TOWARDS TRUSTWORTHY APPLICATION OF MACHINE LEARNING: TWO CASE STUDIES

by

RUOYAN CAI

(Under the Direction of Kyu Hyung Lee & Kang Li)

ABSTRACT

With the widespread usage of machine learning (**ML**), the trustworthy application of ML has gained increasing attention. An ML model integrated into a system is playing a critical decision-making role. Simultaneously, the concerns of security and trustworthiness have been raised by using machine learning techniques in practice. In this dissertation, we identify two research challenges for building secure and trustworthy machine learning applications and propose solutions to address them. Initially, most ML techniques are designed with assumptions that data is correct and running environments are benign. However, these assumptions might not guarantee in the wild. It is challenging to build a trustworthy ML-based system due to the characteristics of ML models, non-stationary run-time environments, and the large surface of attacks along the ML lifecycle.

The first problem is contamination data that causes the counterfactual prediction of an ML model. Data collected from various sources and preprocessed by many processors can be contaminated by equipment malfunction, human errors, or even malicious attacks. We propose an Automated Contaminated Attribute Localization (ACAL) system to pinpoint the faulty attribute in a contaminated data. ACAL quantifies each attribute's suspicious-

ness automatically, which helps users trace the root cause of the data contamination. Our evaluation of ACAL on real-world datasets and distinct models shows that ACAL can reach over 91% accuracy of localization.

The second problem is a violation of model integrity when users outsource an ML model to a cloud. The ML model has at the risk of being modified by dishonest cloud providers and attackers, which leads to incorrect predictions. We propose an Integrity Checking for Neural Network approach to detect the model modification as a normal query. A novel way to generate querying samples is presented. We evaluate our approach on different cases of model modification and validate its effectiveness.

In this dissertation, we argued that estimating ML application risks and preparing for ML task failure is crucial for the development of reliable ML-based applications. We considered ML risks from different perspectives and proposed solutions towards trustworthy applications of ML in two scenarios: contamination data and model integrity violation.

Index words: Machine Learning, Trustworthy Application, Error Diagnosis, Integrity Checking

TOWARDS TRUSTWORTHY APPLICATION OF MACHINE LEARNING: TWO CASE STUDIES

by

Ruoyan Cai

B.S., Nanjing University of Aeronautics and Astronautics,China, 2015

A Dissertation Submitted to the Graduate Faculty of the

University of Georgia in Partial Fulfillment of the Requirements for the Degree.

Doctor of Philosophy

Athens, Georgia

2020

©2020

Ruoyan Cai

All Rights Reserved

TOWARDS TRUSTWORTHY APPLICATION OF MACHINE LEARNING: TWO CASE STUDIES

by

Ruoyan Cai

Major Professor: Kyu Hyung Lee

Co-Major Professor: Kang Li

Committee: Jaewoo Lee
Wenwen Wang

Electronic Version Approved:

Ron Walcott
Interim Dean of the Graduate School
The University of Georgia
August 2020

Acknowledgments

To all those who have supported me through this journey,

I am sincerely thankful to all members of my advisory committee, who have enlightened me during the Ph.D. study. In particular, I am ineffably indebted to my major professors Dr. Kang Li and Dr. Kyu H. Lee, for their conscientious guidance, expert suggestions, kindness, encouragement, and all other considerations throughout the research work and the preparation of this dissertation.

I want to thank my husband, Shibo, who has given me love, support, and courage to overcome difficulties. I also acknowledge my gratitude towards my parents and family members for their unfailing emotional support. I feel grateful to have so many people who love and support me.

Finally, I thank my fellows from the NSS Lab and SSCF Lab, especially Guodong Zhu, Yibin Liao, Yue Yin, Xingzi Yuan, Omid Setayeshfar, Karthika Subramani, and An Chen. I had a pleasant time working with them.

Any omission in this brief acknowledgment does not mean a lack of gratitude.

Contents

Acknowledgments **iv**

List of Figures **vi**

List of Tables **viii**

1 INTRODUCTION **1**

1.1 Problem Scope and Challenge 2

1.2 Dissertation Statement and Contributions 5

1.3 Background 7

1.4 Organization of the Dissertation 11

2 RISKS ALONG MACHINE LEARNING LIFECYCLE **13**

2.1 Lifecycle of Machine Learning Application 13

2.2 Risk Analysis 17

2.3 Evidence Availability and Challenge of Evidence Usage 23

3 AUTOMATED CONTAMINATED ATTRIBUTE LOCALIZATION **26**

3.1 Introduction 26

3.2 Motivating Example 29

3.3 Problem Definition 33

3.4 Methodology 35
3.5 Evaluation 38
3.6 Discussion 52
3.7 Related Work 53

4 INTEGRITY CHECKING FOR NEURAL NETWORKS 56

4.1 Problem Statement 56
4.2 Methodology 60
4.3 Experiment Setting 65
4.4 Evaluation 68
4.5 Discussion 74
4.6 Related Work 75

5 CONCLUSION AND FUTURE WORK 77

5.1 Dissertation Conclusion 77
5.2 Future Work 78

Bibliography 80

List of Figures

1.1 Supervised Machine Learning 10

2.1 Pipeline of Machine Learning Application 14

3.1 Prediction Explanation by SHAP. The sample instance contains a contaminated attribute " READ_HISTORY_BOOKMARKS" = 0 (Not Used) and the prediction class of the sample is "Benign" with 1.0 confidence. 31

3.2 High-Level Workflow of ACAL. 32

3.3 The example of contaminated data generation 41

3.4 Case Study Example: Phishing Website with Random Forest 47

3.5 Top-K Rate of Different Models on Phishing Website Dataset with different Hyper-parameter λ 50

3.6 DBScan Outlier Detection on Contaminated Data 52

4.1 Cifar Examples 67

4.2 GTSRB Examples 67

4.3 VGG-Face Examples 68

4.4 Detection Rate of Backdoor Model over Different Number K 70

4.5 Detection Rate of the Compressed Model with Different Compression Rates over Different Number K 72

4.6 Detection Rate of Weights Changed Model with Ratio=0.1 over Different K . 73

4.7 Detection on Model with Different Ratio of Weight Changed (K=1) 74

List of Tables

3.1 Dataset used in the Evaluation 39

3.2 Evaluation Result 45

3.3 Average Accuracy of Top-1, Top-2, and Top-3 rates 46

3.4 The effect of the size of labeled dataset 48

3.5 The effect of the different Hyper-parameter λ 49

4.1 Model Information 70

4.2 Calculating Detection Rate by Comparing Output Under Different Format Settings When K=1 71

4.3 Methods Comparison of Missing rates(%) w.r.t to K on three types of modification 73

Chapter 1

INTRODUCTION

We are in the era of artificial intelligence and the application of machine learning is universal. With the convergence of algorithm advances, data proliferation, and tremendous increase in computing power and storage, Machine Learning (**ML**) becomes a powerful tool to learn knowledge from a large amount of data and to effectively perform a prediction or classification task. It relieves people from repetitive work and assists us to improve the performance of a system on a certain task. Remarkable success has been seen in many applications of ML. There is a belief that the current widespread usage of ML is just the beginning of an ML-enabled technological evolution and AI will permeate every part of our lives in the future.

While ML is being pervasively deployed in the widespread development of software-based inference and decision making, it raises growing concerns, such as the capability to explain its reasoning and decision-making, its reliability to make critical decisions in a field that is directly related to human safety, to be aware of threats and attacks that occur in development or deployment phases. Specifically, the ML model usually acts as a box that lacks transparency and straightforward interpretation. It tends to raise errors silently by providing incorrect predictions with high confidence. ML error diagnosis is challenging because many components are involved in the lifecycle where the processes are stacked in

the development phase but the deployed model is apart afterward. On the other hand, most machine learning techniques are originally designed with assumptions that data is stationary and running environments are benign. However, when models are deployed in the wild, these assumptions might not hold.

These concerns become obstacles that cause users' hesitation in adopting ML techniques. It is important not only to improve the performance of ML to reach desirable accuracy but also to understand its characteristics for deeper explanation and vulnerabilities for building defense mechanisms.

1.1 Problem Scope and Challenge

When it comes to implementing ML models in real-world conditions, the scenario becomes complicated and sometimes uncontrolled. For example, accuracy metrics are used to evaluate models with a set of validation samples and users are confident with the performance and decide to deploy it according to the evaluated metrics. However, the prediction will be inaccurate if real-world data is anomalous or significantly different from that used in training. The complication of real-world settings poses new challenges to practitioners such as how to estimate faithfulness of the model, how to increase the trustworthiness of an individual prediction, how to diagnose problems.

Since ML model plays an important role as a decision-maker in mission-critical back-end problems such as fraud detection, face authentication, and commendation system, a mistake might cause loses or embarrassment, especially impacts on human loss of health and life. On the one hand, unprecedented efforts have been made to improve ML models' accuracy which has became the primary measure of trust. While practitioners aim at 99.99% accuracy, the rest 0.01% failure is usually being ignored. On the other hand, the risk is likely to reside in any stage along the ML pipeline and it is hard to anticipate failure. It is worth noting

that any violation of operation standards will lead to a high degree of uncertainty of model behaviors.

This dissertation focus on addressing the challenge of user trust of ML by detecting the trace of violation that affects the deployed model's decision-making. We research risks existing that affect model decision under two scenarios where data correctness is not guaranteed, and model safety is not a promise. In the first scenario, the violation of correct data as input results in counterfactual predictions in classification tasks. We aim to find out the trace of violation source. In the second one, we focus on detecting the violation of model integrity, especially for compromised models. More details of the problem statement and challenge are explained as follows.

Study Case 1: Data Contamination Cause Counterfactual Result

When an ML-based solution is deployed to solve real-world problems, it often requires multiple steps to collect and prepare the data to compose a question that the ML solution can answer. If the part of data is contaminated by software bugs, hardware faults, human error, or malicious attackers during data collection or preparation, the question itself can be affected, and thus the answer from the ML cannot be trusted. We call such erroneous data located in the question as *contaminated data* and attribute containing the erroneous value as *contaminated attribute*.

Existing data cleaning or outlier detection techniques mainly aim to detect or mitigate input error during the model training phase to maximize decision accuracy. In practice, such techniques are not effective in detecting input error if the contaminated attribute-value is still in the valid range, which allows a contaminated input to enter the downstream of the system. It is highly likely to cause a counterfactual result. When it happens, it is critical to identify the exact input data value that has been contaminated to trace back the root cause of the problem.

Unfortunately, the problem of counterfactual prediction caused by contaminated data is unexpected and it is hard to be handled with existing error diagnosis techniques. Unlike the debugging process for traditional software to identify the root cause of data errors by inspecting data flow, ML model is difficult to be inspected due to its characteristics of low transparency and interpretability. In addition, the data collection and preprocessing stages are often decoupled from the ML pipeline, and the data delivered to the ML pipeline is typically in a raw format without semantic information, which makes the diagnosis of data error more challenging.

In this scenario, the classification model is assumed to be benign and well estimated. The problem begins with erroneous data containing one contaminated attribute and its model prediction result is known to be counterfactual. The problem we aim to solve is to automatically pinpoint the contaminated attribute.

Study Case 2: Model Contamination Affects Integrity

With the development of AI platforms that enable clients to train and deploy ML models in powerful cloud infrastructures, there are many concerns regarding Machine Learning as a Service (MLaaS). Service correctness is one of the most essential assurances. The client needs to be ensured that the cloud provider performs the task correctly on behalf of the client. Specifically, the ML model must work as intended as if it were done locally and has consistent behaviors as it was tested. Without this assurance, an incorrect prediction could be returned by the ML service.

We consider a scenario where an ML model, specifically a deep neural network (DNN), is outsourced to a cloud as a service but it has been modified to some extent due to malicious attacks or dishonest cloud providers. For example, a malicious third-party hosts an attack to deflect the normal behavior of the model, or the cloud provider itself may cheat the user with a lossy compressed model to reduce resource cost. We define the violation of model integrity

as that the parameter of the outsourced model has been changed and the performance of the model is impaired.

In this scenario, the client wants to verify the integrity of the model running on the cloud and assure that it is the same one as the original model. However, it is challenging to implement the checking of the remote model integrity. Once the model deployed on the cloud, the control over the remote model is non-transparent from the client-side. It is regarded as a black-box and can only be queried by the cloud provided API, which limits useful information applied to integrity checking. Moreover, existing attacks on a model can be unnoticeable. For example, the modified model remains similar performance on normal samples while misclassification is triggered by tampered samples where the trigger is only known by the attacker. In addition, once there exist undetected attack channels, the remote model can be modified in a manner of constant change.

The problem is to check the integrity of the remote model without conspicuous operations that might raise cloud provider attention. Noted that integrity violation is assumed to affect the model itself. Incorrect behaviors due to adversarial inputs or attacks on execution workflow are out of the problem scope.

1.2 Dissertation Statement and Contributions

Dissertation Statement

Estimating risks for ML solution and preparing for task failure is crucial for the development of trustworthy ML-based applications. The reliability of ML models can be enhanced by 1) automatic error diagnosis system for data contamination, which helps users troubleshooting the error cause; 2) integrity checking for outsourced models, which helps users verify model performance and safety status.

Contributions

We considered ML risks from different perspectives. Extensible approaches are proposed to help ML practitioners handle specific situations where ML performance is not as expected. Experiments demonstrated the usefulness of our approaches. Concrete contributions are listed as follows:

- Our first contribution is an analysis of risks along the ML lifecycle. Specifically, we analyze reasons and vulnerabilities that could lead to wrong prediction and discuss the challenges of solving security issues along different stages. One key goal is enhancing the user trust of ML for better deploying the ML model as a service. We find that model checking and error diagnosis are important to help users examine error, and enhance trust to extend ML techniques to safety-related areas. Consider the inference phase where ML models treated as a black box, we apply these insights in our subsequent research to develop model checking and error diagnosis methods.

- Our second contribution introduces techniques to handle error occurrence during data preparation phase when partial data has been contaminated by software bugs, hardware faults, or human error. We demonstrate that existing techniques, such as data cleaning or outlier detection, cannot effectively identify contaminated attributes if the values stay in the valid range. In order to assist analysts with the examination of the counterfactual results, an Automated Contaminated Attribute Localization (ACAL) system is proposed to pinpoint the faulty attribute in a contaminated data, which allows the user to trace the root cause of the data contamination. As we address the problem of explaining an individual prediction where the input of the model is susceptible to incorrectness, other ML explanation tools can be complementary to ACAL in terms of explaining model prediction of data that is assumed to be correct. ACAL is an ML model-agnostic and does not require any prior knowledge about the target model

or attribute. ACAL automatically quantifies the suspiciousness of each attribute in the input of the ML model. Our evaluation of a collection of ten real-world datasets on five distinct models shows that ACAL can accurately pinpoint the contaminated attribute with average 91.4% accuracy.

- Our third contribution introduces techniques to handle integrity violations of the model that is deployed to the could. We explain several situations where integrity violation could occur and exhibit the demand for model integrity checking under cloud service. A novel approach is presented to detect violation of model integrity for a DNN from the client-side. We use a new method to generate querying samples for an original model. It utilizes DNN's incapability of interpreting unseen features to construct perturbation to trigger uncertainty of the model prediction. The prediction uncertainty is defined, and finding a querying sample boils down to an optimization problem where the querying sample maximizes the prediction uncertainty of the model. Then, querying samples are used to query the remote model with black-box access mode. The query results are compared to the original model's prediction outcomes to detect whether the model has been modified. The process of detection is simple, and it acts like a normal query activity. We evaluate the effectiveness of our approach to different types of model modification with two real-world datasets. Experiment results show that it can detect a model integrity breach with high accuracy and low cost.

1.3 Background

1.3.1 Importance of Machine Learning Role in Applications

ML automates analysis and interpretation of complex data and it can be the key to unlock the value of big data. The application of ML is more ubiquitous than you might think. More and more services and solutions embrace ML techniques to make decision with minimum human

intervention. Below are examples of machine learning role in daily-life-related applications across various domains.

- Cybersecurity Defense

  Machine learning techniques are widely leveraged in cybersecurity defense. In addition to a traditional rule-based firewall, the ML traffic analysis system allows businesses and organizations to find threats and malicious activity in encrypted traffic without decryption (e.g. Cisco ETA). It automatically detects anomalies and block threats as needs while keeping administrative overhead and manual work to a minimum.

- Health Care

  Many medical diagnosis are seeking ML-based solutions for characterizing, monitoring, and intervening on human health. With assistance of ML techniques, a large scale operation of health examination are expected to perform safety and potentially without expert supervision[1]. For example, ML-based health system helps diabetic patients regulate their blood sugar. It automates the prescription refills. It assigns a doctor most qualified to health center customers. Such health issues can be quickly identified by the diagnosis system, which makes use of appropriate medications as early treatment possible. It not only improves an individual patient's care experience but also helps define and profile population, simplifying the population analysis.

- Natural Language Processing

  Natural Language Processing(**NLP**) has become an essential tool for translating words and sentences in spoken language into a machine-readable format. It helps human-to-machine communication and scales other language-related tasks. For example, the speech recognition system free people from hand-controlled manipulation in daily life such as audio map navigation and intelligent home-device control. A model can handle large volumes of textual classification tasks by understanding particular content and

categorizing it into a class, such as sentiment analysis of online shopping reviews. Human language is notably complex and diverse, not only are there over six thousands of spoken languages, but each language has its own grammar and syntax rules. NLP helps resolve obscurity in language and structure highly unstructured data for many downstream applications.

- Computer Vision

  Human is talent at processing images into thoughts. When it comes to using the computer to process images in the same way the human brain does, the task has been one of the most difficult to teach computers. Thanks to advances in innovation in deep learning and neural networks, computer vision(**CV**) has been seen astonishing accomplishments in some tasks related to detecting objects[2] and labeling [3]. For example, CV enables self-driving cars to make sense of surroundings, enables match of human faces to their identities for authentication and verification, enables mobile devices to be equipped with augmented and mixed reality technology.

1.3.2 Supervised Machine Learning

Supervised machine learning are widely used for categorical class prediction. In this dissertation, ML application in classification tasks are used for problem illustration and experiment evaluation. Noted that models mentioned in the dissertation are all trained in supervised mode. We introduce the concept of supervised ML in this section.

Supervised learning is a machine learning process of learning a function from a dataset consisting of input-output pairs. It assumes that training instances are generated by a fixed unknown target function over instance space. As the target function is unknown, it defines a hypothesis space where the learning algorithm explores a mapping from input to output. The learned mapping also referred to as a model, describes the relationship between input

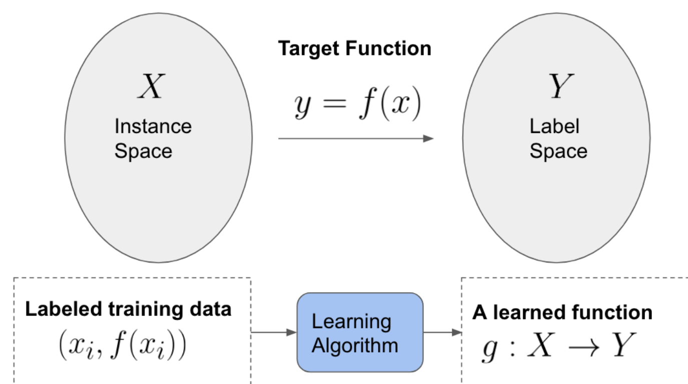


Figure 1.1: Supervised Machine Learning

and output, which can be used to predict never-before-seen data.

Specifically as shown in Figure 1.1, X is instance space of a set of examples and Y is label space of categories of examples. Typically, a machine learning algorithm consists of two main phases: training and testing. The existing set of examples split into two dataset: training set and testing set for training phase and for testing phase, respectively. During training phase, a learning algorithm uses a set of labeled training data $(x_1, f(x_1)), ..., (x_N, f(x_N))$ to learn the mapping relationship from X to Y, denoted as a learned function $y = g(x)$, which is also known as a model. Further in testing phase, examples from testing set are fed to the learned function and then compare the results g(x) to the corresponding ground truth labels. The goal of learning algorithm in the training phase is to find a good approximation for the target function and the testing phase evaluates how successful the algorithm learned a model.

The preparation for using supervised machine learning is listed as the following.

- **Instance Space.** An user needs to structure the input to the problem and define the features of the input as representation.

- **Label Space.** The label determines the type of task such as $f(x) \in \{0, 1\}$ for binary classification, $f(x) \in \{0, 1, 2, .., K\}$ for multi-class classification, and $f(x) \in R$ for

regression.

- **Hypothesis Space.** The set of all the possible legal hypothesis for learning algorithm to explore.

- **Learning Algorithms.** An appropriate machine learning algorithm is selected to learn from labeled examples. The selection is usually depends on empirical experience. More details of learning algorithms are described in the following subsections.

- **Evaluation Metrics.** An user needs to configure a metric to measure the performance of the learning algorithm. The learning procedure is regarded as an optimization problem. The object function, also referred to as a loss function or cost function, is used to calculate the value we want to minimize or maximize.

Implementation

Approaches used in this dissertation include Logistic Regression, Naive Bayes, Decision Tree, Random Forest, Multilayer perceptron, Deep Neural Networks. Implementation of model training and testing follows standard ML processes. Details will be introduced in later related chapters.

1.4 Organization of the Dissertation

We split each contribution into a separate chapter. Each chapter is an independent unit with its own evaluation and related work. The remained of the dissertation is organized as follows:

Chapter 2: Risk Analysis along Machine Learning Lifecycle - We review the lifecycle of ML, including components and processes. To begin to understand the concerns and security risks that will affect ML performance, we introduce components and processes along

the ML lifecycle. Then, we analyze issues for practice problems and real-world setting complications at each stage. We explain existing attacks and how it affects the model. Furthermore, we identify evidence that can be preserved for error analysis and discuss open challenges of applying data lineage for the forensics.

Chapter 3: ACAL: Automated Contaminated Attribute Localization - We make a scenario for analysis of data error that leads to counterfactual results. We propose the method used for investigating data contamination and showed evaluation results on five distinct models.

Chapter4: Integrity Checking for Neural Network Models - We make a scenario for analysis of model integrity violation. We propose the method used for checking model integrity under the cloud environment and run experiments on multiple types of model modification to detect the violation.

Chapter 5: Conclusion and Future Work - This chapter presents the conclusions, implications drawn from the results, and future work.

Chapter 2

RISKS ALONG MACHINE LEARNING LIFECYCLE

In this chapter, we introduce some of the fundamental stages of developing and deploying machine learning applications. Then, we analyze the vulnerabilities at each stage and existing attacks. Moreover, we explain what evidence can be preserved for ensuring ML safety and discuss open challenges.

2.1 Lifecycle of Machine Learning Application

We illustrate the general ML lifecycle that consists of four stages shown as a pipeline in Figure 2.1. The lifecycle starts with *Data Preparation*, where data are acquired and preprocessed to a standard format as input to the next stage. The next stage, *Model Development*, comprises model training and testing. Once the trained model meets all the required criteria, it is ready to assemble into a system. The third stage is *Launch*, referred to as model deployment, where the trained model is launched to a platform, running as a part of an application. The last stage, *Maintenance*, comprises activities concerned with the deployed model, such as performance monitor and regular checking. More description of each

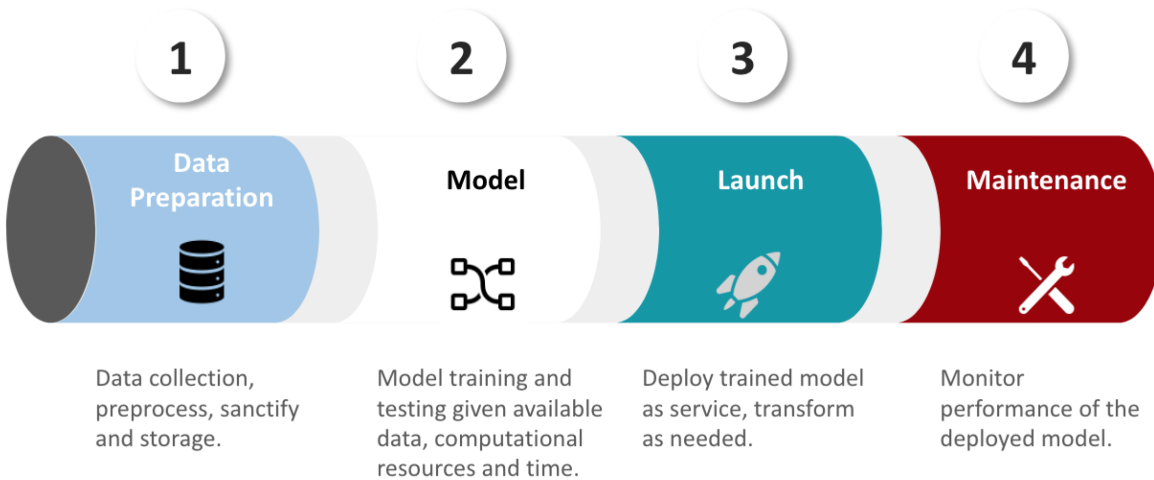


Figure 2.1: Pipeline of Machine Learning Application

stage is introduced below.

2.1.1 Stages and Tasks

Data Preparation

The lifecycle of ML starts with the data preparation stage, consisting of data collection, data augmentation if needed, and data preprocessing. It is a fundamental stage because data is at the core of any application of ML. A set of data can be originated in the real-world such as a collection of real-time sensor data. Synthetic data can be generated for exploring the specific expression of an ML algorithm's behavior [4]. In the context of supervised learning, the primary outcome of data collection is to prepare pairs of features and labels. ML algorithms can further learn a mapping relationship from these pairs. Most ML algorithms require a large amount of data to obtain useful insights. However, acquiring labeled data is challenging for scenarios where sample labeling requires cumbersome human labor. To ameliorate this problem, data augmentation is applied to create more data with limited labeled data such as GANs techniques for image augmentation [5, 6]. Additionally, the quality of input to a

model decides how accurate the model can be, and raw data is hardly directly fed to the ML model as input. Thus, we need data preprocessing step where raw data are transformed into a specific format, cleaned to consistent values, and extracted as needed features—preprocessing benefit the ML model to learn valuable information effectively.

Model Development

Model development consists of three main steps: model selection, model learning, and model testing. *Model selection* is a process of selecting an appropriate ML algorithm to model data. ML engineers usually take a significant amount of time to conduct model selection experiments, which are repeated trials of training and tuning models to achieve desirable outcomes. Typically, model selection is undertaken regarding the problem design and the volume and characteristics of the dataset. Moreover, since ML is an empirical science, the engineers mostly conduct this experimentation with their ad-hoc style and experience [7, 8]. The next step is *Model learning*, where the ML algorithm aims to minimize a cost value to learn parameters of the model from the training dataset. Specifically, in the context of supervised machine learning, a loss function is used to indicate how good the model prediction is on an individual sample. The model learns a set of weights and biases that can have low loss value. In the meantime, a set of validation data from the same distribution as training data is used for estimating test errors during the learning procedure, which helps ML engineers obtain comparable performance characteristics (e.g., accuracy, overfitting, sensitivity, specificity) for parameters tuning. Finally, a fully-specified model is tested on a testing dataset to evaluate the model's performance on unseen data. It is responsible for ML engineers to ensure the final-state model meets a given specification, such as satisfactory performance and reliability requirements.

Model Deployment

Assuming that the final-state model has met all requirements, the model is incorporated into a system in the Model Deployment stage of the ML lifecycle. With the emerging popularity of AI and the development of cloud computing, many platforms (e,g. Microsoft Azure ML, Google Cloud AI platform, Amazon SageMaker, BigML) make it easy for ML practitioners to deploy their ML models into the production. They provide reproducible ML pipelines, automated integration, easy-accessible prediction API, model version control, powerful computation resources, etc. Meanwhile, as the widespread usage of IoT devices, deploying ML models into mobile user-ends has been an increasing trend. For example, Qualcomm QCS400 series of audio System-on-Chips support the voice UI enabled smart home and audio ecosystem. Additionally, mobile-edge deployment requires certain techniques of model compression and optimization due to resource constraint [9]. The outcome of the Model Deployment stage is to make trained models available in production environments, where they can provide predictions as a service.

Maintenance

Model deployment is not a thing that was set up once and for all. In fact, ML models are difficult to maintain in production settings [10]. The task of the *Maintenance* stage is to sustain model performance over time and update the model with minimum time and effort. The deployed model is monitored and checked with automated tests. If it does not fall into a range of performance expectations, the model needs an update or an examination if errors occur. Besides the model monitor, new data inputs need to be monitored to ensure they are within the same distribution as the training dataset.

2.2 Risk Analysis

Considering risks along the lifecycle, we firstly analyze existing issues for practice problems and real-world setting complications at each stage. Then, the discussion of existing attacks proceeds along with two phases of model: Training Phase and Inference Phase. We identify the adversarial goals and means and then explain how it influences model training and inference.

2.2.1 Existing Issues at Each Stage

Issues at the Stage of Data Preparation

Many issues can complicate tasks at the stage of data preparation. It is challenging to maintain consistent quality of data that is collected from various sources and transformed by multiple preprocessors. Data quality problems are raised, such as missing values, integrity constraints violation, unstructured data, and outliers. On the other hand, it is common to reuse pre-existing datasets when the required data is not sufficiently available. Obtaining pre-existing data from third parties raises the data integrity problem during transit. Moreover, using data augmentation to get more samples poses challenges of verifying data completeness and relevancy.

In practice, it is likely to leak sensitive information (e.g., feature leakage, training examples leakage) if input data are not characterized with consideration of privacy protection. An instance of leakage would be a model that uses the target itself as an input feature. For example, "account number" is used as one of the features to predict whether a potential client would open an account at a bank, but "account number" can only be known after an account has been opened [11]. In fact, feature engineering heavily depends on leveraging human intuition to extract predictive features in the complex data, further determining the

success of model learning. It is also a bottleneck in the data preparation stage because both domain expertise and technical experience are required [12].

Issues at the Stage of Model Development

The first step, *model selection*, involves the design of experiments. Considering high-impact decision-making of an ML model, society needs it to be easily understandable, and humans can track output back to the origins, referred to as model interpretation. As known, the capabilities of interpretation varies on different ML models. Sophisticated approaches that provide good prediction accuracy are more challenging to interpret. It becomes a trade-off between accuracy and interpretation capability when practitioners choose a model from a set of eligible ML models. While pursuing high accuracy, the practitioner is at risk of low interpretation and low transparency for ML decision-making. Otherwise, acceptance with lower accuracy means more tolerance with failure.

Although traditional learning algorithms have been well understood, we do have a limited understanding of how training nets actually work in deep learning. There are several technical challenges to understand and analyze deep neural networks due to their characteristics. A deep neural network (DNN) has a complex architecture. A DNN model may consist of tens or hundreds of layers(depth), thousands of neurons(width) in each layer, as well as millions of connections between neurons. Besides, many functional components whose values and roles are not well understood either as individuals or as a whole [13]. Neurons play a fundamental role in a DNN model, and the learning procedure is hardly tracking. It is challenging to endow neurons with human-interpretable concepts, which may be the area as an attack target.

In the testing procedure, model performance mismatch usually happens. Estimation scores of model performance are different between training dataset and testing dataset. There are many possible causes for the model performance mismatch, such as model overfitting the

training data, unrepresentative data sample from the domain, and stochastic nature of the ML algorithm [14]. Moreover, testing does not provide security guarantees because attacks can craft inputs that are different from testing data to manipulate the model's behavior. It poses challenges to the validation of a trained model's performance and how to choose a final-state model. It is also challenging to ensure existing models free from faults.

Issues at the Stage of Model Deployment

ML model deployment is not a self-contained solution. The final-state model is either integrated into an application or embedded into a system. One of the easiest ways to make it available for service requests is to wrap it as a REST API via hosting services. For example, Amazon SageMaker can set up a persistent endpoint to get predictions from the client's model. However, clients are at their own security responsibilities to perform all of the necessary security configuration and management tasks.

Models embedded into edge devices (e.g., mobile, IoT) need to be optimized for CPU and memory usage. Running models on resource-constraint devices raise issues of run-time compatibility and portability. There is a need for cross-platform conversion for a trained model to match the edge devices' infrastructure. In practice, it is rarely feasible to directly deploy a trained ML model to an edge device. The most common way to write a custom C/C++ IO function. However, it becomes an increasingly cumbersome task to implement IO functionality of a complex trained model like a convolutional neural network, requiring more than a month of development effort [15].

From an engineering perspective, end-to-end ML applications consist of multiple components written in different programming languages. While abundant frameworks and tools are available to develop models, each framework and language has its specific types of operations and, consequently, produces a slightly different model. The heterogeneous code bases raise the difficulty of keeping consistency. Furthermore, tools for testing and error checking

on the deployed model can not tackle problems across the language barrier [16].

Issues at the Stage of Maintenance

With deploying an ML model in the wild, the scenario becomes complicated. There are many aspects to monitor, especially for a safety-critical system, such as input data, running environment, internals of the model, and output of the model [8]. Distribution shift and out-of-distribution inputs are prevalent problems [17]. The distribution shift can stem from the unrepresentative training data and non-stationary environments. It is likely to cause the relationship between the input and output is omitted or misrepresented.

If the existing model no longer satisfies performance criteria, a retraining or re-modeling procedure should be triggered. It is challenging to detect issues and changes in datasets that affect the quality of the model and invalidate evaluations. When such data issues have a drastic impact on the model performance, it is impractical to backtrack the root-cause to the dataset without metadata and lineage, which are rarely recorded along the ML lifecycle [16].

When updating a model with a fully or partially new training dataset, we expect that the retrained model is unforgettable, also referred to as backward compatibility. Clearly, given input to the outdated model and the retrained model, both outputs should be similar. For example, a fraud detection model will be updated when new types of fraud are known. The retrained fraud detection model is expected to recognize the new types of fraud as well as previously-recognized fraud. But it is hard to achieve, which may result in an inconsistent output of the system.

2.2.2 Existing Attacks

Since the lifecycle of ML involves many components and processes, it is significantly challenging to ensure the reliability and integrity of these components together or apart. There is growing awareness that potential "penetration points" exist for attackers to intrude an

ML-based system. From the model perspective, we discuss existing attacks in Training Phase and Inference phases where an adversary intrudes the system in the first place.

Training Phase

Adversary Goal. Attacks in training phase can be categorized into two types: *causative attacks* and *exploratory attacks*. Causative attacks attempt to impact or corrupt the model itself. The goal is to manipulate the behavior of an ML model. In a classification task, a tampered model can produce prediction to a particular wanted class, called as targeted attacks. Another type of attack is an untargeted attack, which degrades or corrupts the model's prediction, leading to random outputs. Exploratory attacks attempt to discover as much knowledge as possible about the learning algorithm used for the system and sensitive information from training data.

Attack Methods. One simplest and common way to implement causative attacks is to poison training data by injecting carefully crafted samples to compromise the model learning phase. Specifically, a trainer can learn a model with malicious behavior by adding a small trigger to a partial training dataset. Backdoor adversary in a learning system is first proposed by [18], where benign training samples are either blended with a key pattern or added with an accessory. For example, a particular pair of glasses is added to face photos used for face recognition model training, enabling the backdoored recognizer to misclassify any faces with certain glasses to a target person. In [19], they obtain a backdoored digit recognizer by training a model on samples that have a small pattern in the corner of the image. In another scenario where the adversary can not directly access the training dataset, the model can be retrained with reverse-engineered data added with a trigger, referred to as trojan attacks [20]. Alternatively, the adversaries can modify the learning environment to influence model training procedure, or even tamper the learning logic of the learning algorithm. Exploratory attacks could happen at federated learning where a model is collaboratively

learned by multiple parties without sharing local data with each other. Gradients are shared during distributed training. Zhu et al. [21] use an optimization method to obtain private training data from the publicly shared gradients.

Inference Phase

Adversary Goal. Attacks in the inference phase can be categorized into two types: *adversarial sample attacks* and *private inference attacks*. The goal of adversarial sample attacks is to fool ML models with a malicious sample; thus, the model produces an adversary desired result. Differently, private inference attacks do not target at the model's outputs; it aims to extract information about the training data or the model.
Attack Methods. Adversarial sample attacks are conducted under a black-box setting where attackers have no knowledge about the model. There are many ways to generate adversarial samples. In case of image classification, Gradient-based methods [22, 23, 24, 25, 26] construct adversarial sample generation as an optimization problem and change image pixels simultaneously at a fix scale along the gradient direction. Content-based methods are leveraging semantic information to perform the perturbation. For example, DeepXplore [27] maximize neuron coverage of deep learning system to expose many unexpected behaviors. Adversarial patch [28] inserted to any image can fool classifiers regardless of the patch's scale or location.

As for private inference attacks, here we introduce *membership inference* and *model inversion attacks*. Membership inference is to determine whether a given data record was in training dataset used for model learning. In [29], a basic membership inference attack against black-box models is presented. Shadow Training technique is invented to train an attack model that can distinguish the target model's outputs on members versus non-members of its training dataset. They create multiple "shadow models" that imitate the target model. Then they train an attack model on the labeled inputs and outputs of shadow models with a set of training samples. Further research [30] relax the adversarial assumptions that the

dataset used to train shadow models comes from the same distribution as the target model's training data, and the attacker needs to establish multiple shadow models with each one sharing the same structure as the target model. Therefore, membership inference attacks can be applicable at low cost and pose a more severe risk.

Model inversion attacks exploit confidence information, including sensitive features of input data and model parameters. An Evaluated Maximum a Posterior(MAP) estimator can be used to identify specific instances of the sensitive attribute for decision tree model [31, 32]. Additionally, in the case of image recognition, the attacker can use MI-Face [32] to recover images used in model training. The method uses gradient descent to minimize a cost function for reconstructing a class image with only given the class name and access to the recognition system. Another target of inversion attacks is model properties such as model architecture and parameters. In an ML-as-a-service system where a trained model can be accessed on a pay-per-query basis, models should be deemed confidential due to property protection or privacy consideration. The attacker can learn a close approximation of the target model using as few queries as possible. For extracting unknown parameters of a Logistic Regression model, the attacker at most needs to query N times to solve a linear system of N equations. The more complex the target model's structure, the more queries needed to recover a near-equivalent model [33].

2.3 Evidence Availability and Challenge of Evidence Usage

2.3.1 Evidence Preserving along ML lifecycle

Issues and vulnerabilities, as mentioned above, lead to uncertainties in the output of an ML-based system, and attacks on machine learning will impact its confidentiality, integrity, and availability. Therefore, there is a need to preserve evidence to ensure that the synthesise of ML components is fit for purpose. This evidence can be further used for troubleshooting

when the ML fails. Evidence-preserving along the ML lifecycle explains why the applied ML can be trusted by the system.

First of all, the lineage of data must be stored. The metadata for datasets is captured in *Data Preparation* stage, including sources of raw data, ways of collection, the intermediate status of transformation, versions of datasets, and statistics of datasets. An example of preserving evidence on data preprocessing is a fine-grained lineage of transformation pipeline [34]. Given an input to a transformation pipeline, the lineage system records a mapping of the input and output cell for each data transformer into a table. Then, all tables are jointed as a sequence of mapping tables, enabling users to examine intermediate data when two similar pipelines produce contradictory outputs. Moreover, a data lineage of the safety-critical system needs to keep track of data-accessed operations such as creation, update, deletion, and read on a particular data point.

The metadata for models is collected during *model development* stage, including environment setting, a lineage of the computation, evaluation results, and versions of the model. Specifically, computation metadata can help users to analyze and reason about the computational process and outcomes. For example, an open-source tool noWorkflow [35] collects provenance from experiment scripts such as script definition (e.g., function definitions, function arguments, function calls, and other static data), execution environment (e.g., operating system, platform version, dependencies), and run-time log. Another example is computation lineage for graph-based machine learning algorithms, which uses the derivatives as provenance and outputs a dependency graph with partial derivatives as weights. It can detect bugs that affect such dependency relationships and corresponding weights [36].

As for preserving evidence of a deployed ML-based service in practice, predictions of input instances are stored with a link to the model identifier that indicates which model gives the outcome, usually used for service providers to monitor service performance and check abnormal activities. Besides, model hashing can be stored to verify the integrity of

the model.

2.3.2 Challenge of Applying Evidence to Diagnosis

When a problem occurs, the first step is to establish the source of the problem. One needs to investigate where the origin of the problem is, what causes the problem, when it occurs and how it impacts the pipeline and the result. However, existing data provenance and forensics can not cope with the black-box nature of the ML model as a service. The research of ensuring ML safety is at an early stage, and there are many barriers for evidence collection, provenance analysis, and error diagnosis. It is challenging to decide what to preserve in advance along ML lifecycle, such as what operation needs to be monitored and what footprint is useful for examining inner-workings of ML-based applications.

Traditional programming software uses a logging mechanism to track every operation executed inside the program; thus can be used as evidence and information for back-tracing troubleshooting. However, recording every computation step of an input data forwarding to ML application is a heavy burden and not enough for us to figure out the problem for the following reasons. Computation data can not reveal a direct dependency relationship between individual prediction and the source of the problem due to the nature of ML algorithms. The understanding of model creation (how the algorithm creates model) is not applicable to explain how the trained model makes predictions and how parts of the model affect predictions. The linkage between these processes is blurred. Also, computation data can not directly be used for constructing trace evidence. Features learned by a model are represented by weights (e.g., neural network models) or other parameters (e.g., splits of a decision tree). It poses the challenge of reasoning the captured evidence and connecting it to the problem's actual semantics.

Chapter 3

AUTOMATED CONTAMINATED ATTRIBUTE LOCALIZATION

3.1 Introduction

We observe a proliferation of machine learning models integrated into various systems to solve real-world problems. Accordingly, the quality demands of machine learning (**ML**) are raised, and researchers focus on improving the accuracy of the models to provide a correct answer to the question the user asked. However, machine learning solutions with highly accurate models may not be able to find a correct answer if a question that the user submit contains an error. When the ML solution is deployed to solve real-world problems, it typically requires multiple steps to compose a question, such as collecting, preprocessing, and converting the data. For instance, let's assume a machine learning solution has deployed in a hospital to improve cancer diagnostics. The model classifies the tumor based on multiple aspects, including the patient's medical record, MRI image, and genome sequencing. To utilize this machine learning solution, we first need to compose a question that the ML model can understand. This phase typically includes multiple steps of data conversion and processing,

such as vector conversion, standardization, or normalization. If there exists a software bug, hardware fault, or even a human mistake, the composed question contains incorrect (or contaminated) data, and the ML model might not be able to answer it correctly. Once this type of ML failure happens, it is important to identify the root cause of the incorrect prediction to prevent future problems. However, it is a non-trivial challenge to trace back the root cause as the root cause is located in the question composition phase, that can be semantically far away from the machine learning processing.

Recently, failure examination techniques for machine learning have been proposed. There exist studies to explain individual prediction by model approximation [37, 38], attribution methods [39, 40, 41] , and inspect error by collecting lineage for a specific algorithm or process [34, 36]. However, they focus more on machine learning prediction process, and cannot identify erroneous data located in the testing input. There is a growing body of work on detecting common types of data errors (e.g., duplicates, rule violations, numeric outliers) in a large scale dataset. Krishnan et al. [42] presented a system that automates the process of detecting and repairing a class of data errors that occurs when an attribute value is out of the range of its value domain (e.g., the value of feature "age" > 130). Schelter et al. [43] provides an API for data quality verification by combining quality constraints with user-defined validation code (e.g., custom validation of the number of distinct non-null values in a column). Similar approaches require detailed knowledge of data features and customization of defining quality metrics [44, 45, 46]. However, different from visible errors (e.g., missing values, misspelling words, image rotation) that techniques mentioned above mainly target, data containing inconspicuous erroneous values are hard to detect and often unpreventable.

We call such erroneous data located in the question as *contaminated data* and the attribute containing the erroneous value as *contaminated attribute*. Reasoning the incorrect prediction caused by contaminated data requires non-trivial human effort. Error diagnosis or

debugging techniques for traditional software to identify the root cause of data errors (e.g., data flow tracking, or data provenance tracking) cannot be extended to the machine learning context due to the low interpretability of the machine learning model. In addition, the data collection and preprocessing stages are often decoupled from the ML pipeline, and the data delivered to the ML pipeline is typically in a raw format without semantic information, which makes the diagnosis of data error more challenging [47].

In this study, we propose a novel Automated Contaminated Attribute Localization (ACAL) system that can automatically quantify the relationship between each attribute and the occurrence of contaminated data. The proposed technique is model-agnostic, and we demonstrate that it can accurately pinpoint *contaminated attribute* without any prior knowledge of the attributes or machine learning model. The user only needs to provide a set of data with labels (e.g., training dataset).

In this study, we make the following contributions.

- We focus on a practical but less explored problem: the contaminated attribute localization for machine learning. If the part of data has been contaminated by software bugs, hardware faults, or human error, machine learning solutions cannot make a correct decision. Furthermore, it is a non-trivial challenge to trace back the root cause as it can be semantically far way from the machine learning processing. We develop a solution that can precisely pinpoint contaminated attributes without any prior knowledge of the model or attributes. Therefore the user can track down the root cause of the data contamination.

- We demonstrate that existing techniques, such as data cleaning or outlier detection, cannot effectively identify contaminated attributes if the values stay in the valid range.

- We develop three novel algorithms for 1) prototypical vector generation, 2) counterfeit data composition, and 3) Suspiciousness metric calculation. ACAL uses them to

quantify the suspiciousness of each attribute automatically and provides the rank of suspicious attributes to the user.

- We evaluate ACAL with ten real-world datasets, and the results show that ACAL can accurately identify the erroneous attribute with over 91% accuracy on average.

The rest of the chapter is organized as follows. To motivate our study, we discuss a realistic problem with existing outlier detection techniques and our solution in Section 3.2. Then we formally define the problem in Section 3.3, and present details of our solution in Section 3.4. Section 3.5 evaluates the performance of ACAL with ten real-world datasets and presents the results. Section 3.6 discusses the limitation of ACAL and future work. We review related work in Section 3.7.

3.2 Motivating Example

In this section, we present an example to demonstrate the problem we aim to tackle and the solution we propose in this study. Our example is based on a fictitious engineer Alice, who is a mobile App auditor, and her job is to inspect Android applications before allowing them shown on customers' mobile market. Due to the vast amount of mobile Apps to audit, Alice adopts a machine learning classifier for this task. The classifier is a decision-tree based model that uses Android app permissions as features to determine whether the app is malicious or benign. This classifier uses *requested permissions* as features for the classification. It uses 92 permissions as features in this case, and the model shows 94% accuracy to detect malicious Android applications.

Alice obtains the permission list directly from her Android APK files using APK analyzer [1], and homegrown python script to automatically parse output from APK analyzer and convert them into vector format to feed into the classifier. Unfortunately, the Python script

[1] GoogleAPKAnalyzer.https://developer.android.com/studio/command-line/apkanalyzer

has a bug that misses *READ_HISTORY_BOOKMARKS* permission in a specific situation. In particular, if an Android application requests both *READ_HISTORY_BOOKMARKS* and *WRITE_HISTORY _BOOKMARKS*, the Python script only recognizes *WRITE_HISTORY _BOOKMARKS*, but misses *READ_HISTORY_BOOKMARKS*. Therefore, the Python script misses the read permission request leading to the extracted feature value of *READ_HISTORY _BOOKMARKS* as contaminated value. The classifier shows that the app is benign with 100% confidence score.

In fact, the app contains malicious content and it infects the user device. Later, the user noticed the infection and wants to understand why the classifier failed to detect. Note that if she collects all permissions without error (i.e., including *READ_HISTORY_BOOKMARKS*) the result from the same model is *malicious* with a confidence score of 100%. Alice first confirms that the classification model has excellent accuracy 94% in detecting malware, and the classification process does not have any problem.

Then, she tries to investigate input data that she submitted to the model. First, Alice leverages the model interpretation technique to understand the importance of each feature and take a look into the important features. Particularly, she used SHAP [39], the most popular interpretation model to achieve the goal. It explains a particular prediction by assigning each feature an importance value to indicate its corresponding contribution to the output. It is widely used in practice such as clinical diagnosis and nature environment forecasting [40, 48, 49, 50].

Figure 3.1 shows the output from SHAP on the contaminated data. The feature vector of the Malware App contains a faulty value, which makes the classifier predict it as “Benign”. The value of *READ_HISTORY _BOOKMARKS* should be 1 but mistakenly collected as 0. The color box represents the relative impact of attributes on the prediction. The base value is the average model output over the training dataset. Red features contribute to the prediction towards “benign” while blue features push the prediction towards “malicious”.

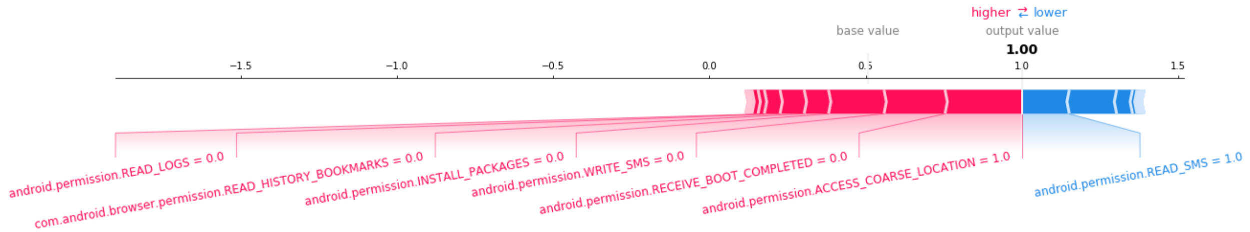


Figure 3.1: Prediction Explanation by SHAP. The sample instance contains a contaminated attribute " READ_HISTORY_BOOKMARKS" = 0 (Not Used) and the prediction class of the sample is "Benign" with 1.0 confidence.

We can see that the faulty-value feature is only ranked in the fifth most influential feature contributing to the prediction towards "benign". The top four features are the correct values.

Next, Alice applies data outlier detection technique [51] to identify unusual objects that are suspiciously different than the majority of observations. However, the user can not find the contaminated value because it is still in the range of valid value (i.e., either "0: Permission Not Used" or "1: Permission Used"). Then, the user applies DBScan [52] to detect noisy data from the input; however, it fails to detect the contaminated data in this case because the faulty value is too close to normal data in the feature space and grouped into a particular cluster thus can not be identified as an outlier.

Then, Alice decides to use ACAL to investigate the problem in the input data. ACAL requires 1) contaminated input, 2) the correct class (i.e., the fact for the non-contaminated input), 3) a set of in-distribution data (e.g., training data set or validation data set). We assume that the user confirms that the prediction result is incorrect, and the correct class can be easily discovered as the output of the model is either "malicious" or "benign". If the ML model is a multi-class classifier, we believe the user can discover the correct class during the process of detection and confirmation of the incorrect prediction. ACAL does not require any other knowledge, such as the model hierarchy or the information of attributes. In this scenario, we assume that Alice is using the training dataset. Note that we evaluate ACAL with various size of dataset in Section 3.5.5, and we observe that the 50% size of the

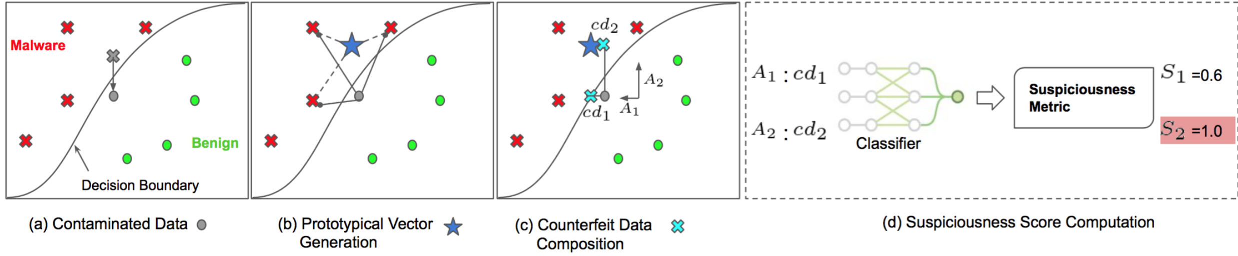


Figure 3.2: High-Level Workflow of ACAL.

original training set performs almost the same as the full-size training set (1% lower overall accuracy).

Alice provides the contaminated data, the correct class, and the training set to ACAL, and ACAL automatically calculates the suspiciousness of each attribute in the contaminated data, and outputs the rank of suspiciousness attributes. In this case, ACAL correctly put *READ_HISTORY_BOOKMARKS* as the most suspicious attribute. Figure 3.2 shows a high-level workflow of how ACAL works. In (a), the gray $\bullet$ symbol shows the contaminated input. The correct class is "malicious," but due to the erroneous attribute value, the model predicts it as "benign". As a first step, ACAL generates a *prototypical vector* by searching particular nearest neighbors of the contaminated data and then averaging these points. In Figure 3.2 (b), the blue $\star$ symbol represents the generated prototypical vector computed from 3-nearest neighbors. Then, ACAL composes *counterfeit data* by perturbing the values of each attribute of the contaminated input. Figure 3.2 (c) shows generated two counterfeits, cd_1 and cd_2, by perturbing two attributes, A_1 and A_2, respectively. For the last step, ACAL computes the suspiciousness score for each attribute. Figure 3.2 (d) shows that the attribute A_2 is the most suspicious one. We discuss the details of each step and our algorithms in section 3.4.

3.3 Problem Definition

Classification Model. Let X be the feature space of the target model, Y the target variable. Given a dataset consisting of N labeled instances $S = \{(x_1, y_1), ..., (x_N, y_N)\}$ where each $x_i \in \mathbb{R}^D$ is the D-dimensional feature vector of an instance and $y_i \in \{1, ..., M\}$ is the corresponding true label denoted as y^*, a function $f : X \to Y$ represents the model that is used to predict the class of the target variable for an instance $x \in \mathbb{R}^D$. For an input x, the classification model assigns its label to be $\hat{y}$ =argmax$_i f_i(x)$ with a class possibility $p(y = \hat{y})$. The assigned label is correct if $\hat{y}$ is the same as the true label y^*.

Contaminated Data. Existing actual data $x \in X$ with D dimensional features/attributes for an observation target, $x = (a_1, ..., a_d)$, its ground truth is class y^* and the classifier f is capable of making a correct prediction. We call x' contaminated data of the observation target if

$$x' \in \mathbb{R}^D \wedge f(x') \neq y^* \wedge \sum_{i=1}^{d} \Delta(a_i, a_i') \leq t$$

$$, where\ x' = (a_1', ..., a_d'),\ 1 \leq t \leq d,\ \hat{y} = \text{argmax}_i f_i(x'),$$

$$\Delta(a_i, a_i') = \begin{cases} 1 & \text{if } a_i = a_i' \\ 0 & \text{if } a_i \neq a_i' \end{cases}$$

$\Delta(\cdot)$ denotes a distance function measuring the similarity of two attribute values and t is the threshold which limits the number of attributes with faulty value. Contaminated data x' is a valid input but it contains t number of contaminated attributes that are different from the actual value of the fact, which renders the model classifying x' to a class that the observation target not belong to.

The contradiction with the facts should be imputed to the data generation rather than model performance if the cause is due to contaminated data. Faulty values are an unexpected

problem in dealing with most of the real-world sources. There are various reasons for such a faulty occurrence. It may be due to the malfunction of equipment, absence of calibration measurement and human operation error. It may occurs during data preparation phrase where data collected from various sources and then fed to a series of preprossessing. However, most machine learning methods are based on the assumption that data are correct and complete.

Contaminated data is usually generated accidentally and it tends to be humanly unrecognizable. It is worth noting that any faulty attribute-value we consider is within normal distribution of the corresponding attribute. Investigators can hardly utilize abnormal analysis to pinpoint the untrustworthy attribute in a contaminated data.

Even though the contaminated data might not evidently noticeable, attention should be raised once it perceived or suspected. It could be an alarm that informs a problem in data sources at the early stage and thus avoid same accident in the future. A key challenge is how to define a metric which can clearly indicate the causality between input components and predictions. Our approach is to aid users by suggesting which attribute to inspect with a ordered list of suspiciousness scores. In this work, we consider one contaminated attribute for an individual instance ($t = 1$). Solution of other t settings will be discussed and addressed in the future work.

We highlight our assumptions and requirements of ACAL as follows:

- We target the problem of contaminated input data, and we assume that the classification model is benign and well trained. It is out of the scope of this work if the incorrect decision has made by the inaccuracy or incompleteness of the model itself.

- In this study, we focus on machine learning-based classification models. We assume that the input data is multivariate, and the attribute can be categorical or numerical.

- The user needs to provide 1) contaminated data, 2) the correct (or expected) class if the

data is not contaminated, 3) a set of in-distribution data (e.g., training or validation data set).

3.4 Methodology

Our approach, an Automated Contaminated Attribute Localization (ACAL), is based on the idea that if we can recover the correct value for the contaminated attribute, we can determine the cause of the counterfactual result. The proposed method includes the following main steps: ① *prototypical vector generation* (Algorithm 1), ② *counterfeit data composition* (Algorithm 2), ③ *suspiciousness metric computation* (Algorithm 3). The first step aims to generate a prototypical vector, where each dimension is assumed as an attribute-value closed to the corresponding actual value of the observation target. *Counterfeit data composition* utilizes the prototypical vector to compose a number of counterfeit data trying to mimic actual data. In the last step, the predicted output of those counterfeit data are used to measure the suspiciousness of attributes.

Prototypical vector generation. Given a classifier and a contaminated data (x') of an observation target (noted that the fact of x' is known (y^*) but the actual data is unknown), we try to recover the faulty attribute-value. In Algorithm 1, we search a few nearest neighbors of x' in existing data set (e.g., training set, validation set) to generate a prototypical vector (PV). Line 4-9 generates a set of candidate data used for searching particular nearest neighbors. A data is an candidate data (z) if the data is predicted to a class $\hat{y}$ the same as the true class that observation target belongs to ($\hat{y} = y^*$). The candidate set contains all candidate data, denoted as S_c. Line 10-12 computes the distances between x' and each candidate data in S_c, and Line 13 selects k candidate data closest to x' as k-nearest-neighbors, denoted as the subset $S_n = \{z_1, ..., z_k\}$. The number k is empirically selection. We use k=3 for all experiments in the evaluation section. At the end, a prototypical vector is computed

by mean of data in the subset S_n. Let $z_i^{(j)}$ denote jth attribute of ith data point in S_n, then defined a prototypical vector of x' as:

$$PV(x') = (\overline{a}_1, ..., \overline{a}_d), where\ \overline{a}_j = \frac{1}{k} \sum_{z_i \in S_n} z_i^{(j)}.$$

Algorithm 1: Generation of the Prototypical Vector of an Contaminated Data

---

Data: $x' \in \mathbb{R}^d,\ y^*,\ D_{train}$

1. $S_c \leftarrow \emptyset$;
2. $S_n \leftarrow \emptyset$;
3. $PV \leftarrow [0] * d$
4. **foreach** *trainingData* $x_i \in D_{train}$ **do**
5. $\quad \hat{y} \leftarrow Classifier(x_i)$;
6. $\quad$ **if** $\hat{y} == y^*$ **then**
7. $\quad\quad S_c \leftarrow S_c \wedge x_i$;
8. $\quad$ **end**
9. **end**
10. **foreach** *candidateData* $z_i \in S_c$ **do**
11. $\quad$ Compute Distance $d(x', z_i)$;
12. **end**
13. Compute set S_n containing z_i for the k smallest $d(x', z_i)$
14. **for** $j \leftarrow 1$ **to** d **do**
15. $\quad PV[j] \leftarrow \frac{1}{k} \sum_{z_i \in S_n} z_i^{(j)}$;
16. **end**
17. **return** PV;

Algorithm 2: Compose Counterfeit Data For Each Attribute of the Contaminated Data

---

Data: $x' = (a'_1, ..., a'_d), PV = (\overline{a}_1, ..., \overline{a}_d)$

1. $A_{cd} \leftarrow (cd_1, .., cd_d), \forall cd_i :\ cd_i \leftarrow x'$;
2. **for** $i \leftarrow 1$ **to** d **do**
3. $\quad cd_i^{(i)} \leftarrow \overline{a}_i$;
4. **end**
5. **return** A_{cd};

Counterfeit data composition. After generating a prototypical vector $PV(x')$, the next step is to compose counterfeit data for each dimension, trying to mimic actual data, denoted

as cd. Let the contaminated data be $x' = (a'_1, ..., a'_d)$ as defined in section 3.3. For each dimension i, we compose a counterfeit data cd_i by copying x' and replacing ith attribute value with $\overline{a}_i$. In this way, we generate counterfeit data for each attribute. The intuition behind this is that the contaminated value attribute can be recovered by replacing it with a value closed to the actual data. Thus, every attribute is considered, and the corresponding counterfeit data is constructed.

Algorithm 3: Compute Suspiciousness Metric for Each Attribute of the Contaminated Data

```
   Data: cd_1, cd_2, ..., cd_d
1  Scores ← [0] * d;
2  for i ← 1 to d do
3  |  ŷ, p ← Classifier(cd_i);
4  |  if ŷ == y* then
5  |  |  Scores[i] ← p;
6  |  else
7  |  |  Scores[i] ← −1 * p;
8  |  end
9  end
10 return Scores;
```

Suspiciousness metric computation. In Algorithm 3, we utilize prediction of the counterfeit data to infer the suspiciousness of the corresponding attribute. We feed the counterfeit data to the classifier one by one. The prediction of the classifier on counterfeit data has two cases. One is the predicted class $\hat{y}$ same as the fact of contaminated data, $\hat{y} = y^*$(first case). Otherwise, $\hat{y} \neq y^*$(second case). If a counterfeit data cd_i predicted as y^*, cd_i is more likely be closer to the actual data of x' than those falls into the second case. Because the replaced ith attribute brings it across the decision boundary to class y^* and the attribute might be the contaminated one causing the misclassification of x'. In other word, the replaced ith attribute is more suspicious than other attributes that can not alter prediction to the fact. Formally, for any counterfeit data cd_i, the prediction result consisting of predicted class and class probability $(\hat{y}_i, p_i)$, define a suspiciousness score for ith attribute of the contaminated

data x' as:

$$score_i(x') = \begin{cases} p_i & \text{if } \hat{y}_i = y^* \\ -p_i & \text{if } \hat{y}_i \neq y^* \end{cases}$$

,where $1 \leq i \leq d$ and d represents the number of attributes. For cd_i falls into first case, we use predicted class possibility p_i as suspiciousness metric to decide the ranking of relevant attribute a_i. For cd_i falls into second case, the contaminated data is either pushed closer to the decision boundary or pushed to other wrong classes by replacing ith attribute. Thus, we think that the replaced attribute of data cd_i is more suspicious if the prediction possibility of cd_i is lower. The suspiciousness score of the ith attribute is defined as negative of predicted class possibility.

Finally, the mechanism score each attribute by its "suspiciousness contribution" to the output such that the most suspicious attribute has the highest score. Ranking suspiciousness score in descending order, an analyst is able to examine attributes ranked at the top of suspiciousness metric table and back trace to the source of the faulty, whenever a contaminated data is found.

3.5 Evaluation

In this section, we empirically evaluate our mechanism with respect to efficiency and effectiveness. First, we describe the experiment setup, data sets and models used in our evaluation. Second, we describe our process to generate contaminated data, and present the evaluation results. Third, we discuss details of cases that ACAL shows low accuracy during the experiments.

Class Type	Data Set	#Samples	Dimension d
Binary	Breast Cancer	569	30
	Back Pain	310	12
	Mushroom	8124	21
	Android Malware	398	92
	Phishing Website	11,054	31
	Firm Audit	775	23
Multi-Class	Cardiotocography (3)	2126	21
	Iris (3)	160	4
	Orthopedic (3)	310	6
	Dermatology (6)	366	33

Table 3.1: Dataset used in the Evaluation

3.5.1 Experiment Setup

We use general machine learning training and testing processes in this evaluation. Specifically, we use scikit-learn (http://scikit-learn.org) for model implementation and we use a server equipped with 28 Xeon CPU E5-2697 2.6GHz processors, 188GB of RAM to conduct all experiments.

3.5.2 Data Preparation

Datasets. We evaluate the performance of ACAL on 10 open-source datasets from UCI [53] and Kaggle. All these datasets are widely used and reflect real-world problems. There are 6 binary classification dataset: Breast Cancer [2], Back Pain [3], Mushroom [4], Android Malware [5] [54], Phishing Website [6] [55], Firm Audit [7], and 4 multi-class classification:

[2] http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)
[3] https://www.kaggle.com/sammy123/lower-back-pain-symptoms-dataset
[4] http://archive.ics.uci.edu/ml/datasets/Mushroom
[5] https://www.kaggle.com/xwolf12/datasetandroidpermissions
[6] http://archive.ics.uci.edu/ml/datasets/Phishing+Websites
[7] https://www.kaggle.com/sid321axn/audit-data

Cardiotocography [8], Iris [9], Orthopedic [10], Dermatology [11]. Details of datasets are listed in Table 3.1. The number in the parentheses denotes the number of classes. Each data set is split into training set D_{train} and testing set D_{test} with ratio 0.2, which is commonly used in practice. In a experiment, D_{train} is used to learn a classifier and also used as local data set for prototypical vector generation. D_{test} is used for model testing and contaminated data generation.

Models. Our approach is ML-model agnostic. We evaluate ACAL with five distinct models, including Logistic Regression (**LR**), decision tree (**DT**), random forest (**RF**), Naive Bayes (**NB**) and neural network Multi-layer Perceptron (**MLP**).

Simulating Data Contamination. In this evaluation, we simulate contaminated data such that we have ground truth of the correct class and contaminated attribute to perform evaluation. As discussed earlier, if the contaminated data is out of valid range, they can be easily detected by outlier detection techniques, and we only consider erroneous inputs that still remain in the valid range. Furthermore, the contaminated data should show the incorrect (or different) result from the original input. Thus we do not consider cases where the ML decisions of the correct input and contaminated input are the same. Given a trained ML model with a testing data set, a set of contaminated data ($D_{x'}$) is generated with the following steps:

1. **Identifying Valid Range.** For each attribute (a_i) considered by the model, we identify valid values for the attribute by collecting all values of the attribute from training samples. We denoted valid values for attribute a_i as V_i.

2. **Data Perturbation.** We perturb each attribute of the testing data x_j by converting the value to other valid values in V_i. We generate a perturbed data set that includes all

[8] http://archive.ics.uci.edu/ml/datasets/Cardiotocography
[9] http://archive.ics.uci.edu/ml/datasets/iris
[10] https://www.kaggle.com/uciml/biomechanical-features-of-orthopedic-patients
[11] http://archive.ics.uci.edu/ml/datasets/dermatology

possible valid but perturbed data. We then feed them into the ML model and discard if the prediction of the perturbed data is the same as the original one. We also discard the data if the prediction confidence is lower than the threshold, λ. The reason is that we believe lower confidence results can be easily noticed by the user, and the incorrect prediction may less affect the user than the output with high confidence. We set λ to 0.65 in this evaluation, and we discuss how λ affects the quantity of generated data and our detection accuracy in the later in this section.

3. **Test Data Generation.** The data perturbation process completes after it considers all data in the testing set. Finally, we obtain a set of contaminated data $D_{x'} = \{x'_{ij}\}$ and each point x'_{ij} stands for a contaminated data with faulty-value attribute i generated from testing data x_j.

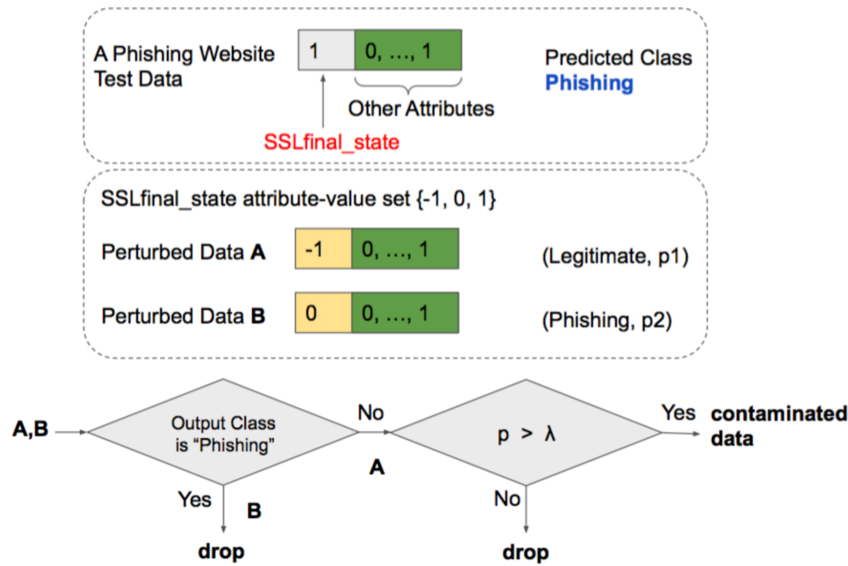


Figure 3.3: The example of contaminated data generation

Case Study. We take the Phishing Website dataset as a case study to demonstrate how we generate contaminated data from the testing set D_{test}. We use the Random Forest model in this example. In Figure 3.3, we illustrate how we generate a set of contaminated data. This

dataset has 31 attributes, and we consider the attribute *SSLfinal_state* in this example. This attribute indicates the extent of risk of the https certificate issuer and the certificate age (a website using SSL is assigned a certificate by Certificate Authorities). There exist three valid values in this attribute; low (https certificate is trusted), moderate (https certificate is not trusted), or high (otherwise, for example, the website does not use SSL). These values encoded as {-1, 0, 1}. The first testing data has {1} for *SSLfinal_state* attribute, and it is labeled as "phishing". The random forest model correctly predicts it with a confidence score of 100%. To generate contaminated data from this, we perturb the value for the attribute to {-1} and {0} and feed them to the model. Note that we keep original values for other attributes. The random forest predicts "Legitimate" for the perturbed data with {-1} but "phishing" for the data with the value of {0}. We discard the later one as the prediction output is the same as the original data. For the perturbed data that causes incorrect prediction, and its confidence score is higher than the threshold (65% in this case), we treat it as the contaminated data. We repeat this for all testing data and all attributes. Noted if we set a threshold of confidence score lower, we might generate more contaminated data; however, incorrect output with significantly low confidence less affects the user. We further discuss how the threshold, λ, affects contaminated data generation, and the detection accuracy of ACAL in the later this section.

3.5.3 Evaluation Result

Evaluation Metric. In this section, we present the accuracy of ACAL against generated contaminated data set $D_{x'}$. To measure the detection accuracy of ACAL, we use rank-at-top-K rate. As we discussed in Section 3.4, ACAL automatically calculates suspiciousness score for each attribute in the input data and rank the scores in descending order. If the contaminated attribute is ranked at top K, we count it as detected. The rank-at-top-K rate

is calculated as:

$$topKRate(D_{x'}) = \frac{\mu}{|D_{x'}|}$$

,where μ is the number of true positive and $|D_{x'}|$ is the number of contaminated data in the set $D_{x'}$. In this evaluation, we use rank-at-top-1 rate, meaning that we only count as detected if the contaminated attribute is ranked as the most suspicious attribute.

Data Set	**Model**	**Accuracy**	$\|D_{x'}\|$	**Top-1 Rate**	
		mean	mean	mean	std
Phishing Website	**LR**	0.929	2198.0	0.900	0.0
	NB	0.595	1802.0	0.874	0.0
	RF	0.974	2252.2	0.773	0.015
	DT	0.970	5791.8	0.746	0.002
	MLP	0.944	3515.0	0.875	0.010
Android Malware	**LR**	0.950	24.0	0.792	0.0
	NB	0.800	1042.0	0.971	0.0
	RF	0.928	36.8	0.902	0.032
	DT	0.925	292.4	0.940	0.004
	MLP	0.938	270.6	0.757	0.027
Firm Audit	**LR**	0.961	170.0	0.876	0.0
	NB	1.000	98308.0	0.999	0.0
	RF	1.000	13296.6	1.000	0.0
	DT	1.000	46520.0	1.000	0.0
	MLP	0.963	1087.8	0.939	0.053
Back Pain	**LR**	0.774	289.0	0.952	0.0
	NB	0.758	5413.0	0.985	0.0
	RF	0.794	4276.6	0.817	0.036

	DT	0.790	6639.6	0.734	0.001
	MLP	0.852	7554.0	0.829	0.034
Mushroom	**LR**	0.952	3536.0	0.999	0.0
	NB	0.932	8622.0	0.985	0.0
	RF	1.000	47.2	1.000	0.0
	DT	1.000	12342.0	0.998	0.001
	MLP	0.979	5823.4	0.999	0.002
Breast Cancer	**LR**	0.982	1229.0	1.000	0.0
	NB	0.921	30662.0	0.844	0.0
	RF	0.928	1267.0	0.856	0.235
	DT	0.900	31649.4	0.790	0.067
	MLP	0.977	20051.4	0.833	0.045
Cardiotocography	**LR**	0.883	1349.0	0.999	0.0
	NB	0.707	5246.0	0.950	0.0
	RF	0.946	305.0	0.878	0.124
	DT	0.931	5070.6	0.917	0.005
	MLP	0.876	1890.2	0.992	0.006
Orthopedic	**LR**	0.742	90.0	1.000	0.0
	NB	0.855	9645.0	0.949	0.0
	RF	0.826	7762.8	0.907	0.033
	DT	0.806	13717.0	0.763	0.017
	MLP	0.787	5352.6	0.944	0.050
Iris	**LR**	0.967	11.0	1.000	0.0
	NB	0.967	1007.0	0.985	0.0
	RF	0.953	132.8	0.884	0.093
	DT	0.967	699.6	0.942	0.036

	MLP	0.967	989.6	0.949	0.004
Dermatology	**LR**	0.986	17.0	1.00	0.0
	NB	0.851	681.0	0.930	0.0
	RF	0.968	18.0	0.816	0.130
	DT	0.941	563.8	0.963	0.005
	MLP	0.957	445.5	0.956	0.034

Table 3.2: Evaluation Result

Results. We evaluate all ten data sets with five distinct ML models, and we repeat all processes, including contaminated data generation and detection, five times. The third column of the table represents the mean value of the accuracy of the model over five runs. The column $|D_{x'}|$ shows the mean size of contaminated data and we used Top-1 Rate column shows detection rate of ACAL over five runs. For instance, the first row in the table represents that we evaluate ACAL on the Phishing Website dataset with Logistic Regression (LR) model. The LR model has 92.9% accuracy in detecting phishing web sites, and we generate contaminated data sets using the testing set, and the mean value of the generated contaminated data set is 2,198. We evaluate ACAL against the contaminated data set, and the mean of the detection rate is 90% over five runs. We highlight the Top-1 detection accuracy of over 90%. The average detection accuracy of all ten datasets and five models is 91.4%. In Table 3.3, we present the mean and standard deviation of detection rates for rank-at-top-1, rank-at-top-2, and rank-at-top-3 on each model. As expected, Top-3 shows the best results, and Top-2 has slightly better accuracy than Top-1.

$\lambda = 0.65$	**Top-1 Rate**		**Top-2 Rate**		**Top-3 Rate**	
Model	mean	std	mean	std	mean	std
LR	0.952	0.069	0.982	0.038	0.984	0.032
NB	0.947	0.048	0.974	0.032	0.979	0.030
RF	0.883	0.071	0.932	0.066	0.936	0.064
DT	0.879	0.102	0.929	0.068	0.934	0.063
MLP	0.910	0.077	0.946	0.047	0.959	0.036
Total	0.914	0.082	0.953	0.056	0.958	0.052

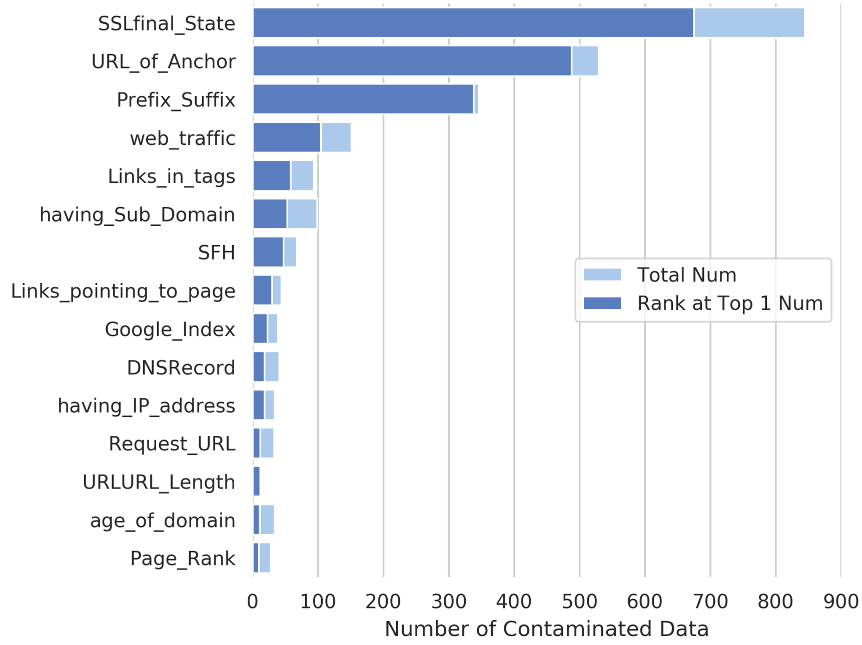
Table 3.3: Average Accuracy of Top-1, Top-2, and Top-3 rates

3.5.4 Case Study: Phishing Website with RF

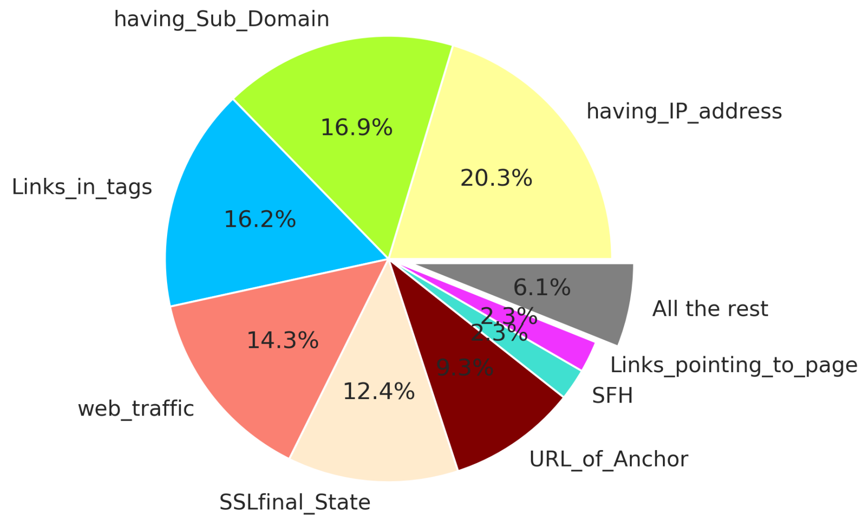
In this section, we take the Phishing Website dataset[55] and the Random Forest (RF) model as a case study example and discuss details of the detection accuracy. The accuracy of RF classifier is 97.42% and the number of testing data ($|D_{test}|$) provided by the dataset is 2,211. As discussed earlier, we generate 2,252 contaminated data ($|D_{x'}|$), and use them to evaluate the accuracy of ACAL in this case study.

Result Break Down. The mean of the detection accuracy of the Top-1 rate is 77.3%, which is one of the worst accuracies in our evaluation. To take a closer look into this result, we list the top 15 attributes (out of 31) that have the most contaminated data in Figure 3.4(a). The bar graphs show the total number of contaminated data in each attribute, as well as the number of data correctly detected by ACAL. For instance, there are 845 data that have contaminated value in *SSLfinal_state*, and ACAL detects 675 of them as the contaminated attribute at Top-1 and misses 170.

Then we take a closer look into the missed cases that non-contaminated attributes are incorrectly ranked at top 1 by ACAL. Figure 3.4(b) shows the percentage of instances where our approach incorrectly ranks non-contaminated attributes at Top-1. For example, *having_IP_address* attribute is incorrectly ranked as the most suspicious attribute in 22.2% of total failure detections. We observed that six attributes contribute almost 90% of the detec-



(a) Distribution of True Positive Instance over Attributes



(b) Distribution of Non-contaminated Attribute Ranked at Top 1 in Failure Instance

Figure 3.4: Case Study Example: Phishing Website with Random Forest

tion failures. We found that the reason is that the counterfeit data of the non-contaminated attribute are closer to the group of neighbors than the counterfeit data of the contaminated attribute. It shows that the prediction of the corresponding counterfeit data can be overwhelmed by the prediction of other counterfeit data.

Dataset Size (% of the training set)	**Top-1 Rate**		**Top-2 Rate**		**Top-3 Rate**	
	mean	std	mean	std	mean	std
25%	0.892	0.114	0.937	0.085	0.944	0.077
50%	0.906	0.064	0.948	0.064	0.955	0.059
75%	0.914	0.086	0.952	0.059	0.957	0.055
100%	0.914	0.082	0.953	0.056	0.958	0.052

Table 3.4: The effect of the size of labeled dataset

3.5.5 The Effect of Dataset Size

In the evaluation, we use training set D_{train} as user known dataset for prototypical vector generation. In this section, we test ACAL with different sizes of the datasets to understand the effect of the dataset size on ACAL. We first generate a known dataset with different sizes by randomly sampling a fraction of training data. Except for the known datasets, we use the exact same experiment setting as Table 3.2, including the contaminated input data. Table 3.4 shows the results. The first column shows the size of the dataset and the proportional size compared to the original training set. We present the mean and standard deviation of the detection rates of Top-1, Top-2, and Top-3. In this evaluation, we can observe that if the user provides a labeled dataset that is half of the original training set size, ACAL can still perform well. The mean accuracy is only 1% lower than using a full training set.

λ	**Top-1 Rate**		**Top-2 Rate**		**Top-3 Rate**	
	mean	std	mean	std	mean	std
0.55	0.878	0.117	0.927	0.083	0.940	0.072
0.65	0.914	0.082	0.953	0.056	0.958	0.052
0.75	0.936	0.077	0.963	0.052	0.968	0.049
0.85	0.945	0.081	0.966	0.061	0.969	0.058
0.95	0.934	0.119	0.958	0.097	0.961	0.095

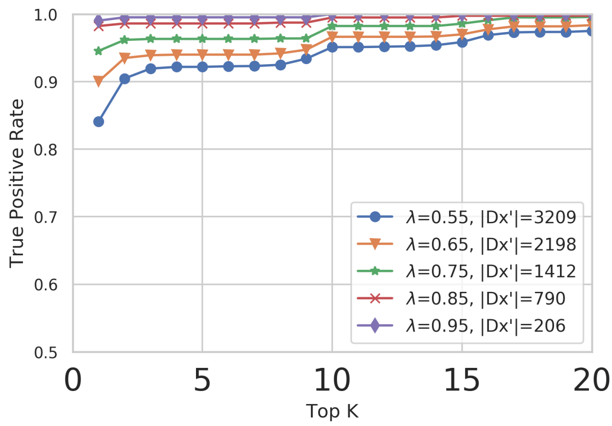
Table 3.5: The effect of the different Hyper-parameter λ

3.5.6 The Effect of Threshold (λ)

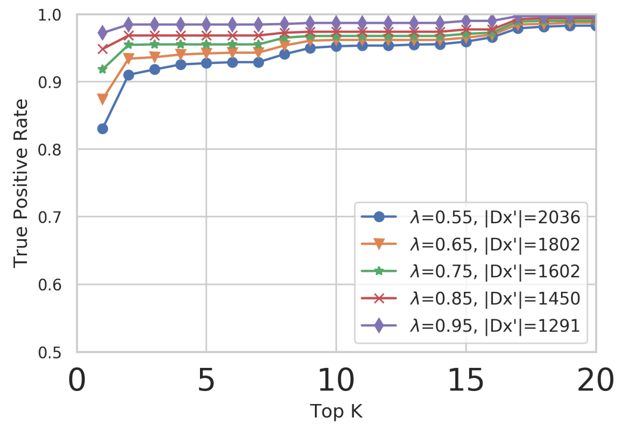
In this section, we discuss how the threshold (λ) affects the contaminated data generation and the accuracy of ACAL. As we discussed earlier, we discard the perturbed data if the prediction confidence is lower than the threshold, λ. Therefore, if we set the threshold high, we can generate less number of contaminated data. On the other hand, if we use the lower threshold, we can use more contaminated data for the evaluation, but we believe the incorrect prediction with high confidence can be a more significant problem as it can have a severe effect on the user.

Table 3.5 shows average Top-1, Top-2,and Top-3 rates on different λ settings. For Top1-Rate, the average performance has its peak at 0.945 when λ is 0.85. We observed that results had fluctuated a bit among different λ settings, but all can reach over 0.94 on Top3-Rate. We can see that the mean values are not significantly different between λ of 0.85 and λ of 0.95. However, the deviation is at the highest on the Top-1 rate at $\lambda = 0.95$. We examined it by further breaking down to each model, and the results show that the RF model has the lowest average Top-1 rate, 0.848, while the average of LR, NB, and MLP is 0.998, 0.98, and 0.979, respectively. As expected, when the λ is set to 0.95, we cannot generate any contaminated data for some datasets and models, such as Mushroom RF and Breast Cancer RF.

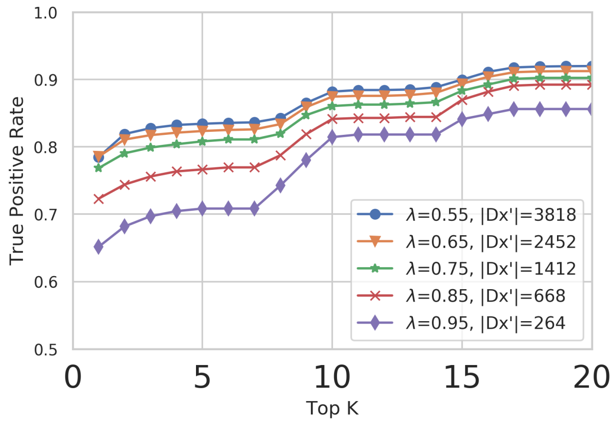
Furthermore, to take closer look into the effect of λ, we use the Phishing Website dataset



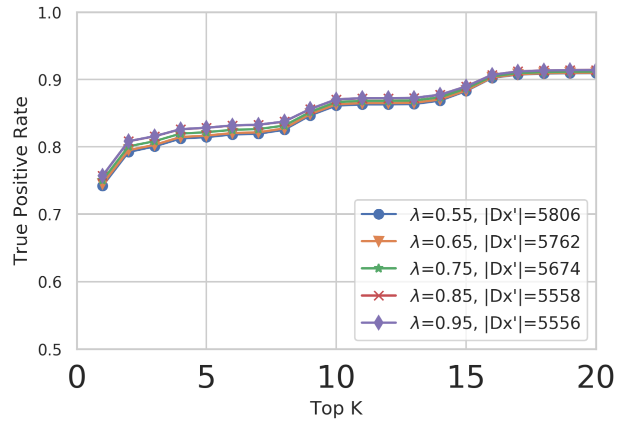
(a) Top-K Rate of Logistic Regression (**LR**)



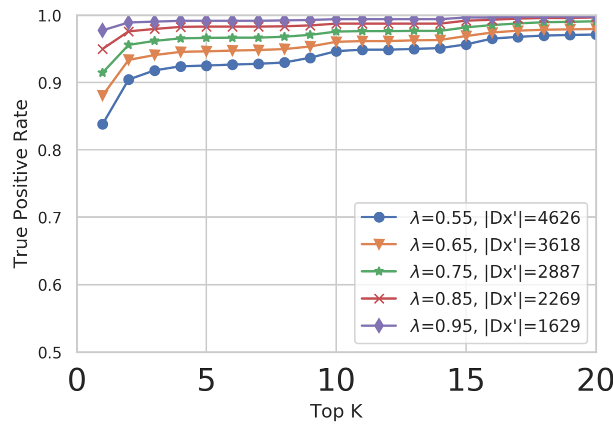
(b) Top-K Rate of Naive Bayes (**NB**)



(c) Top-K Rate of Random Forest (**RF**)



(d) Top-K Rate of Decision Tree (**DT**)



(e) TopK Rate of Multi-layer Perceptron (**MLP**)

Figure 3.5: Top-K Rate of Different Models on Phishing Website Dataset with different Hyper-parameter λ

as an example. Figure 3.5 shows the accuracy of Top-K rate for five models. Each line represents the result of a λ setting and the size of generated contaminated data ($|D_{x'}|$) is also listed. As expected, higher thresholds generate less contaminated data in all cases. We also observe that the Top-1 accuracy of ACAL mostly increases with the higher rate threshold.

We use the threshold, λ, of 0.65 for all experiments across the study. Although higher λ has better performance, we use relatively low value to generate a large number of contaminated data and to make the evaluation more comprehensive.

3.5.7 Existing Technique: Outlier Detection

Outlier detection is a technique to identify unusual objects that are suspiciously different from the majority of observations [51]. It is widely accepted in practice to prepare data for machine learning models. In order to evaluate whether contaminated data can be detected by the outlier detection technique, we used a well-known data clustering algorithm, DBScan[52], to build a outlier detector. DBScan is one of the most effective outlier detection methods with unsupervised learning mode. The basic idea is that DBScan groups data in high-density regions into clusters and marks data in low-density regions as outliers.

We use data from Phishing Website dataset with Random Forest (**RF**) classifier to demonstrate the performance of outlier detection on contaminated data problem. We adopted HDBSCAN API (https://hdbscan.readthedocs.io) to implement outlier detection. We randomly sample 950 data from testing set (D_{test}), and 50 contaminated data from contaminated dataset ($D_{x'}$), thus we got 1000 samples in total. Figure 3.6 visualizes the output of DBScan in two dimensions by using T-SNE. The outliers detected by DBScan are marked as red dots whose outlier scores are over 0.95 percentile. It only detects 2 out of 50 contaminated data as outliers marked as $\times$ symbol in the figure.

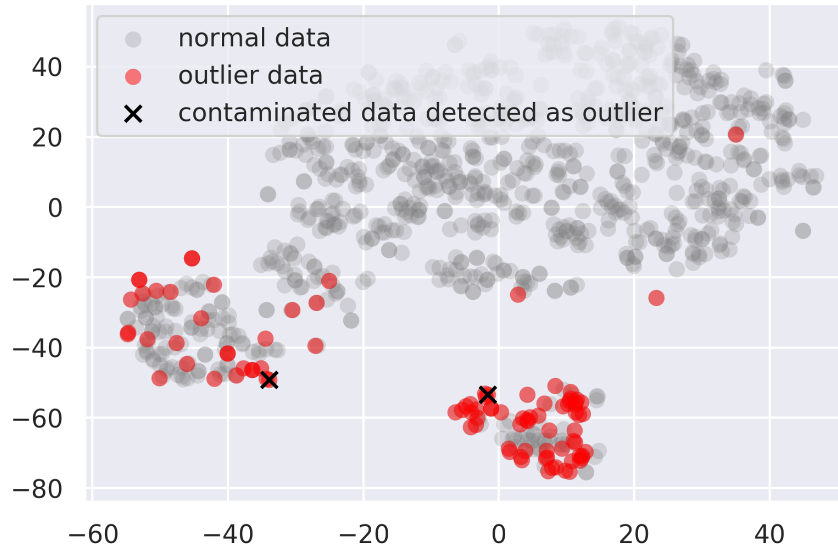


Figure 3.6: DBScan Outlier Detection on Contaminated Data

3.6 Discussion

Limitations of the Current Design. The current design of ACAL requires the user to provide the correct class for the contaminated input. ACAL searches and calculates a distance to neighbors using that information. We cannot handle the case where the user only knows that the answer is incorrect, but cannot be sure about the correct class. We believe this limitation can be addressed with a slight modification to the ACAL. Specifically, we plan to add one more layer on top of the current ACAL that can calculate suspiciousness scores for each attribute, and also for each available class. We then calculate the overall suspiciousness score of each attribute and present the most suspicious one to the user. This extension will be our immediate future work. Another limitation of the current design of ACAL is that we do not support if the input has more than two contaminated attribute values.

Choice of Distance Function. It is important to choose a proper distance function to measure the similarity distance between the contaminated data and each of the training data.

We have tested several distance functions, including Cosine, Euclidean, and Manhattan. We observed that Manhattan distance outperforms others in all cases and particularly performs well if the dataset contains various types of data, such as numerical datasets and categorical datasets. Therefore, we use Manhattan distance for all experiments.

3.7 Related Work

Data Cleaning and Outlier Detection. Data cleaning is a process of ensuring data to meet basic "data are right" standards[56] and this is an essential step for data preparation before feeding it to the machine learning pipeline. Various data quality problems (e.g, missing values and integrity constraints violations) can be captured by qualitative techniques, such as rules and pattern detection approaches. Statistics-based techniques are largely used for abnormality detection (e.g., more than 3 times of the standard deviation in Gaussian data). However, the faulty attribute-value in contaminated data is amenable to these data cleaning techniques. It is neither violating constraints nor outside the expected range. Outlier detection is a technique to identify suspicious data that are abnormally different from the majority of observations [51]. In section 3.5.7, we discuss and demonstrate that outlier detection does not become effective in detecting contaminated inputs as it still appears to be closed to normal data.

Decision Explanation. Decision explanation methods in machine learning aim to explain the reason why the model made the prediction. A model-agnostic explanation is to train a local surrogate model to explain the prediction of a single instance. LIME [37] finds a model that approximates the original model to learn local behaviors, and it selects a few explanations as representatives to help the user to understand the original model's prediction. There exists other methods [57, 58, 59] that leverage the similar locality intuition. Another direction is to provide influence estimates for each feature. For instance, QII [60] informed

the design of transparency reports that accompany system decisions and quantified the joint influence of a set of features on outcomes. Similarly adoption of cooperative game theory as an influence metric, SHAP [39] calculates the marginal contribution of individual feature to the output and it is better consistency with human intuition than previous methods [61, 62]. These feature attribution methods try to explain what are the main features that affect the decision-making process. Decision explanation methods focus on revealing how attributes contribute to the prediction that the model made. On the other hand, it is not specifically designed for pinpointing the erroneous data from the input, and thus quantifying the suspiciousness attributes that we aim to solve is not in the focus of decision explanation approaches.

Counterfactuals Handling. Explanations[63, 64] are receiving increased attention to answer questions such as "why this prediction, not that". For instance, Rathi [64] proposes a technique that utilizes SHAP value to find features that are against the prediction, and it mutates such features to generate counterfactual data points. Then it demonstrates the answer to the contrastive query. However, it is impractical to validate the causal features for the counterfactuals.

Root Cause Analysis. While root cause analysis for traditional software problems been intensively studied [65, 66, 67], there is limited techniques proposed for root cause identification in context of ML. Recently, Ma et al. [36] proposed a novel technique that leverages data provenance tracking to address the problem of fault propagation in graph-based machine learning algorithms. Their technique focuses on identifying the input-output dependencies and quantifying the importance of individual inputs. They used a weighted dependency graph with partial derivatives as the weight to detect bugs that affect such dependency relationships and corresponding weights. Zhang et al. [34] captured fine-grained lineage for each data transformation process during the data preparation stage, which enables users to examine intermediate results when two similar pipelines produce contradictory outputs. In

addition, undesirable behaviors of models and fault identification of models have been studied, including state differential analysis [68] and fuzz testing [69, 70]. Existing root cause analysis methods for machine learning likely have a specific target model (e.g., graph-based model) or specific data preprocessing step (e.g., image cell calculation), and they try to preserve the intermediate result rather than identify the contaminated attribute in the input data. On the other hand, ACAL specifically targets detecting contaminated attributes, and ACAL is an ML model-agnostic approach.

Adversarial Machine Learning. Recently, many studies show that Neural Network (**NN**) can be vulnerable to adversarial sample attacks. For instance, Szegedy et al. [71] demonstrate the existence of adversarial negatives that appears to be in contradiction with the neural network's ability, and adversarial samples that contain imperceptible perturbations can cause the misclassification. Other studies also demonstrate successful adversarial attacks on deep neural networks [22, 23, 24, 25, 26, 72]. Content-based methods are leveraging semantic information to perform the perturbation. Specifically, DeepXplore [27] maximize neuron coverage of deep learning system to expose many unexpected behaviors. Adversarial patch [28] inserted to any image can fool classifiers regardless of the scale or location of the patch.

There also have been substantial studies performed on defending against adversarial samples. Feinman et al. [73] utilized kernel density estimates in the subspace of the last hidden network layer and Bayesian neural network uncertainty estimates to identify adversarial samples. Ma et al. [74] characterized the dimensional properties of adversarial regions with Local Intrinsic Dimensionality (LID) and used LID values as a measure to discriminate adversarial samples. Inspired by ideas from invariant checking in the detection of software attacks, NIC [26] proposed a combined channel of value invariant and provenance invariant to detect input that violates the invariant distributions. We believe ACAL can be extended to detect adversarial samples that can cause a misclassification and it will be our future work.

Chapter 4

INTEGRITY CHECKING FOR NEURAL NETWORKS

4.1 Problem Statement

Cloud computing amplifies the power of machine learning, especially DNN. More and more platforms offer a cloud environment wherein users can outsource a model as a service, and a large number of service requests can be handled. While hosting a model in the cloud reliefs users of the computational and management heavy-lifting, there exist uncertain risks that the model is being modified. For example, malicious third-party host an attack to deflect the normal behavior of the model, or the cloud provider itself may cheat the user with a lossy compressed model to reduce resources cost [75]. Thus, ensuring the integrity of the model performed by the cloud becomes a security concern. In this section, possible scenarios of integrity violation are discussed, and then the idea of integrity checking is described.

4.1.1 Integrity Violation

Attacks from malicious third parties. Recent attacks on deep learning model are variant including adversarial ML [76], backdoor injection [18, 19, 77], and trojan attack [20, 78]. We focus on the type of attacks that aim to classify a tampered input to a target or an arbitrary class without impacting model performance on benign inputs. There are two scenarios for such attacks based on whether it directly uses the client's training data.

a) The scenario of directly accessing training data is that the client deploys model on the cloud and outsources training/updating procedures to the cloud. During the training procedure, it gives attackers the chance to obtain the users' training data and train a malicious version of the model simultaneously. The attacker can either poison partial training data before training his model or use the benign model to retrain a few more epochs with a trigger injected training data. Once the training procedure finishes, the client downloads the benign model and applies detection techniques to check the model locally. At the same time, the attacker can stealthily replace the benign model with the malicious one. Assuming that the remote model remains the same, the client directly deploys the remote model, which is now malicious as a service.

b) The scenario of not directly accessing training data is that the client deploys a model on the cloud without uploading the training data. For example, existing trojan attacks eschew the requirement of the original training data. An attacker can reverse engineer inputs that lead to strong activation of particular neurons and then used reversed inputs as training data. Similarly, the attacker inverses the neural network to generate trojan triggers. With reverse engineered training data and the triggers, retraining part of the model can create a trojan model with attack success over 99%.

Threats from cloud providers. There are many possible threats from dishonest providers.

As known, deep neural networks are highly demanding in aspects of computation and memory resources. Considering a scenario of a DNN model deployed on the cloud as a service, the cloud provider charges the client according to the resources that the model service needs to run. Still, the dishonest provider reduces the client's actual resources to save costs. To reduce the service's actual cost, the cloud provider can compress the remote model or even run a simpler network in place of deep networks. Examples of common compression techniques are parameters pruning [79], reducing activation number [80], weight quantization [81] and binning parameters into buckets with hashing [82]. Although the compressed model's accuracy may be closed to the original model, modification without the client's agreement violates the integrity of the model, which matters both the users' trust of the service and the reputation of the provider. On the other hand, the threats might arise due to the provider's negligence of the security check. It is not equipped with a mechanism to check the security status of service components frequently, which may unintentionally lead to the breaking of model integrity.

4.1.2 Integrity Checking

Our goal is to check the integrity of a DNN model hosted on cloud platforms on behalf of the client. The client is the model owner and keeps the original model $f(W, x)$, but the client does not supervise the model running on the cloud. The remote model as a service is denoted as $f(W', x)$. Security threats might emerge from different parties. The attacker might exploit the weakness of transmission protocol and modify the model when it is uploaded to the cloud [43]. During the remote model's deployment, the malicious insiders have a chance to compromise it and inject malicious behaviors without raising the client's attention by maintaining its normal performance. On the other hand, the cloud provider possibly becomes dishonest and cheats at completeness service. Consequently, the remote model has been modified, and the weights of the model are slightly different from the original

model. In this scenario, the client wants to verify the remote model's integrity to assure that the remote model is the same one as the original model, namely, testing $W = W'$.

However, the client will face several challenges to implement the verification. Firstly, the control over the remote model is non-transparent from the client-side. Direct verification may lose efficacy. If the cloud is dishonest, conspicuous operations might make the cloud provider aware of the verification attempt and then act further cheating actions. Secondly, the remote model is regarded as a black-box. It can only be queried by the cloud service API, which provides limited information for verification. Furthermore, attacks on the model are unnoticeable, and the modified model remains accurate performance on normal samples. Specifically, the backdoor is only known by the attacker, and it is hard to detect by the client without knowing the particular triggers. Besides, once there exist undetected attack channels, the remote model can be modified in a manner of constant change.

Our goal is to detect whether the remote model has been modified or not via making queries to the cloud as less as possible. The main idea is to construct samples by using the original model locally and then use the samples to query results from the remote model. A set of query samples used for the verification is denoted as $S_X = \{X_i; i = 1, \ ..., \ k\}$, where k is the number of constructed samples. The client draws one sample $x = X_i$ from S_X to make the query and compares the output of remote model $f(W', x)$ with the output of the original model $f(W, x)$. If outputs are different, then we say the remote model has been modified. Certainly, how precisely the client can compare outputs of the models depends on the format of the remote model's output that is predefined in service schema. For example, when a model is deployed on the cloud platform (e.g., Micro-software Azure) as a service, there is an option for the client to define the data transformation of output before returning it to the end-users. Considering a DNN for classification tasks, we define different situations with the formats of outputs as the following:

1) Predicted label $\hat{y} = argmax_j f_j(W', x)$, where $f_j(W', x)$ is the probability of x predicted

to class j by the model $f(W', x)$.

2) Predicted label $\hat{y}$ with the corresponding predicted class probability $f_j(W', x)$.

3) Top-K predicted labels with probabilities, K is equal or larger than 2.

The difficulties for the clients to verify the result is decreasing from situation 1 to 3. The requirement for the sensitivity of the query sample to weight modification is also relaxing. Specifically, if the comparison of the outputs of a query sample results in a difference in situation 1, the query sample is also sufficiently effective to detect the modified model in the other two conditions.

We aim to construct samples that can handle the most challenging situation. As to how the attacker modifies the remote model is unexpected, the query sample should be sensitive to slight changes. Besides effectiveness, the desired sample should be easy to generate without too much extra effort. It would be better to implement verification privately, thus the sample is expected to be similar to normal inputs such that the cloud service takes it as a usual query.

4.2 Methodology

4.2.1 Intuition

In our scenario, the remote model has been modified; even worst, the modified model remains a similar prediction ability for normal inputs as the original model. As known, DNN learns meaningful representation from training data. It is incapable of interpreting unseen features and how the model explains the unseen feature is uncertain, which means the behavior of models varies on unrelated input. We take advantage of this characteristic and construct a perturbation to trigger such uncertainty, forcing the sample's outputs of $f(W, x)$ and $f(W', x)$ different.

The main idea is to add perturbation to an input that maximizes the randomness of the original model's prediction. The perturbation disturbs the model's recognition of learned patterns and renders the output less confident. We hypothesize that the modified model's output is different from the original model when given an input containing disturbing patterns. We call such data *disturbed sample*.

4.2.2 Generation of Disturbed Samples

Let the input represented by a vector, $f(W, x)$ be the DNN model which expects n-dimensional inputs, $x = (x_1, ..., x_n)$, and outputs $y = (f_1(W, x), ..., f_m(W, x))$. m is the total number of classes. We consider Shannon entropy to express the randomness of the predicted outcome. Given an input x, its entropy $\mathbb{H}(x)$ is calculated as:

$$\mathbb{H}(x) = -\sum_{j=1}^{m} f_j(W, x) log_2 f_j(W, x)$$

,where $f_j(W, x)$ is the probability of x predicted to class j by $f(W, x)$. $\mathbb{H}$ stands for the uncertainty of the input being classified. Higher the $\mathbb{H}$, lower the ability of the model predicting on the input.

We utilize entropy to construct disturbed samples. Drawing a sample from training dataset or testing dataset, denoted as x. The vector $s(x) = (s_1, ..., s_N)$ is perturbation applied on x. Since the disturbed sample x' should look similar to natural input x, the distance between x and x' is constrained by the number of modified dimensions. The goal is to find a perturbation $s(x)^*$ that maximizes the entropy of the disturbed image $\mathbb{H}(x + s(x))$. Hence, it can generally be described as a box-constrained optimization problem:

$$\max_{s(x)^*} \ \mathbb{H}(x + s(x)) \ \ s.t. \ \ \|s(x)\|_0 \leq d,$$

, where d defines the number of dimensions wherein values are perturbed. Explicitly considering the image as an input vector, each pixel stands for one dimension. $d = 1$ represents that only one pixel is modified while other pixels remain unchanged.

As the DNN function is a non-convex function with many local maxima and minima, it is efficient to use a global optimizer to find an optimized solution for our objective function, which is also a non-convex function. Inspired by [83], we adopt the differential evolution (DE) optimization algorithm to generate disturbed samples. As DE is a minimization method, we rewrite our objective function to:

$$\min_{s(x)^*} \; 1 - \mathbb{H}(x + s(x)) \;\; s.t. \;\; \|s(x)\|_0 \leq d,$$

4.2.3 Solution Search using Differential Evolution

Applications of differential evolution in solving real-world problems have been successful across diverse domains since the late 1990s. DE is one of the most popular evolutionary algorithms and primarily used for solving numerical optimization problems. It is designed to be a stochastic direct search method with the ability to handle complex optimization problems, including non-differentiable, nonlinear, and multimodal objective functions. In addition, it is parallel method that is suitable for computation-intensive tasks [84].

Similar to other types of evolutionary algorithms, DE follows general four main stages [85]: (i) initialization of the parameter vectors, (ii) difference vector based mutation, (iii) generation of candidate solutions via crossover, (iv) selection. The operation of (ii),(iii),(iv) repeats until termination criteria meets.

In the experiment section, We use image classification as an application. The task of DE optimization is a search for perturbations, which minimizes our objective function. As the dimension of a perturbation vector is constrained to d. For RGB images, the perturbation

consists of five elements: x-y coordinates and three channels values. Thus, we construct an input parameter vector for DE as a D-dimensional vector, where $D = 5 \times d$. The main stages for generating a disturbed image are described in the following.

Initialization of the Parameter Vector. In the first step, DE starts with a random population of NP d-dimensional vectors, denoted as $X_{i,G} = [x_{1,i,G}, x_{2,i,G}, ..., x_{D,i,G}], i = 1, 2, ..., NP$ (current population). Each parameter vector $X_{i,G}$ is a candidate solution for the optimization problem in current generation. For jth parameter $x_{j,i,G}$, there is a pair of values defining the finite lower and upper bounds. For RGB images with width W and height H, we can define value bounds for x-coordinate corresponding parameter as (1, W), for y-coordinates corresponding parameter as (1, H), for RGB values corresponding parameter as (0, 256). Therefore, minimum and maximum bounds for all parameters can be expressed by $X_{min} = [x_{1,min}, x_{2,min}, ..., x_{D,min}]$ and $X_{max} = [x_{1,max}, x_{2,max}, ..., x_{D,max}]$. The initial population should cover the entire parameter spaces within the search space limited by X_{min} and X_{max}.

Difference Vector Based Mutation. After initialization, one iteration begins with mutation operation. New vectors (mutant vector) are generated by adding a weighted difference between two parameter vectors to a third parameter vector. The three distinct vectors are randomly selected from the current population, called target vectors. Formally, the process of generating a mutant vector can be expressed as:

$$V_{i,G+1} = X_{r_1,G} + F \cdot (X_{r_2,G} - X_{r_3,G})$$

, where indices $r_1,\ r_2,\ r3\ \in \{1, 2, ..., NP\}$ are mutually different, and a float number $F\ \in [0, 2]$ is a factor controlling the amplification of the difference between $X_{r_2,G}$ and $X_{r_3,G}$. For each target vector ($X_{i,G}$) in current population, a mutant vector $V_{i,G+1}$ is constructed. Noted, the $X_{i,G}$ itself is out of consideration for target vectors selection.

Generation of Candidate Solutions via Crossover. Crossover is introduced to augment the diversity of newly generated vectors. Mutant vectors and target vectors form the offspring of the current population together. The offspring parameter vectors are called trial vectors. We adopt a commonly used strategy, binomial method, for operating crossover. A trial vector $U_{i,G+1} = [u_{1,i,G+1}, u_{2,i,G+1}, ..., u_{D,i,G+1}]$ is obtained by exchanging the parameter values of the mutant vector $V_{i,G+1}$ with the target vector $X_{i,G}$ according to:

$$u_{j,i,G+1} = \begin{cases} x_{j,i,G} & rand(0,1) > CR \ and \ j \neq randint, \\ v_{j,i,G+1} & Otherwise \end{cases}$$

, where $CR \in [0, 1]$ is a user-predefined factor indicating the crossover rate. *randint* is a randomly chosen integer between 1 and D. The purpose of *randint* is to make sure $U_{i,G+1}$ remains at least one parameter value from $V_{i,G+1}$. Considering one element $u_{j,i,G+1}$ at a time, uniform random number generator, $rand(0, 1)$, outputs a real number between 0 and 1. In this way, we generate trial vectors from mutant vectors and corresponding target vectors one by one.

Selection. In the last step of the current iteration, DE determines which vectors to survive and become a population of generation G+1 for the next optimization iteration. The criteria is comparing the trial vector and the target vector with regards to the objective function value. The one whose objective function value is smaller will become one solution candidate. Formally, the selection of the next population is obtained by:

$$X_{i,G+1} = \begin{cases} X_{i,G} & Objective_Func(X_{i,G}) < Objective_Func(U_{i,G+1}), \\ U_{i,G+1} & Otherwise \end{cases}$$

DE continues to perform (ii),(iii),(iv) until it reaches the stop criteria. In our experiment, the process stops when it runs up to the predefined maximum number of iteration or when it

hits early stop condition. The size of the population is initialized with a predefined number, and each iteration maintains the same size. More details of the experiment setting will be discussed in the evaluation section. In the end, DE returns an optimized parameter vector with minimum objective function value among all candidate solutions.

4.3 Experiment Setting

4.3.1 Datasets and Original Model $f(W, x)$

We conduct experiments to show the effectiveness of our proposed method on real-world image datasets *CIFAR-10* and *GTSRB Traffic Sign*. Both original models use a convolution neural network. CIFAR-10 has 10 classes with 6000 color images per class. The network for CIFAR-10 consists 6 Conv layers and 3 fully-connected layers. GTSRB has 43 classes with more than 50,000 traffic sign images in total. The network for GTSRB uses 3 Conv layers and 2 fully-connected layers. Both use *relu* activation function for hidden layers and *softmax* activation function for the output layer. In the end, we train one original model per dataset.

Besides, to compare our method to other state-of-the-art, we also use VGG-Face datasets with 2622 classes and pre-trained models[1]. The network architecture adopts VGG16. The watermark trojan trigger is generated from the layer FC6 [20].

4.3.2 Generating Modified Model $f(W', x)$

To mimic different behaviors of the remote that has been modified, we generate three types of modified models: 1) Backdoor models. We train the model by adding a trigger (from [20]) to partial training data. The Backdoor model can classify a trojan input to a target class while makes a correct prediction on normal input. 2) Compressed models. We use pruning and quantization to reduce the size of the original model in the same way as [86]. The

[1]https://github.com/PurduePAML/TrojanNN

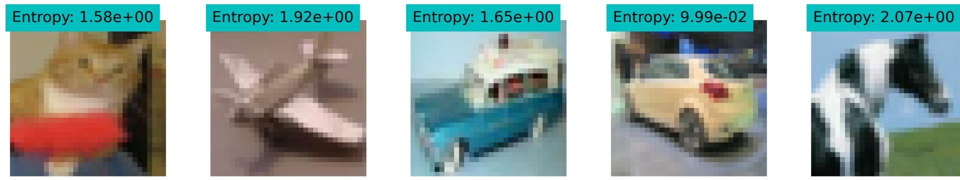
compressed model does not carry any malicious behaviors, and its prediction accuracy stays in an acceptable range depending on the compression rate. 3) Arbitrary weight changed models. Gaussian noise is added to the weights of the original model in the last hidden layer with mean 0 and unit standard deviation. Overall, we train one original model and generate four types of mimic remote models.

4.3.3 Hyper-parameter Setting

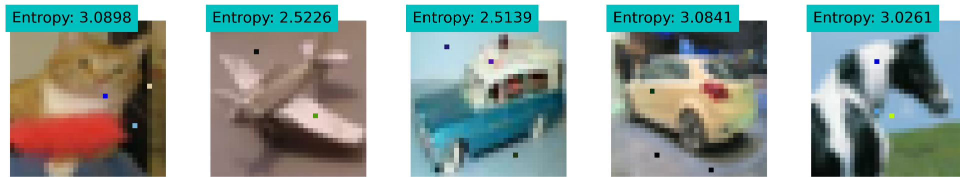
We use the differential evolution algorithm to find the perturbations to construct disturbed images. To make the disturbed images look similar to normal images, we set the number of perturbed pixels to 3 for 32×32 images and 50 for 224×224 images. The maximum number of iteration for DE searching the optimized solution is set to 100 for 32×32 images and 300 for 224×224 images. The initialized size of the population for the generation is 500, and the crossover factor is set to 0.8.

4.3.4 Sample Set S_X for Evaluation

A sample set S_X consists of disturbed images that are used to query the remote model for integrity verification. We randomly sample 2000 data from the testing dataset and then generate a disturbed image for each sample. Next, we rank prediction entropy of disturbed images in descending order and pick the top 400 samples to be elements of S_X. The process of generating disturbed images is a one-time effort. Examples of original testing images and generated disturbed images for CIFAR-10, GTSRB, and VGG-FACE are shown in Figure 4.1, Figure 4.2, and Figure 4.3. The original model's prediction entropy of the image, $\mathbb{H}(x)$, is calculated and displayed in the figure.

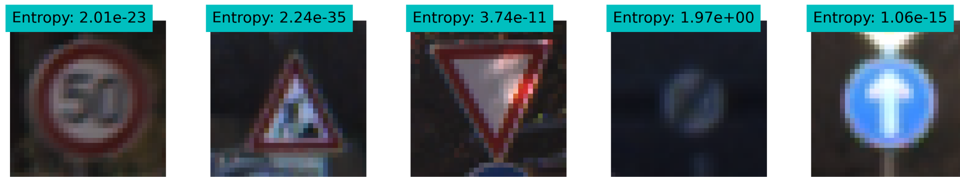


(a) Original Image

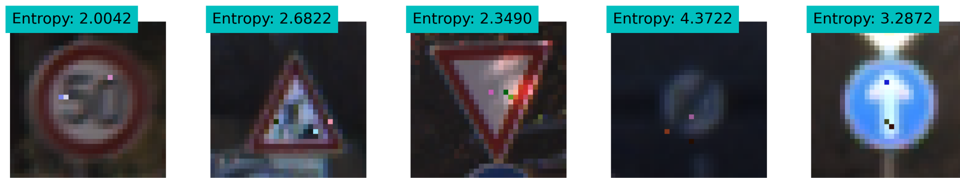


(b) Disturbed Image

Figure 4.1: Cifar Examples



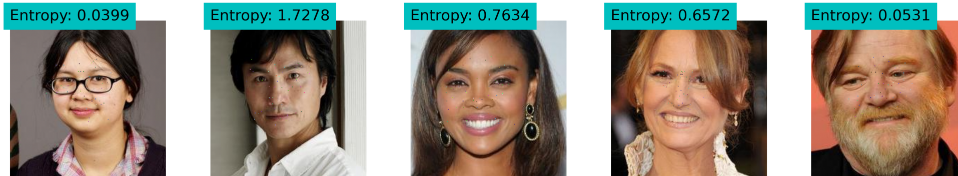
(a) Original Image



(b) Disturbed Image

Figure 4.2: GTSRB Examples

(a) Original Image



(b) Disturbed Image

Figure 4.3: VGG-Face Examples

4.3.5 Experiment Environment

We implement DNN models with Keras API (version 2.3.1) and Tensorflow Backend (version 1.15.2). Experiments run on a server with 28 Xeon CPU E5-2697 2.6GHz processors, 188GB of RAM, and 1 NIVIDIA Quadro M5000 GPU card.

4.4 Evaluation

In this section, we first explain how we test the effectiveness of disturbed images in integrity checking and define the criteria of detection success. Then, we generate different types of modified models described in subsection 4.3.2, and show the corresponding results under different settings of output formats defined in subsection 4.1.2. ***Noted, if the description of the result does not mention output format, the default setting for calculating a detection rate uses the Top-1 predicted label only (output format 1).***

4.4.1 Testing Process and Detection Rate

Given an original model $f(W, x)$ and a mimic remote model $f(W', x)$, randomly drawing K ($K \geq 1$) sample(s) from S_X, we compare the output of $f(W, x)$ and the output of $f(W', x)$ per sample according to the output format. We regard the detection effective for the process as long as at least one sample resulting the output of $f(W, x)$ different from the output of $f(W', x)$. The experiment repeats H times and count the number of times (h) where the detection is effective. We set H to 10,000, which means the step of drawing K sample(s) followed by the output comparison run 10000 times. The **detection rate** is defined as $\frac{h}{H} \times 100$ and the **missing rate** is defined as $(1 - \frac{h}{H}) \times 100$.

To compare the effectiveness of disturbed images and original images, we repeat the same experiment for original images by randomly choose 400 data from the testing dataset and evaluate its detection rate.

4.4.2 Results for Modification Type as Backdoor

The performance of trained models are listed in Table 4.1. ACC represents the prediction accuracy. We prepare a backdoor model that has the same network structure as the original model. For example, the CIFAR-10 backdoor model has 86.94% prediction accuracy on normal inputs while it classifies tampered images to a target class 'automobile' with 100% attack accuracy.

The result of detecting the backdoor model over different drawing number K is shown in Figure 4.4. The X-axis is the number of drawing samples K used for the detection process per experiment, and the y-axis is the average detection rate of over 10,000 experiments. The solid line represents the result of the experiment, where disturbed images are used as verification samples to detect the remote model. When K=1, the average detection rate for CIFAR-10 and GTSRB is 83.26% and 92.61%, respectively. CiFAR-10 can reach 99.9%

when K=3, while GTSRB reaches 99.9% at K=2. The dashed line represents the detection rate with randomly chosen testing images. We observe that disturbed images are useful to detect backdoor types of attack.

We run the same experiments by setting the output as format-2, where the predicted label with two-decimal probability (e.g., 0.88) is available for output comparison. As shown in Table 4.2, both results of output format2 are 100% when only using one disturbed sample. As expected, the more information for output comparison, the better the detection rate.

Dataset	**Original Model**	**Backdoor Model**	
	ACC	ACC	Attack Acc
CIFAR-10	85.50%	86.94%	100%
GTSRB	96.62%	95.05%	97.96%

Table 4.1: Model Information

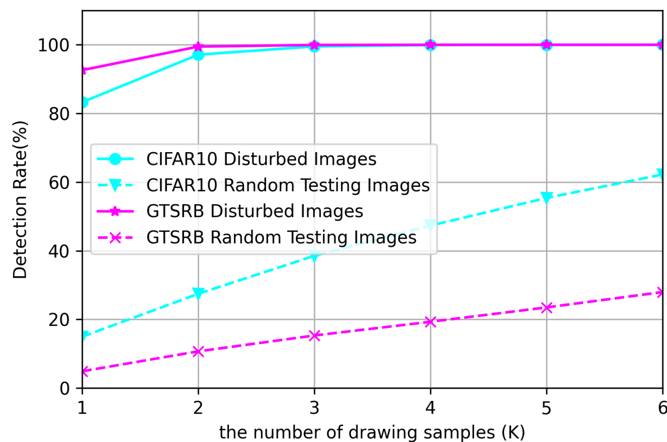


Figure 4.4: Detection Rate of Backdoor Model over Different Number K

4.4.3 Results for Modification Type as Compression

In our scenario, the dishonest provider tries to reduce the service's actual cost while charging the client service fee according to how many resources used for hosting the original model.

Setting	Output Format1	Output Format2
CIFAR-10 Disturbed Images	83.26%	100%
CIFAR-10 Random Testing Images	14.94%	67.74%
GTSRB Disturbed Images	92.61%	100%
GTSRB Random Testing Images	5.69%	5.78%

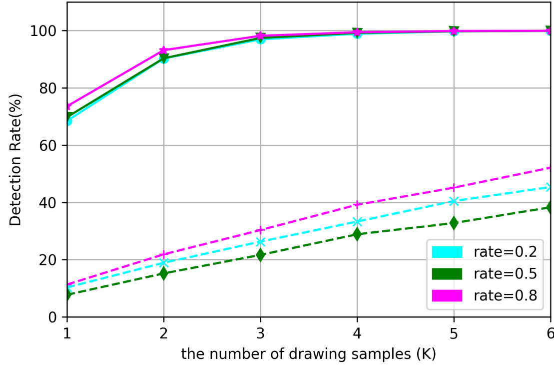
Table 4.2: Calculating Detection Rate by Comparing Output Under Different Format Settings When K=1

We prepare compressed models by parameter pruning and quantization with a compression rate of 0.2, 0.5, and 0.8. Compared to the original model, the decrease of the prediction accuracy is at most 2% for CIFAR-10 and 5% for GTSRB.

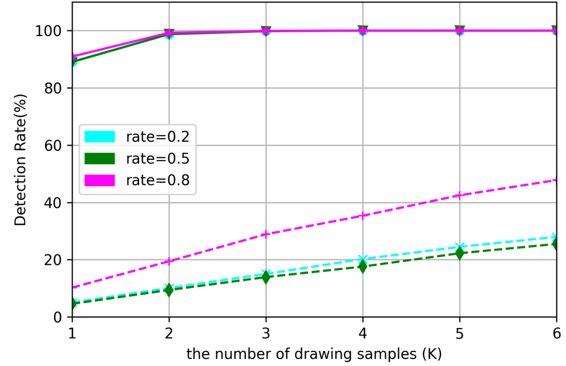
Figure 4.5 shows the detection rates of compressed models with different drawing number K. Solid lines represent the results of using disturbed images, and dashed lines represent the results of using random testing images. Color represents the compressed model at a different rate used as the modified model. For example, the solid red line means that the experiment uses disturbed images as query samples to detect the model with a compression rate of 0.8 over different K. It seems that there is no explicitly different performance of disturbed images detection between rate 0.2 and rate 0.5, while detection rates of rate 0.8 are slightly higher on both datasets. CIFAR-10 and GTSRB average detection rates of all three compression cases can reach over 96.9% and 98.7% when using three and two samples, respectively.

4.4.4 Results for Modification Type as Arbitrary Change

We explore another modification type where the weights of the original model have been arbitrarily changed. As most attacks target at altering the weights at the second-to-the-last layer of the model, we mimic the modification by adding Gaussian noise to partial weights at the second-to-last layer. Figure 4.6 is the detection rate on the model with weights changed ratio 0.1 over different drawing number K. It indicates that we only need less than four disturbed images to detect the integrity breach with 100%.



(a) CIFAR-10 Result



(b) GTSRB Result

Figure 4.5: Detection Rate of the Compressed Model with Different Compression Rates over Different Number K

We also run experiments on weight change, with a ratio ranging from 0.2 to 0.9. Figure 4.7 shows the result of detecting the modified model with different ratios of weights changed by using only one sample (K=1). As the importance of the weights at the layer is different, we repeat the generation of a noise added model 10 times per ratio. Each time we generate a model with a particular ratio and obtain a detection rate. A point at a line shows the average detection rates over 10 different models with the same ratio, and the shadow represents the variance. We observe that the disturbed images detection is effective and stable for different ratio weight changed models. Moreover, we find that random testing images detection becomes comparable to the disturbed images detection as the ratio of weight changed increases, but its deviation is large, especially in CIFAR-10 shown as the wide orange shadow.

4.4.5 Comparison With Sensitive-Sample

We compare our method with Sensitive-Sample fingerprint [87], which is the first to use carefully designed transformed inputs as a defense for DNNs' integrity protection. As their

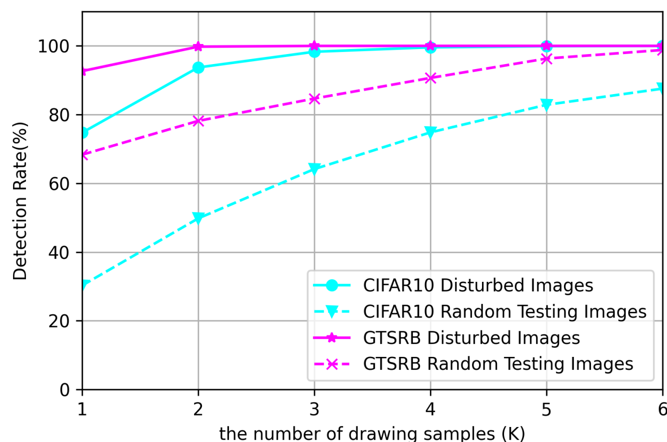
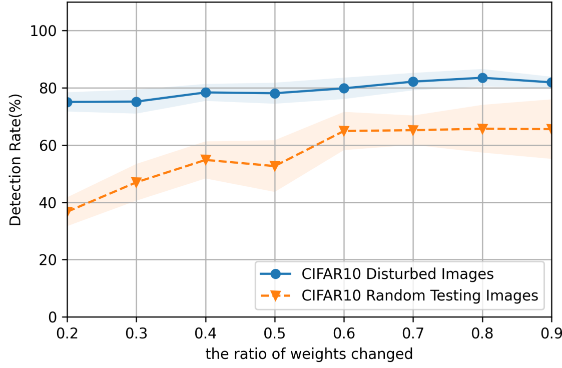


Figure 4.6: Detection Rate of Weights Changed Model with Ratio=0.1 over Different K

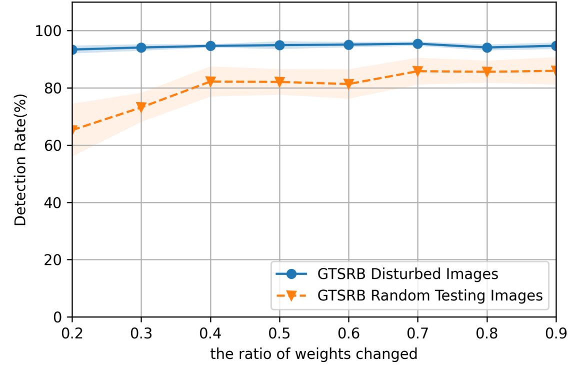
work is not open-source, we try our best to conduct experiments in the same setting as they described in the paper to perform the evaluation. We directly use the pre-trained original model and trojan model from [20] for neural network trojan attack and train other models by ourselves. Table 4.3 shows the missing rates on detecting three types of modification over different drawing number K. For examples, K=1 means we randomly draw one sample from the generated sample set to query the modified model for each of 10,000 experiments. Our method outperforms Sensitive-Sample in scenarios of trojan attack and compression, while the missing rate of poisoning attack is comparable. Our method's missing rate of the poisoning attack is also lower than 1% when K=2. The most improvement is the detection of Model Compression, where users can use two disturbed images to detect integrity breach with less than 7% missing rate.

Modification Types	**Sensitive-Sample** [87]					**Our Method**				
\K	1	2	3	4	5	1	2	3	4	5
Neural Network Trojan Attack	5.93	0.22	0.00	0.00	0.00	3.62	0.09	0.00	0.00	0.00
Error-specific Poisoning Attack	2.20	0.01	0.00	0.00	0.00	7.39	0.53	0.05	0.00	0.00
Model Compression 4x	48.93	15.56	4.72	1.81	0.83	26.43	6.83	1.80	0.56	0.15

Table 4.3: Methods Comparison of Missing rates(%) w.r.t to K on three types of modification



(a) CIFAR-10 Result



(b) GTSRB Result

Figure 4.7: Detection on Model with Different Ratio of Weight Changed (K=1)

4.5 Discussion

We utilize the neural network characteristic of uncertain prediction on unseen features to generate querying samples. When users prepare query samples for the integrity checking, a heuristic way is to choose original testing images that already have substantial prediction entropy value of the original model. Here uses VGG-Face dataset as an example because it contains several not well-collected images, leading to high randomness of the prediction. We calculate prediction entropy for original testing data and sort the entropy in descending order. Then we choose top-50 samples as querying samples to detect the trojan face recognizer. The detection rate can reach 73.9%. Thus, it provides an alternatively heuristic way to prepare query samples.

The verification of DNN model integrity by querying the model with crafted inputs as few as possible is a line of research at an early stage. The current scenario only considers compromised means that target the model itself but not including the downstream or upstream of the model prediction module. Exploration of more types of integrity violations is direct future work.

Existing approaches to verify ML cloud service might not be feasible in our scenarios. It is common to use checksum like MD5 to verify data integrity. However, a dishonest cloud provider can cheat the client and return the hashing of the original model's parameters while still running a compressed model. Verifiable computation based approaches [75, 88] are inapplicable to neural networks where hidden layers use sum pooling or non-polynomial activation function (e.g., ReLU, sigmoid).

4.6 Related Work

Outsourcing the deployed model to the cloud raises a fundamental issue of trust. Many adversarial attacks exploit vulnerabilities of deep learning by inducing particular outputs or behaviors. Attackers attempt to prevent an accurate model‘s outcome, which affects the availability of the ML service [89]. The security concern includes the correctness of computation and integrity of the model. Many work leverage verifiable computation (**VC**) to verify the result of machine learning as a service. In [75], an interactive-proof based framework, SafetyNets, is proposed to assure that the cloud has performed correct computation. The result of model inference returned from the cloud server is attached with a piece of mathematical proof, and the client accepts the outcome as long as the corresponding proof is verified. SafetyNets represent neural networks as arithmetic circuits and utilize check-sum to verify the execution of networks. VeriML [88] aims to check the running of loop iterations during the model training phase, which ensures the fairness of service payments.

In the scenario of checking the integrity of outsourced models which works as a black-box, He et al., [87] is the first to use crafted inputs to detect the integrity violation of deep neural networks (**DNN**) by comparing outputs of local model and remote model on the crafted input. The main idea is to construct inputs that are sensitive to the modification of model parameters. How to generate sensitive samples is described as an optimization problem.

They use a local optimizer to find the sensitive sample that can maximize the difference in the prediction between the original model and the modified model. But only weights in the last layer of the DNN are considered as parameters-of-interest. Similarly, method BCD [90] uses the same objective function as [87], while BCD employs a global optimizer, Bayesian optimization, to find the sensitive sample. Their designed experiment model is a network with only one hidden layer; thus, it might not be feasible to DNN.

Chapter 5

CONCLUSION AND FUTURE WORK

5.1 Dissertation Conclusion

ML model becomes an integral part of modern systems and makes critical decisions for the system's outcome. However, It is likely to fail silently by providing incorrect predictions with high confidence scores, and the failure might cause financial loss or even worse accidents. Moreover, the lifecycle of ML consists of complex components and processes. The Attack surface of ML is large along the lifecycle, which gives the adversary opportunities to poison the dataset and to compromise the model. It is challenging to ensure the safety of ML and enhance the trust of the application.

In this dissertation, we argued that estimating risk for ML solution and preparing for ML task failure is crucial for the development of trustworthy ML-based applications. We considered ML risks from different perspectives and proposed solutions towards reliable applications of ML in two scenarios. From a data perspective, we propose an Automated Contaminated Attribute Localization (ACAL) system for deployed machine learning solutions that can pre-

cisely pinpoint the erroneous attribute value from the input data that caused the incorrect decision. ACAL can automatically quantify each attribute's suspiciousness in the contaminated data and output the rank of the suspicious attributes. The research suggests that error diagnosis is demanding for enhancing the trust of extending ML techniques to safety-critical systems. From the model perspective, we propose an Integrity Checking for Neural Networks approach that can detect compromised models on the cloud with black-box access. We present a novel method to generate integrity-testing samples, which is similar to normal querying samples. The generated samples are evaluated on different types of modifications and shown to be useful to detect the integrity violation. Our research explores potential attacks on the model as a service on the cloud and shows the necessity of integrity checking for ensuring ML safety.

5.2 Future Work

- The current ACAL system has an assumption about knowledge of the expected class that the contaminated input should belong to. We can further extend the current approach to relax such an assumption by adding a layer at the top. The top layer firstly estimate which class could be the fact. Moreover, ACAL has a limitation of supporting contaminated data containing only one faulty attribute. But multiple contaminated attributes might occur. We believe that contamination data with multi-faulty attributes are more likely to behave as a novelty, which tends to be detected by abnormality detection. Detection of contamination data and solutions for locating multi-contaminated attributes will be explored.

- Our current integrity checking method only considers the modification of a neural network model's parameters; therefore, it cannot handle tricky scenarios such as the intended modification of the model's output. We can amplify the definition of model

integrity and address the problem in more complex situations. Under the current assumption, how the model has been modified is unknown. We can detect whether there is a change in the model's parameters or not. To differentiate the types of modification, we need to generate querying samples with different sensitivity to different modifications.

- Currently, we focus on the problems that affect an individual's prediction of the final-state model. Although issues usually raised inevitably in the practice of ML applications, we plan to look for the possibility of precautionary measures that can mitigate prediction task failure in earlier stages.

Bibliography

[1] Matthew B. A. McDermott, Shirly Wang, Nikki Marinsek, Rajesh Ranganath, Marzyeh Ghassemi, and Luca Foschini. Reproducibility in Machine Learning for Health. *arXiv:1907.01463 [cs, stat]*, July 2019.

[2] Li Liu, Wanli Ouyang, Xiaogang Wang, Paul Fieguth, Jie Chen, Xinwang Liu, and Matti Pietikäinen. Deep Learning for Generic Object Detection: A Survey. *International Journal of Computer Vision*, 128(2):261–318, February 2020. ISSN 1573-1405. doi: 10.1007/s11263-019-01247-4.

[3] Lifeng He, Xiwei Ren, Qihang Gao, Xiao Zhao, Bin Yao, and Yuyan Chao. The connected-component labeling problem: A review of state-of-the-art algorithms. *Pattern Recognition*, 70:25–43, October 2017. ISSN 0031-3203. doi: 10.1016/j.patcog.2017.04.018.

[4] Kiri Wagstaff. Machine Learning that Matters. *arXiv:1206.4656 [cs, stat]*, June 2012.

[5] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[6] Luis Perez and Jason Wang. The Effectiveness of Data Augmentation in Image Classification using Deep Learning. *arXiv:1712.04621 [cs]*, December 2017.

[7] Sebastian Schelter, Joos-Hendrik Böse, Johannes Kirschnick, Thoralf Klein, and Stephan Seufert. Automatically Tracking Metadata and Provenance of Machine Learning Experiments. 2017.

[8] Rob Ashmore, Radu Calinescu, and Colin Paterson. Assuring the Machine Learning Lifecycle: Desiderata, Methods, and Challenges. *arXiv:1905.04223 [cs, stat]*, May 2019.

[9] Nimit Sharad Sohoni, Christopher Richard Aberger, Megan Leszczynski, Jian Zhang, and Christopher Ré. Low-Memory Neural Network Training: A Technical Report. *arXiv:1904.10631 [cs, stat]*, April 2019.

[10] Sebastian Schelter, Tammo Rukat, and Felix Biessmann. Learning to Validate the Predictions of Black Box Classifiers on Unseen Data. page 11, 2020.

[11] Shachar Kaufman, Saharon Rosset, and Claudia Perlich. Leakage in data mining: Formulation, detection, and avoidance. page 8, 2011.

[12] Feature Engineering: Secret to data science success. https://community.alteryx.com/t5/Data-Science-Blog/Feature-Engineering-Secret-to-data-science-success/ba-p/545041, March 2020.

[13] Mengchen Liu, Jiaxin Shi, Zhen Li, Chongxuan Li, Jun Zhu, and Shixia Liu. Towards Better Analysis of Deep Convolutional Neural Networks. *arXiv:1604.07043 [cs]*, April 2016.

[14] Jason Brownlee. The Model Performance Mismatch Problem (and what to do about it), April 2018.

[15] kexugit. Machine Learning - Machine Learning with IoT Devices on the Edge. https://docs.microsoft.com/en-us/archive/msdn-magazine/2018/july/machine-learning-machine-learning-with-iot-devices-on-the-edge.

[16] Sebastian Schelter, Felix Bießmann, Tim Januschowski, David Salinas, Stephan Seufert, and Gyuri Szarvas. On Challenges in Machine Learning Model Management. *IEEE Data Eng. Bull.*, 2018.

[17] Jasper Snoek, Yaniv Ovadia, Emily Fertig, Balaji Lakshminarayanan, Sebastian Nowozin, D. Sculley, Joshua Dillon, Jie Ren, and Zachary Nado. Can you trust your model\textquotesingle s uncertainty? Evaluating predictive uncertainty under dataset shift. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d\textquotesingle Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 13969–13980. Curran Associates, Inc., 2019.

[18] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Xiaodong Song. Targeted Backdoor Attacks on Deep Learning Systems Using Data Poisoning. *ArXiv*, 2017.

[19] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain. 2017.

[20] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning Attack on Neural Networks. In *NDSS*, 2018. doi: 10.14722/ndss.2018.23291.

[21] Ligeng Zhu, Zhijian Liu, and Song Han. Deep Leakage from Gradients. *arXiv:1906.08935 [cs, stat]*, December 2019.

[22] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and Harnessing Adversarial Examples. *arXiv:1412.6572 [cs, stat]*, March 2015.

[23] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv:1607.02533 [cs, stat]*, February 2017.

[24] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. DeepFool: A simple and accurate method to fool deep neural networks. *arXiv:1511.04599 [cs]*, July 2016.

[25] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The Limitations of Deep Learning in Adversarial Settings. *arXiv:1511.07528 [cs, stat]*, November 2015.

[26] Shiqing Ma, Yingqi Liu, Guanhong Tao, Wen-Chuan Lee, and Xiangyu Zhang. NIC: Detecting Adversarial Samples with Neural Network Invariant Checking. In *Proceedings 2019 Network and Distributed System Security Symposium*, San Diego, CA, 2019. Internet Society. ISBN 978-1-891562-55-6. doi: 10.14722/ndss.2019.23415.

[27] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. DeepXplore: Automated Whitebox Testing of Deep Learning Systems. *Proceedings of the 26th Symposium on Operating Systems Principles - SOSP '17*, pages 1–18, 2017. doi: 10.1145/3132747.3132785.

[28] Tom B. Brown, Dandelion Mané, Aurko Roy, Martín Abadi, and Justin Gilmer. Adversarial Patch. *arXiv:1712.09665 [cs]*, May 2018.

[29] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership Inference Attacks Against Machine Learning Models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 3–18, May 2017. doi: 10.1109/SP.2017.41.

[30] Ahmed Salem, Yang Zhang, Mathias Humbert, Pascal Berrang, Mario Fritz, and Michael Backes. ML-Leaks : Model and Data Independent Membership Inference Attacks and Defenses on Machine Learning Models. (February), 2019. ISSN 189156255X.

[31] Matt Fredrikson, Eric Lantz, Somesh Jha, Simon Lin, David Page, and Thomas Ristenpart. Privacy in Pharmacogenetics: An End-to-End Case Study of Personalized Warfarin Dosing. *USENIX Security Symposium*, 2014.

[32] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 1322–1333, Denver, Colorado, USA, October 2015. Association for Computing Machinery. ISBN 978-1-4503-3832-5. doi: 10.1145/2810103.2813677.

[33] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Stealing Machine Learning Models via Prediction APIs. *arXiv:1609.02943 [cs, stat]*, October 2016.

[34] Zhao Zhang, Evan R. Sparks, and Michael J. Franklin. Diagnosing Machine Learning Pipelines with Fine-grained Lineage. In *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing - HPDC '17*, pages 143–153, Washington, DC, USA, 2017. ACM Press. ISBN 978-1-4503-4699-3. doi: 10.1145/3078597.3078603.

[35] João Felipe Pimentel, Leonardo Murta, Vanessa Braganholo, and Juliana Freire. noWorkflow: A tool for collecting, analyzing, and managing provenance from python scripts. *Proceedings of the VLDB Endowment*, 10(12):1841–1844, August 2017. ISSN 21508097. doi: 10.14778/3137765.3137789.

[36] Shiqing Ma, Yousra Aafer, Zhaogui Xu, Wen-Chuan Lee, Juan Zhai, Yingqi Liu, and Xiangyu Zhang. LAMP: Data provenance for graph based machine learning algorithms through derivative computation. In *Proceedings of the 2017 11th Joint Meeting on*

Foundations of Software Engineering - ESEC/FSE 2017, pages 786–797, Paderborn, Germany, 2017. ACM Press. ISBN 978-1-4503-5105-8. doi: 10.1145/3106237.3106291.

[37] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. *arXiv:1602.04938 [cs, stat]*, August 2016.

[38] Wenbo Guo, Dongliang Mu, Jun Xu, Purui Su, Gang Wang, and Xinyu Xing. LEMNA: Explaining Deep Learning based Security Applications. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security - CCS '18*, pages 364–379, Toronto, Canada, 2018. ACM Press. ISBN 978-1-4503-5693-0. doi: 10.1145/3243734.3243792.

[39] Scott M Lundberg and Su-In Lee. A Unified Approach to Interpreting Model Predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc., 2017.

[40] Scott M. Lundberg, Bala Nair, Monica S. Vavilala, Mayumi Horibe, Michael J. Eisses, Trevor Adams, David E. Liston, Daniel King-Wai Low, Shu-Fang Newman, Jerry Kim, and Su-In Lee. Explainable machine-learning predictions for the prevention of hypoxaemia during surgery. *Nature Biomedical Engineering*, 2(10):749–760, October 2018. ISSN 2157-846X. doi: 10.1038/s41551-018-0304-0.

[41] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic Attribution for Deep Networks. *arXiv:1703.01365 [cs]*, June 2017.

[42] Sanjay Krishnan, Michael J. Franklin, Kenneth Y. Goldberg, and Eugene Wu. BoostClean: Automated Error Detection and Repair for Machine Learning. *ArXiv*, 2017.

[43] Sebastian Schelter, Dustin Lange, Philipp Schmidt, Meltem Celikel, Felix Biessmann, and Andreas Grafberger. Automating large-scale data quality verification. *Proceedings*

of the VLDB Endowment, 11(12):1781–1794, August 2018. ISSN 21508097. doi: 10.14778/3229863.3229867.

[44] Nick Hynes, D. Sculley, and Michael Terry. The Data Linter: Lightweight Automated Sanity Checking for ML Data Sets. 2017.

[45] Shrey Shrivastava, Dhaval Patel, Anuradha Bhamidipaty, Wesley M. Gifford, Stuart A. Siegel, Venkata Sitaramagiridharganesh Ganapavarapu, and Jayant R. Kalagnanam. DQA: Scalable, Automated and Interactive Data Quality Advisor. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 2913–2922, December 2019. doi: 10.1109/BigData47090.2019.9006187.

[46] Vraj Shah and A. Suneel Kumar. The ML Data Prep Zoo: Towards Semi-Automatic Data Preparation for ML. In *DEEM'19*, 2019. doi: 10.1145/3329486.3329499.

[47] Eric Breck, Marty Zinkevich, Neoklis Polyzotis, Steven Whang, and Sudip Roy. Data Validation for Machine Learning. In *Proceedings of SysML*, 2019.

[48] William Caicedo-Torres and Jairo Gutierrez. ISeeU: Visually interpretable deep learning for mortality prediction inside the ICU. *Journal of Biomedical Informatics*, 98:103269, October 2019. ISSN 1532-0464. doi: 10.1016/j.jbi.2019.103269.

[49] Andreja Stojić, Nenad Stanić, Gordana Vuković, Svetlana Stanišić, Mirjana Perišić, Andrej Šoštarić, and Lazar Lazić. Explainable extreme gradient boosting tree-based prediction of toluene, ethylbenzene and xylene wet deposition. *Science of The Total Environment*, 653:140–147, February 2019. ISSN 0048-9697. doi: 10.1016/j.scitotenv.2018.10.368.

[50] María Vega García and José L. Aznarte. Shapley additive explanations for NO2 forecasting. *Ecological Informatics*, 56:101039, March 2020. ISSN 1574-9541. doi: 10.1016/j.ecoinf.2019.101039.

[51] Arthur Zimek and Erich Schubert. Outlier Detection. In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of Database Systems*, pages 1–5. Springer, New York, NY, 2017. ISBN 978-1-4899-7993-3. doi: 10.1007/978-1-4899-7993-3_80719-1.

[52] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *KDD*, 1996.

[53] Dheeru Dua and Casey Graff. UCI machine learning repository. 2017.

[54] Christian Camilo Urcuqui Lopez and Andres Navarro Cadavid. Machine learning classifiers for android malware analysis. *2016 IEEE Colombian Conference on Communications and Computing (COLCOM)*, 2016.

[55] Rami M. Mohammad, Fadi Thabtah, and Lee McCluskey. An assessment of features related to phishing websites using an automated technique. In *2012 International Conference for Internet Technology and Secured Transactions*, pages 492–497, December 2012.

[56] Tadhg Nagle, Thomas C. Redman, and David Sammon. Only 3% of Companies' Data Meets Basic Quality Standards. *Harvard Business Review*, September 2017. ISSN 0017-8012.

[57] David Baehrens, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert Müller. How to Explain Individual Classification Decisions, August 2010.

[58] Mark Craven and Jude W. Shavlik. Extracting Tree-Structured Representations of Trained Networks. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8*, pages 24–30. MIT Press, 1996.

[59] Vicente Iván Sánchez Carmona, Tim Rocktäschel, Sebastian Riedel, and Sameer Singh. Towards Extracting Faithful and Descriptive Representations of Latent Variable Models. In *AAAI Spring Symposia*, 2015.

[60] Anupam Datta, Shayak Sen, and Yair Zick. Algorithmic Transparency via Quantitative Input Influence: Theory and Experiments with Learning Systems. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 598–617, May 2016. doi: 10.1109/SP.2016.42.

[61] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning Important Features Through Propagating Activation Differences. April 2017.

[62] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation. *PLoS ONE*, 10(7), July 2015. ISSN 1932-6203. doi: 10.1371/journal.pone.0130140.

[63] Jasper van der Waa, Marcel Robeer, Jurriaan van Diggelen, Matthieu Brinkhuis, and Mark Neerincx. Contrastive Explanations with Local Foil Trees. *arXiv:1806.07470 [cs, stat]*, June 2018.

[64] Shubham Rathi. Generating Counterfactual and Contrastive Explanations using SHAP. *arXiv:1906.09293 [cs, stat]*, June 2019.

[65] J.A. Jones, M.J. Harrold, and J. Stasko. Visualization of test information to assist fault localization. In *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002*, pages 467–477, May 2002. doi: 10.1145/581396.581397.

[66] W. Eric Wong, Vidroha Debroy, and Dianxiang Xu. Towards Better Fault Localization: A Crosstab-Based Statistical Approach. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(3):378–396, May 2012. ISSN 1094-6977, 1558-2442. doi: 10.1109/TSMCC.2011.2118751.

[67] Claire Le Goues, ThanhVu Nguyen, Stephanie Forrest, and Westley Weimer. GenProg: A Generic Method for Automatic Software Repair. *IEEE Transactions on Software Engineering*, 38(1):54–72, January 2012. ISSN 1939-3520. doi: 10.1109/TSE.2011.104.

[68] Shiqing Ma, Yingqi Liu, Wen-Chuan Lee, Xiangyu Zhang, and Ananth Grama. MODE: Automated neural network model debugging via state differential analysis and input selection. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering - ESEC/FSE 2018*, pages 175–186, Lake Buena Vista, FL, USA, 2018. ACM Press. ISBN 978-1-4503-5573-5. doi: 10.1145/3236024.3236082.

[69] Xiaofei Xie, Lei Ma, Felix Juefei-Xu, Minhui Xue, Hongxu Chen, Yang Liu, Jianjun Zhao, Bo Li, Jianxiong Yin, and Simon See. DeepHunter: A coverage-guided fuzz testing framework for deep neural networks. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2019, pages 146–157, Beijing, China, July 2019. Association for Computing Machinery. ISBN 978-1-4503-6224-5. doi: 10.1145/3293882.3330579.

[70] Xiang Gao, Ripon K Saha, Mukul R Prasad, and Abhik Roychoudhury. Fuzz Testing based Data Augmentation to Improve Robustness of Deep Neural Networks. page 12, 2020.

[71] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv:1312.6199 [cs]*, February 2014.

[72] Naveed Akhtar and Ajmal Mian. Threat of Adversarial Attacks on Deep Learning in Computer Vision: A Survey. *arXiv:1801.00553 [cs]*, February 2018.

[73] Reuben Feinman, Ryan R. Curtin, Saurabh Shintre, and Andrew B. Gardner. Detecting Adversarial Samples from Artifacts. *arXiv:1703.00410 [cs, stat]*, November 2017.

[74] Xingjun Ma, Bo Li, Yisen Wang, Sarah M. Erfani, Sudanthi Wijewickrema, Grant Schoenebeck, Dawn Song, Michael E. Houle, and James Bailey. Characterizing Adversarial Subspaces Using Local Intrinsic Dimensionality. *arXiv:1801.02613 [cs]*, March 2018.

[75] Zahra Ghodsi, Tianyu Gu, and Siddharth Garg. SafetyNets: Verifiable Execution of Deep Neural Networks on an Untrusted Cloud. *arXiv:1706.10268 [cs]*, June 2017.

[76] Ling Huang, Anthony D. Joseph, Blaine Nelson, Benjamin I.P. Rubinstein, and J. D. Tygar. Adversarial machine learning. In *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*, AISec '11, pages 43–58, Chicago, Illinois, USA, October 2011. Association for Computing Machinery. ISBN 978-1-4503-1003-1. doi: 10.1145/2046684.2046692.

[77] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Xiaodong Song. Robust Physical-World Attacks on Deep Learning Visual Classification. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018. doi: 10.1109/CVPR.2018.00175.

[78] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How To Backdoor Federated Learning. *ArXiv*, 2018.

[79] Song Han, Jeff Pool, John Tran, and William Dally. Learning both Weights and Connections for Efficient Neural Network. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 1135–1143. Curran Associates, Inc., 2015.

[80] Hien Van Nguyen, Kevin Zhou, and Raviteja Vemulapalli. Cross-Domain Synthesis of Medical Images Using Efficient Location-Sensitive Deep Network. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, Lecture Notes in Computer Science, pages 677–684, Cham, 2015. Springer International Publishing. ISBN 978-3-319-24553-9. doi: 10.1007/978-3-319-24553-9_83.

[81] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. Fixed point optimization of deep convolutional neural networks for object recognition. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1131–1135, April 2015. doi: 10.1109/ICASSP.2015.7178146.

[82] Wenlin Chen, James T Wilson, Stephen Tyree, Kilian Q Weinberger, and Yixin Chen. Compressing Neural Networks with the Hashing Trick. page 10, 2015.

[83] Jiawei Su, Danilo Vasconcellos Vargas, and Sakurai Kouichi. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 23(5): 828–841, October 2019. ISSN 1089-778X, 1089-778X, 1941-0026. doi: 10.1109/TEVC.2019.2890858.

[84] Rainer Storn and Kenneth Price. Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces. *Journal of Global Optimization*, 11 (4):341–359, December 1997. ISSN 1573-2916. doi: 10.1023/A:1008202821328.

[85] Swagatam Das and Ponnuthurai Nagaratnam Suganthan. Differential Evolution: A Survey of the State-of-the-Art. *IEEE Transactions on Evolutionary Computation*, 15 (1):4–31, February 2011. ISSN 1941-0026. doi: 10.1109/TEVC.2010.2059031.

[86] Song Han, Huizi Mao, and William J. Dally. Deep Compression: Compressing

Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *arXiv:1510.00149 [cs]*, February 2016.

[87] Zecheng He, Tianwei Zhang, and Ruby Lee. Sensitive-Sample Fingerprinting of Deep Neural Networks. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4724–4732, Long Beach, CA, USA, June 2019. IEEE. ISBN 978-1-72813-293-8. doi: 10.1109/CVPR.2019.00486.

[88] Lingchen Zhao, Qian Wang, Cong Wang, Qi Li, Chao Shen, Xiaodong Lin, Shengshan Hu, and Minxin Du. VeriML: Enabling Integrity Assurances and Fair Payments for Machine Learning as a Service. *arXiv:1909.06961 [cs]*, September 2019.

[89] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael Wellman. Towards the Science of Security and Privacy in Machine Learning. *arXiv:1611.03814 [cs]*, November 2016.

[90] Deepthi Praveenlal Kuttichira, Sunil Gupta, Dang Nguyen, Santu Rana, and Svetha Venkatesh. Detection of Compromised Models Using Bayesian Optimization. In Jixue Liu and James Bailey, editors, *AI 2019: Advances in Artificial Intelligence*, volume 11919, pages 485–496. Springer International Publishing, Cham, 2019. ISBN 978-3-030-35287-5 978-3-030-35288-2. doi: 10.1007/978-3-030-35288-2_39.