M. Al-Rawi and D. Karatzas. On the labeling correctness in computer vision datasets. In *Proceedings of the Workshop on Interactive Adaptive Learning*, pages 1–23, 2018.

[91] A. Esuli and F. Sebastiani. Improving text classification accuracy by training label cleaning. *ACM Transactions on Information Systems (TOIS)*, 31(4):19, 2013.

[92] A. Kolcz and G. V. Cormack. Genre-based decomposition of email class noise. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 427–436. ACM, 2009.

[93] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013. URL: `http://arxiv.org/abs/1310.4546`, `arXiv:1310.4546`.

[94] E. (Github). eyaler/word2vec-slim: word2vec google news model slimmed down to 300k english words. https://github.com/eyaler/word2vec-slim. (Accessed on 03/28/2019).

[95] J. Leskovec, A. Rajaraman, and J. D. Ullman. *Mining of massive datasets*, pages 1–19. Cambridge university press, 2014.

[96] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[97] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[98] D. Dheeru and E. Karra Taniskidou. UCI machine learning repository, 2017. URL: `http://archive.ics.uci.edu/ml`.

[99] I.-C. Yeh and C.-h. Lien. The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. *Expert Systems with Applications*, 36(2):2473–2480, 2009.

[100] T. A. Almeida, J. M. G. Hidalgo, and A. Yamakami. Contributions to the study of sms spam filtering: new collection and results. In *Proceedings of the 11th ACM symposium on Document engineering*, pages 259–262. ACM, 2011.

[101] F. Chollet et al. Keras. https://keras.io, 2015.

[102] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL: `http://www.aclweb.org/anthology/P11-1015`.

[103] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2/2011, page 5, 2011.

[104] S. Ekins, A. C. Puhl, K. M. Zorn, T. R. Lane, D. P. Russo, J. J. Klein, A. J. Hickey, and A. M. Clark. Exploiting machine learning for end-to-end drug discovery and development. *Nature materials*, 18(5):435, 2019.

[105] H. Kuchler. The start-up striving to accelerate drug discovery, May 2019. URL: https://www.ft.com/content/374a3aa8-6bf9-11e9-80c7-60ee53e6681d.

[106] R. Wigglesworth. Why hedge fund managers are happy to let the machines take over, Oct 2019. URL: https://www.ft.com/content/338962c0-eeaf-11e9-ad1e-4367d8281195.

[107] K. Porzecanski. Jpmorgan commits hedge fund to ai in technology arms race, Jul 2019. URL: https://www.bloomberg.com/news/articles/2019-07-02/jpmorgan-to-start-ai-hedge-fund-strategy-in-technology-arms-race.

[108] Handelsblatt. Zahlungsverhalten vorhersagen, Oct 2018. URL: https://www.handelsblatt.com/adv/verovis/zahlungsverhalten-vorhersagen-cash-forecasting-mit-predictive-analytics/23117258.html?ticket=ST-49709747-JhbnSilLfnpK0OTZfGf2-ap5.

[109] PWC. *Predictive Maintenance 4.0*. 2017. URL: https://www.pwc.nl/nl/assets/documents/pwc-predictive-maintenance-4-0.pdf.

[110] U. A. de Ciudad Juárez. (pdf) using regression models for predicting the product quality in a tubing extrusion process, Aug 2019. URL: https://www.researchgate.net/publication/324132141_Using_regression_models_for_predicting_the_product_quality_in_a_tubing_extrusion_process.

[111] W. Contributors. Warfarin, Oct 2019. URL: https://en.wikipedia.org/wiki/Warfarin#Dosing.

[112] IWPC. Pharmgkb. downloads - iwcp dataset, 2019. URL: https://www.pharmgkb.org/page/downloadDrugsHelp.

[113] A. Sharabiani, A. Bress, E. Douzali, and H. Darabi. Revisiting warfarin dosing using machine learning techniques. *Computational and Mathematical Methods in Medicine*, 2015:1–9, 2015. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4471424/, doi:10.1155/2015/560108.

[114] Z. Ma, P. Wang, Z. Gao, R. Wang, and K. Khalighi. Ensemble of machine learning algorithms using the stacked generalization approach to estimate the warfarin dose. *PloS one*, 13(10):e0205872, 2018. URL: https://www.ncbi.nlm.nih.gov/pubmed/30339708, doi:10.1371/journal.pone.0205872.

[115] G. Truda and P. Marais. Evaluating warfarin dosing models on multiple datasets with a novel software framework and evolutionary optimisation. *Journal of Biomedical Informatics*, page 103634, 2020.

[116] M. Eddy. Hundreds of bodies, one nurse: German serial killer leaves as many questions as victims. *The New York Times*, May 2019. URL: https://www.nytimes.com/2019/05/10/world/europe/germany-serial-killer-nurse.html.

[117] T. Rogers. The get-rich-quick scheme that almost killed a german soccer team, Oct 2018. URL: https://www.bloomberg.com/news/features/2018-10-29/the-get-rich-quick-scheme-that-almost-killed-a-german-soccer-team.

[118] S. Chen, M. Xue, L. Fan, S. Hao, L. Xu, H. Zhu, and B. Li. Automated poisoning attacks and defenses in malware detection systems: An adversarial machine learning approach. *Computers and Security*, 73:326–344, 2018. URL: https://arxiv.org/pdf/1706.04146.pdf, doi:10.1016/j.cose.2017.11.007.

[119] R. Perdisci, D. Dagon, W. Lee, P. Foglat, and M. Sharif. Misleading worm signature generators using deliberate noise injection. In *Proceedings - IEEE Symposium on Security and Privacy*, volume 2006, pages 17–31, 2006. URL: https://personal.utdallas.edu/{~}muratk/courses/dmsec{_}files/ieee-sp-06.pdf, doi:10.1109/SP.2006.26.

[120] Applications - Keras Documentation, Sep 2019. [Online; accessed 6. Nov. 2019]. URL: https://keras.io/applications.

[121] J. Steinhardt, P. W. Koh, and P. Liang. Certified defenses for data poisoning attacks. In *Advances in Neural Information Processing Systems*, volume 2017-Decem, pages 3518–3530, 2017. URL: http://papers.nips.cc/paper/6943-certified-defenses-for-data-poisoning-attacks.pdf, arXiv:1706.03691.

[122] P. Branco. GitHub Imbalanced-Regression-Datasets, 2019. https://github.com/paobranco/Imbalanced-Regression-DataSets, last checked September 11th, 2020.

[123] J. Alcalá-Fdez, A. Fernández, J. Luengo, J. Derrac, S. García, L. Sánchez, and F. Herrera. KEEL Data-Mining Software Tool: Data Set Repository, Integration of Algorithms and Experimental Analysis Framework. Technical report, 2011. URL: http://the-data-mine.com/bin/view/Software.

[124] P. J. Huber. Robust estimation of a location parameter. In *Breakthroughs in statistics*, pages 492–518. Springer, 1992.

[125] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.

[126] A. Graves, A.-r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE, 2013.

[127] W. Xiong, J. Droppo, X. Huang, F. Seide, M. Seltzer, A. Stolcke, D. Yu, and G. Zweig. Achieving human parity in conversational speech recognition. *arXiv preprint arXiv:1610.05256*, 2016.

[128] Y. Sun, X. Wang, and X. Tang. Deep learning face representation from predicting 10,000 classes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1891–1898, 2014.

[129] R. Wang, C. Han, Y. Wu, and T. Guo. Fingerprint classification based on depth neural network. *arXiv preprint arXiv:1409.5188*, 2014.

[130] B. Huval, T. Wang, S. Tandon, J. Kiske, W. Song, J. Pazhayampallil, M. Andriluka, P. Rajpurkar, T. Migimatsu, R. Cheng-Yue, et al. An empirical evaluation of deep learning on highway driving. *arXiv preprint arXiv:1504.01716*, 2015.

[131] B. Nelson, M. Barreno, F. J. Chi, A. D. Joseph, B. I. Rubinstein, U. Saini, C. A. Sutton, J. D. Tygar, and K. Xia. Exploiting machine learning to subvert your spam filter. *LEET*, 8:1–9, 2008.

[132] J. Saxe and K. Berlin. Deep neural network based malware detection using two dimensional binary program features. In *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*, pages 11–20. IEEE, 2015.

[133] A. Javaid, Q. Niyaz, W. Sun, and M. Alam. A deep learning approach for network intrusion detection system. In *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS)*, pages 21–26, 2016.

[134] B. Biggio, K. Rieck, D. Ariu, C. Wressnegger, I. Corona, G. Giacinto, and F. Roli. Poisoning behavioral malware clustering. In *Proceedings of the 2014 workshop on artificial intelligent and security workshop*, pages 27–36, 2014.

[135] B. Biggio, B. Nelson, and P. Laskov. Support vector machines under adversarial label noise. In *Asian conference on machine learning*, pages 97–112, 2011.

[136] P. W. Koh and P. Liang. Understanding black-box predictions via influence functions. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1885–1894. JMLR. org, 2017.

[137] A. Paudice, L. Muñoz-González, A. Gyorgy, and E. C. Lupu. Detection of adversarial training examples in poisoning attacks through anomaly detection. *arXiv preprint arXiv:1802.03041*, 2018.

[138] A. Paudice, L. Muñoz-González, and E. C. Lupu. Label sanitization against label flipping poisoning attacks. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 5–15. Springer, 2018.

[139] E. Keogh and A. Mueen. *Curse of Dimensionality*, pages 314–315. Springer US, Boston, MA, 2017. URL: `https://doi.org/10.1007/978-1-4899-7687-1_192`, `doi:10.1007/978-1-4899-7687-1192`.

[140] M. Köppen. The curse of dimensionality. In *5th Online World Conference on Soft Computing in Industrial Applications (WSC5)*, volume 1, pages 4–8, 2000.

[141] J. Feng, H. Xu, S. Mannor, and S. Yan. Robust logistic regression and classification. In *Advances in neural information processing systems*, pages 253–261, 2014.

[142] D. Dua and C. Graff. UCI machine learning repository, 2017. URL: `http://archive.ics.uci.edu/ml`.

A Appendix

A.1 The Need For Activation Functions in DNN

In this section, we show that DNNs consisting solely of a sequence of linear layers have severely limited expressiveness: They are comparable to a one-layer linear DNN.

Consider a DNN with T linear layers $L_1, \ldots, L_t$ where $t \in 1, \ldots T$. These layers can be expressed as matrices $M_1, \ldots, M_t$ of dimensionality

$$M_t \in \mathbb{R}^{d_{in}^t, d_{out}^t}. \tag{A.1}$$

In order for the layers M_t to be applicable in sequence, the input dimensions have to match up with the output dimensions of the respective previous layers, i.e.

$$d_{out}^t = d_{in}^{t+1} \tag{A.2}$$

The sequential application of M_1 through M_t is equivalent to

$$M_t(M_{t-1}(...(M_1(x)))) = xM_1M_2...M_{t-1}M_t =: xZ \tag{A.3}$$

where Z is a real-valued matrix with dimensionality

$$\begin{aligned} dim(Z) &= dim(M_1M_2...M_{t-1}M_t) \\ &= (d_{in}^1, d_{out}^1) \times (d_{in}^2, d_{out}^2) \times \cdots \times (d_{in}^T, d_{out}^T) \\ &= (d_{in}^1, d_{in}^2) \times (d_{in}^2, d_{in}^3) \times \cdots \times (d_{in}^T, d_{out}^T) \\ &= (d_{in}^1, d_{out}^T) \end{aligned} \tag{A.4}$$

since $d_{out}^t = d_{in}^{t+1}$ as per equation (A.2). Thus, stacking linear layers always yields a linear model. This is why *activation functions* are used.

A.2 Identifying Mislabeled Instances

This section holds supplementary material for chapter 5.

Instance	Label
6093	crab
24900	cloud
33823	television
31377	camel
48760	motorcycle
31467	shark
45694	forest

Table A.1: Mislabeled instances in CIFAR-100 for $\alpha = 0.003$. This dataset seems to be labeled quite accurately, as we could identify only seven instances in the training set where the image's label and content would not conform with the content.

Instance	original label	suggested label
3415	pullover	coat
28264	dress	shirt
29599	pullover	top/shirt
37286	top/shirt	shirt
36049	shirt	dress
9059	shirt	dress OR coat
18188	shirt	dress OR coat
1600	dress	top/shirt
34381	shirt	dress
22750	top/shirt	shirt
39620	sneaker	ankle boot
50494	shirt	dress OR coat
38208	ankle boot	sneaker
53257	top/shirt	shirt
29487	shirt	dress
13026	shirt	dress
20544	shirt	dress
51464	top/shirt	shirt
28764	pullover	top/shirt
29154	shirt	dress
24804	top/shirt	shirt
28341	top/shirt	dress
46125	pullover	top/shirt
46259	dress	coat ?
25419	top/shirt	shirt
36325	shirt	?
29728	coat	?
43703	top/shirt	?
45536	pullover	top/shirt
3512	top/shirt	dress
22264	top/shirt	dress
4027	shirt	dress
33982	coat	top/shirt
17243	shirt	dress
34804	pullover	top/shirt
20701	pullover	dress
55829	dress	top/shirt
35505	dress	shirt
36061	dress	shirt
38722	top/shirt	shirt
33590	top/shirt	shirt
44903	top/shirt	dress
50013	shirt	dress
40513	pullover	top/shirt
46926	top/shirt	dress
21771	shirt	top/shirt
1074	shirt	top/shirt
42018	pullover	dress
42110	dress	pullover
51735	top/shirt	dress
45592	shirt	dress
11885	pullover	dress
27350	coat	top/shirt
35321	top/shirt	dress
34598	top/shirt	dress
49983	top/shirt	shirt
2843	shirt	top/shirt
20319	pullover	dress OR coat
29628	shirt	top/shirt
49736	pullover	top/shirt
2468	shirt	dress
21245	top/shirt	shirt
34063	pullover	top/shirt
25643	shirt	top/shirt

Table A.2: List of mislabeled instances in Fashion-MNIST for $\alpha = 0.3\%$. Among the 180 pictures returned by our tool set (i.e. $\alpha = 0.003$), we identify 64 mislabeled instances. Hence, more than 35,5% of the reviewed images indeed carry a wrong label.

A.3 Identifying Data Poisoning in Regression Learning

This section holds supplementary material for chapter 6.

A.3.1 Evaluation of the *StatP* Attack

In additions to the evaluation of our newly proposed poisoning attack (c.f. 6), we evaluate the performance of existing poisoning attacks on regression learning, and identify its shortcomings. As described in Section 6.4.2, the only black-box data poisoning attack on regression data found in literature is the *StatP* attack [7]. We implement this attack and evaluate it twice: First, we apply the attack on the Warfarin dataset and three non-linear models. Second, we evaluate all 26 datasets against both the *statP* and *Flip* attack (Figure A.3). We observe the following:

- ***StatP* does not work on nonlinear regressors.** For nonlinear regressors, *StatP* is ineffective. The MAE remains near constant when adding even significant amounts of poison samples (see Figure A.2). When analyzing the poison data created by *StatP*, we find an intuitive explanation for this: *StatP* pushes all data points 'to the corners', i.e. to the edge of the feasibility domain. While these data do conflict with the 'clean' data for linear regressors, nonlinear models can easily accommodate both poison and clean data, as long as the samples don't overlap in feature-space.

- ***Flip* outperforms *StatP***. When evaluating both attacks on all 26 datasets, averaging four linear and three non-linear models, we can confirm the above results: *Flip* consistently outperforms *StatP* (c.f. Figure A.3).

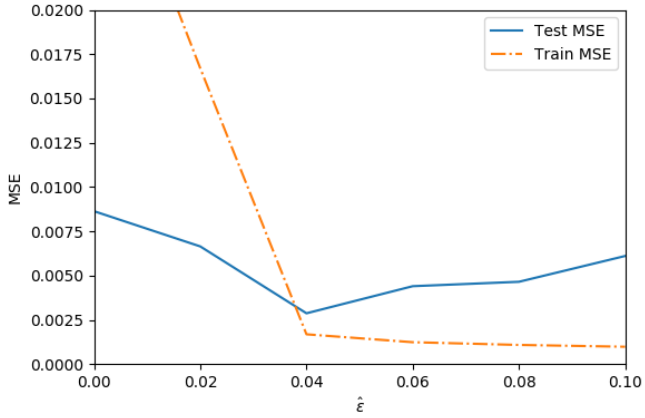


Figure A.1: Similarly to Figure 6.2, this plot shows the train and test error for varying degrees of $\hat{\epsilon}$. The model is a KernelRidge regressor, the dataset is the *loan* dataset, and the true poison rate is $\epsilon = 0.04$. We observe that the train loss gives clear indication as to when all poison samples are removed via *iTrim*: The lowest test loss (blue) coincides with the sharp discontinuity in training loss (orange).

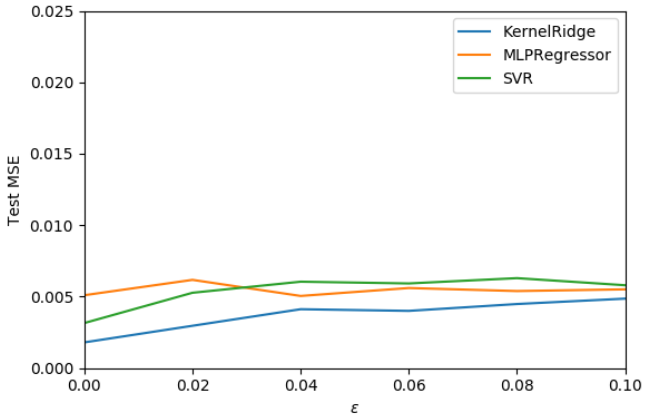


Figure A.2: Evaluation of the performance of the *StatP* attack [7] against nonlinear regressors. The *StatP* attack is applied to three nonlinear regressors, averaged over all 26 datasets. The x axis shows the degree of poisoning, while the y axis shows the test MSE. The attack is not effective. While these do disturb linear regressors, they are easily accommodated by nonlinear models without increasing any test loss.

Table A.3: The 26 datasets used for the empirical evaluation of [5].

	Name	features	n
0	ANACALT.dat	7.0	4052.0
1	accel	22.0	1732.0
2	ailerons.dat	40.0	13750.0
3	armesHousing	248.0	1460.0
4	availPwr	49.0	1802.0
5	bank8fm	8.0	4499.0
6	california.dat	8.0	20640.0
7	compactiv.dat	21.0	8192.0
8	concrete.dat	8.0	1030.0
9	cpu	12.0	8192.0
10	elevators.dat	18.0	16599.0
11	friedman.dat	5.0	1200.0
12	fuelCons	88.0	1764.0
13	heat	30.0	7400.0
14	house.dat	16.0	22784.0
15	loan	202.0	5000.0
16	mortgage.dat	15.0	1049.0
17	plastic.dat	2.0	1650.0
18	pole.dat	26.0	14998.0
19	quake.dat	3.0	2178.0
20	rings	10.0	4177.0
21	torque	95.0	1802.0
22	treasury.dat	15.0	1049.0
23	wankara.dat	9.0	1609.0
24	warfarin	177.0	5528.0
25	wizmir.dat	9.0	1461.0

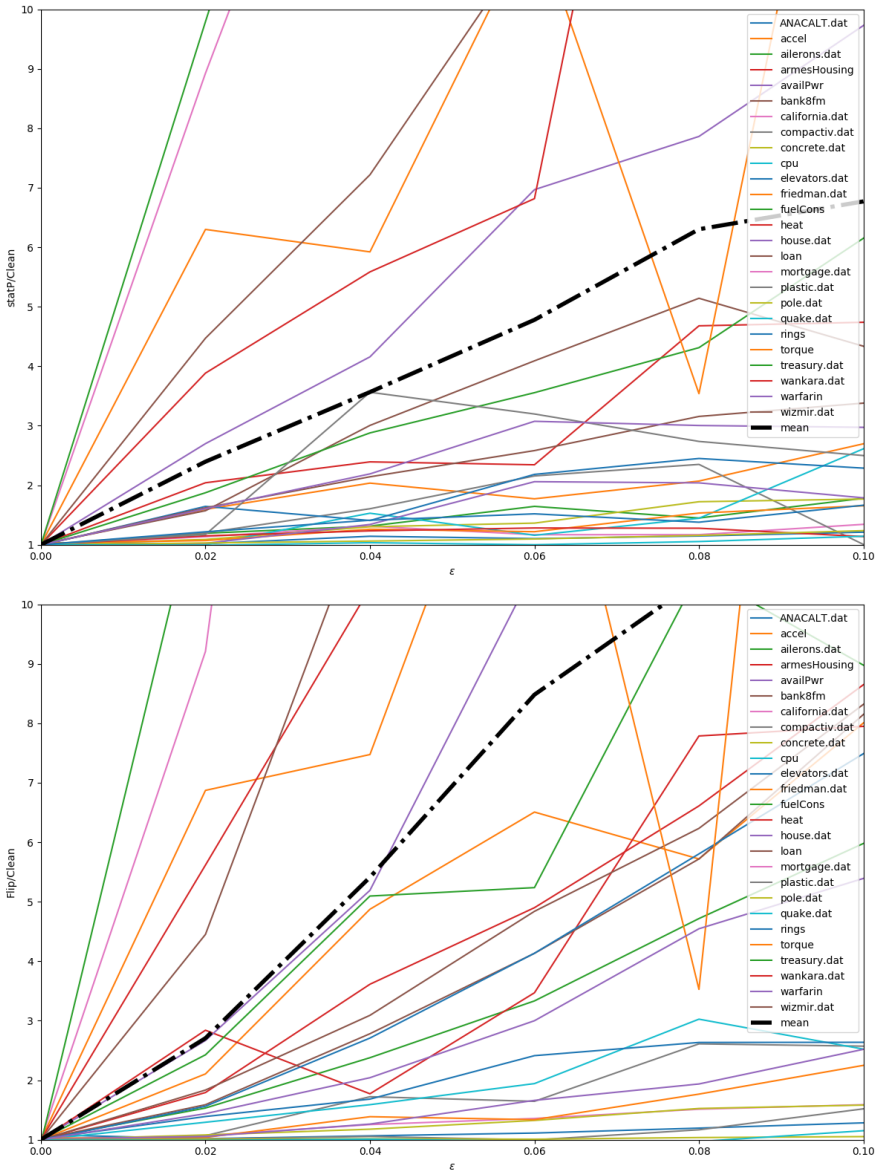


Figure A.3: Comparison between the *StatP* (top) and the *Flip* (bottom) data poisoning attack. This plot shows the increase in MAE when using a poisoned dataset instead of a clean dataset during model training. The results are averaged over all seven regressors, but displayed individually per dataset. For an average over all datasets, refer to Figure 6.3. *Top Image:* All 26 datasets when attacked with the *StatP* attack. While the degree of effectiveness varies between datasets, we generally observe a linear correlation between degree of poisoning and increase in test error. *Bottom Image:* The same datasets attacked with the *Flip*. Note that this attack consistently outperforms *StatP*, as seen on the higher MAE.

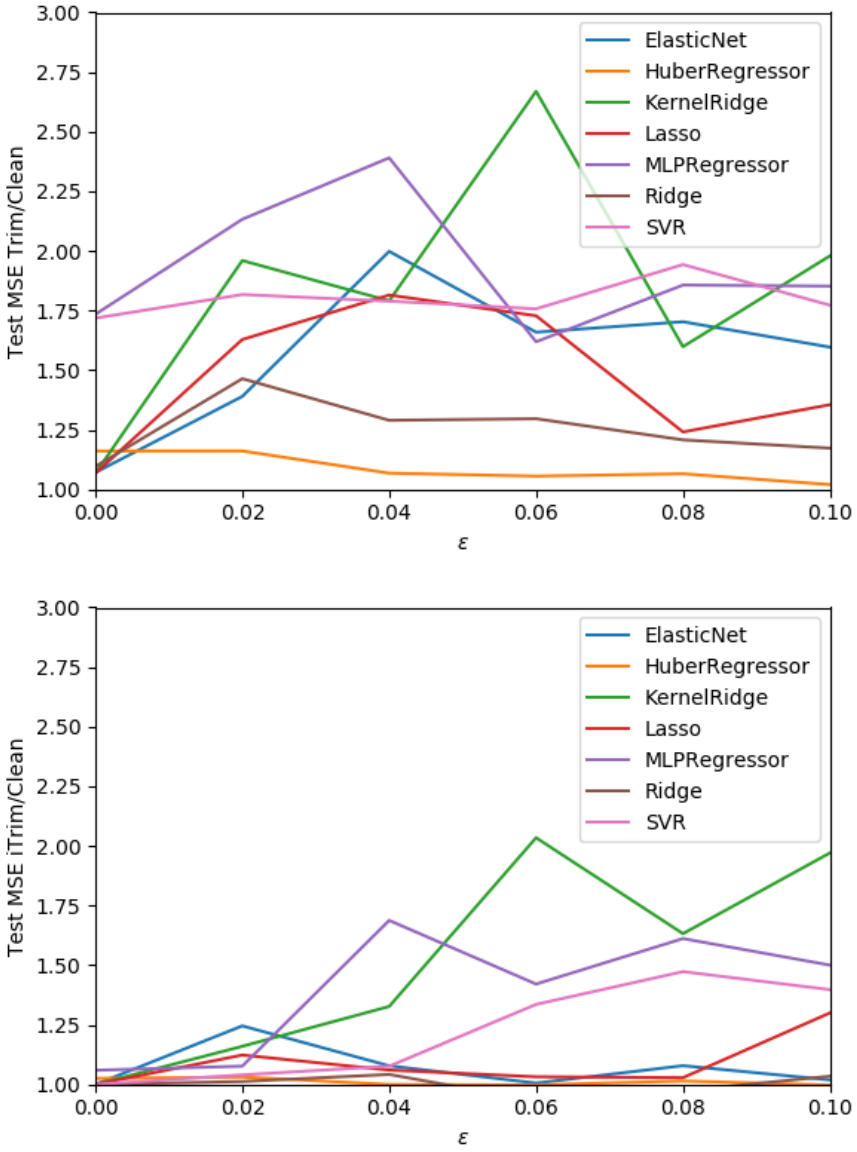


Figure A.4: Comparison between the *Trim* (top) vs. *iTrim* (bottom) data poisoning defense, averaged by dataset. *Top Image:* This plot shows the effectiveness of *Trim* when defending against the *Flip* attack, averaged over all 26 datasets. The datasets are poisoned as indicated by ϵ on the x axis. Then, the *Trim* defense is applied, the regressor is fitted to the dataset, and the resulting test MSE is compared against the test MSE on a clean, unpoisoned dataset. We see that, depending on the regressor, datasets cleaned with *Trim* still incur significant decrease in test performance. *Bottom Image:* The same process is applied to *iTrim*. We observe that test MSE is considerably decreased, especially for $\epsilon < 0.08$. The larger ϵ, the more similar the two defences become. Additional information is provided by Figure A.5.

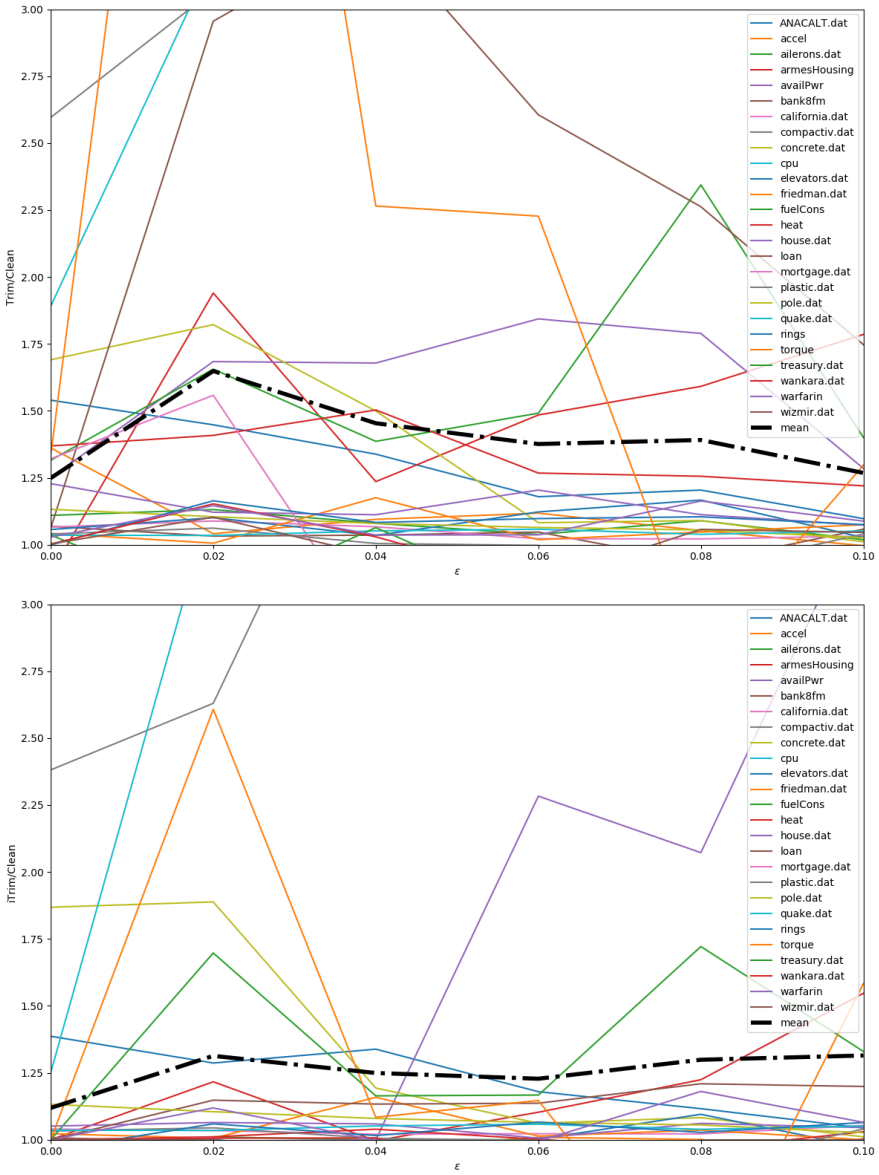


Figure A.5: *Trim* (top) vs. *iTrim* (bottom), averaged by regressor. Similar to Figure A.4, this plot shows the increase in MAE when using a poisoned and subsequently defended dataset instead of a clean dataset during model training. The results are averaged over all seven regressors, but displayed individually per dataset (as opposed to figure A.4, where the results are aggregated by regressor). *Top image:* Increase in MAE when defending against a *Flip* attack using *Trim*. *Bottom image:* Increase in MAE when defending against a *Flip* attack using *iTrim*. Notice that *iTrim* outperforms *Trim* almost consistently (as seen by the overall lower MAE, shown in bold black)

A.4 Identifying Data Poisoning in Classification Learning

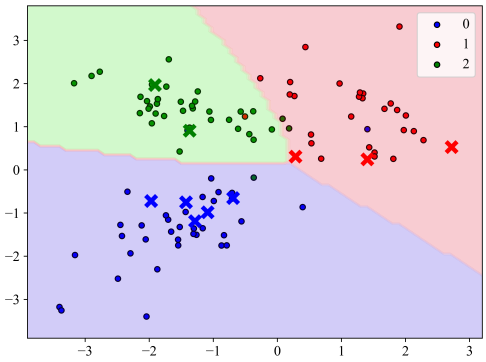
This section holds supplementary material for chapter 7.

dataset	$\hat{\nu}$	FP		FN		test loss		test accuracy	
		random	defence	random	defence	before	after	before	after
fashion_mnist_156	0.125	0.09	0.029	0.09	0.002	0.17	0.17	0.94	0.97
fashion_mnist_17	0.097	0.09	0.000	0.09	0.003	0.13	0.01	0.99	1.00
mnist_156	0.117	0.09	0.025	0.09	0.007	0.39	0.22	0.86	0.95
mnist_17	0.100	0.09	0.006	0.09	0.005	0.16	0.08	0.95	0.98
mean	**0.110**	**0.09**	**0.015**	**0.09**	**0.004**	**0.21**	**0.12**	**0.94**	**0.98**
std	**0.012**	**0.00**	**0.012**	**0.00**	**0.002**	**0.10**	**0.08**	**0.05**	**0.02**

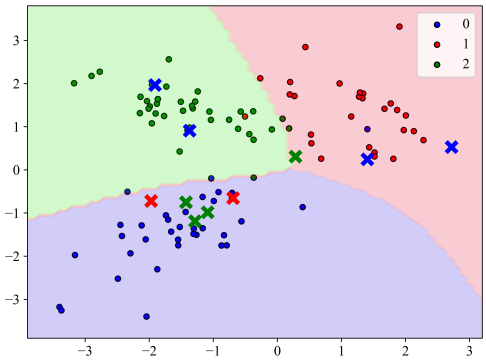
Table A.4: Results of our defence against the back-gradient optimization attack on a convolutional neural network, averaged over 5 runs. The datasets contain 10% poison samples each. The column $\hat{\nu}$ shows the poison rate estimated by the defence algorithm while the next two columns compare FPR and FNR of our defence against a random baseline.

dataset	ν	$\hat{\nu}$	FP		FN		test loss		test accuracy	
			random	defence	random	defence	before	after	before	after
breast_cancer	0.00	0.028	0.000	0.028	0.000	0.000	0.16	0.18	0.96	0.95
fashion_mnist_156	0.00	0.010	0.000	0.010	0.000	0.000	0.09	0.11	0.98	0.98
fashion_mnist_17	0.00	0.010	0.000	0.010	0.000	0.000	0.00	0.00	1.00	1.00
mnist_156	0.00	0.009	0.000	0.009	0.000	0.000	0.13	0.14	0.96	0.96
mnist_17	0.00	0.028	0.000	0.028	0.000	0.000	0.05	0.08	0.98	0.97
points	0.00	0.024	0.000	0.024	0.000	0.000	0.20	0.23	0.94	0.94
spambase	0.00	0.025	0.000	0.025	0.000	0.000	0.48	0.53	0.88	0.88
mean	0.00	0.019	0.000	0.019	0.000	0.000	0.16	0.18	0.96	0.95
std	0.00	0.017	0.000	0.017	0.000	0.000	0.16	0.17	0.04	0.04
breast_cancer	0.05	0.071	0.048	0.027	0.057	0.004	0.30	0.25	0.91	0.95
fashion_mnist_156	0.05	0.066	0.048	0.022	0.048	0.004	0.19	0.15	0.95	0.97
fashion_mnist_17	0.05	0.046	0.048	0.000	0.057	0.002	0.07	0.00	0.97	1.00
mnist_156	0.05	0.053	0.047	0.014	0.047	0.009	0.36	0.18	0.91	0.95
mnist_17	0.05	0.057	0.046	0.012	0.055	0.002	0.33	0.06	0.91	0.98
points	0.05	0.037	0.048	0.006	0.057	0.017	0.22	0.24	0.93	0.94
spambase	0.05	0.069	0.050	0.043	0.050	0.021	0.68	0.59	0.84	0.88
mean	0.05	0.057	0.048	0.018	0.053	0.008	0.31	0.21	0.92	0.95
std	0.00	0.021	0.003	0.019	0.005	0.010	0.19	0.19	0.04	0.04

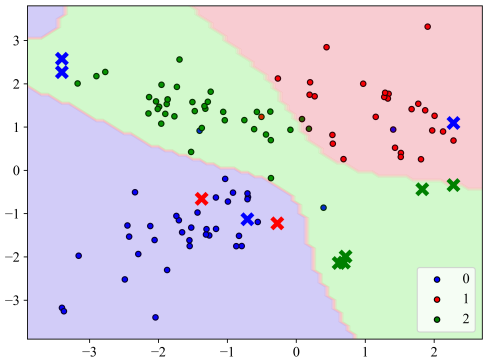
Table A.5: Results of our defence against the back-gradient optimization attack on a neural network with one inner layer, averaged over 5 runs. The datasets contain 0% or 5% poison samples each. The column $\hat{\nu}$ shows the poison rate estimated by the defence algorithm while the next two columns compare FPR and FNR of our defence against a random baseline. Test loss and accuracy remain consistent in the case of no poisoning or improve considerably when using the defence on the poisoned datasets.



Before poisoning attack



After label flipping



After back-gradient attack

Figure A.6: The effect of introducing 11 poison instances into a *points* classification dataset. The top figure shows the unpoisoned dataset, where the bold crosses indicate the data instances to be changed during the upcoming attacks. The middle figure shows the small change of the decision surface when the flip attack is applied. The bottom figure shows the considerably altered decision surface after the back-gradient attack. Note that not only the labels, but also the feature values of the poison data have changed.