MEng Individual Project Final Report

Imperial College London

Department of Computing

Mitigating Evasion Attacks against Machine Learning Systems through Dimensionality Reduction and Denoising

Author:
Ziyi Bao

Supervisor:
Dr. Luis Muñoz-González

June 18, 2018

Acknowledgements

Thank you Luis for the passion, inspirations and guidance you have shared with me. Most importantly, thank you for trusting me to implement our ideas.

Contents

Chapter 1

Introduction

As machine learning techniques are reaching maturity, an increasing number of fields are seeking to employ them to solve problems that would be extremely difficult to solve through explicit programming. Current and emerging applications include security (e.g. detection of malware, network intrusion, spam and fraudulent transactions), autonomous vehicles, data analytics and recommender systems [21].

A machine learning system can be seen a function approximator—it approximates the *true function* reasonably well but is not perfect. The field of *adversarial machine learning* studies the worst-case performance of these systems, which an adversary can exploit in order to cause unexpected behavior. While much of past research focused on their flaws in security applications such as spam filtering, Szegedy et al. [30] discovered that they also have critical vulnerabilities in image recognition: By applying calculated, imperceptible perturbation to an image, they could cause a system to misclassify it with high confidence. This discovery has gained much attention from other security researchers and gave the field of adversarial machine learning a new momentum, as it has severe implications on emergent applications such as self-driving cars. Figure 1.1 shows that an attacker can trick a neural network into misclassifying a stop sign as a yield sign by slightly altering the image. Moreover, these adversarial examples are still effective after being printed out on paper and processed by a camera used by a machine learning system [18] (see Figure1.2).

Numerous defense techniques have been proposed by researchers over the last few years. Initial attempts focused on making machine learners more robust against attacks, but after a lack of convincing results, the focus shifted towards detecting these attacks instead. However, Carlini et al. [5] have recently demonstrated that most of the state-of-the-art detection mechanisms are ineffective, especially when used in classification tasks involving complex images.

We evaluate two techniques which could enhance the robustness of a classifier:

- *Dimensionality reduction* reduces the feature space of data, leaving the adversary a smaller attack surface. This will restrict them to less ineffective attacks. Some recent papers [34][2] on robustness enhancement using DR appear somewhat promising. However, the evaluation of the techniques are flawed in these papers (explained later), and we wish to investigate them properly in this project.

- *Denoising* is a technique used to remove noise in noisy images. It can be a novel way to remove adversarial noise from input before it is fed to a machine learning system.

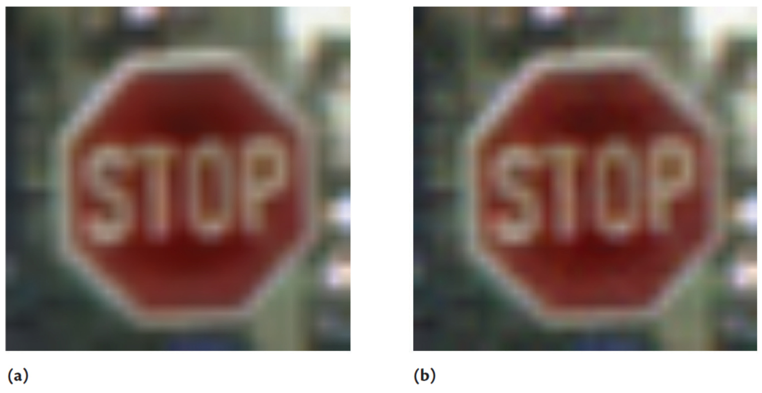


Figure 1.1: (a) shows a a normal image of a stop sign. (b) is an altered version of (a) that a classifier sees as a yield sign [21]

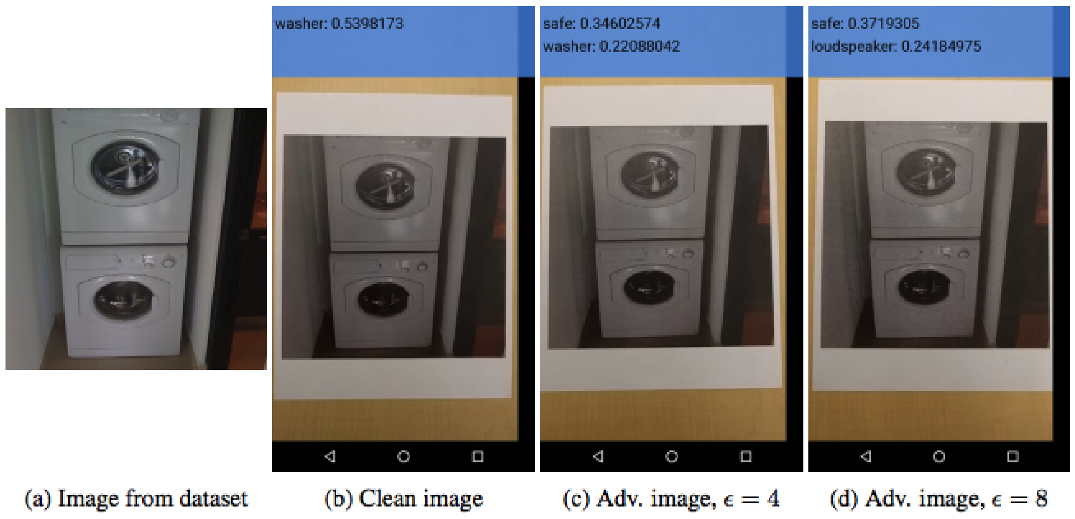


Figure 1.2: The first image (a) is a clean photo of a washer. After printing it out (b), the classifier recognizes it as a washer through a smartphone camera. Perturbing the image at different strengths, the classifier labels the print-outs (c) and (d) as safes. [18]

1.1 Contributions

The contributions of this work are as follows:

- We devised a methodology to examine the security implications of applying feature selection techniques to machine learning systems.
- Using this methodology, we proved, against literature, that feature selection techniques such as Lasso can enhance a machine learning classifier's resistance against adversarial samples.
- We devised a novel feature selection technique and evaluated its impact on classifier performance and security
- We investigated how autoencoders, which are popular for dimensionality reduction, affect the security of classifiers when both are used in conjunction.
- We tested whether denoising autoencoders could make classifiers more secure by remove adversarial perturbation from the input.
- We devised and evaluated a novel technique to extend protection against adversarial input through noising and denoising.

Chapter 2

Background

2.1 Machine Learning

Informally, machine Learning is a field of study in which computers learn without being explicitly programmed [26].

A more formal definition: Given experience E (e.g. experience of playing games of chess), class of tasks T (e.g. task of playing chess) and performance measure P (e.g. probability of winning the next game), a computer program learns from E with respect to T and P, if its performance at tasks in T, measured by P, improves with E [22].

The three main types of machine learning algorithms are supervised learning, unsupervised learning and reinforcement learning. Only the former is relevant to this project.

In supervised learning, the algorithm starts in the *training phase*, in which it learns from a *training data* set of which each training sample maps an input to a *label* (correct output). The job of the algorithm is to learn the relationship between the input and output such that in the *inference phase*, it is able to predict the output for a given input.

Problems in supervised learning can be split into two categories: *regression* and *classification* problems. In both types of problems, the input is a set of features; they differ only in the kind of output they produce: A regression problem deals with predicting a continuous output value, whereas in a classification problem, there is a discrete set of classes and we try to determine which class an instance belongs to. We will only consider classification problems in this project, as they are more relevant in adversarial machine learning.

2.1.1 Hypothesis Function

We can describe our goal as learning a hypothesis function $h : X \to Y$ such that $h(x)$ predicts the corresponding y. x is a vector of n features values and y is a single value (although in some machine learning models, y can be a vector of multiple output values, but here we focus on the simple case of a single output value).

For classification, the hypothesis function can be represented by different machine learning models. The parameters of these models are tuned to minimize a *cost function* (aka *loss* function) that measures the difference between the target values and the values predicted by h. A widely used cost function is the squared error [23]:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2$$

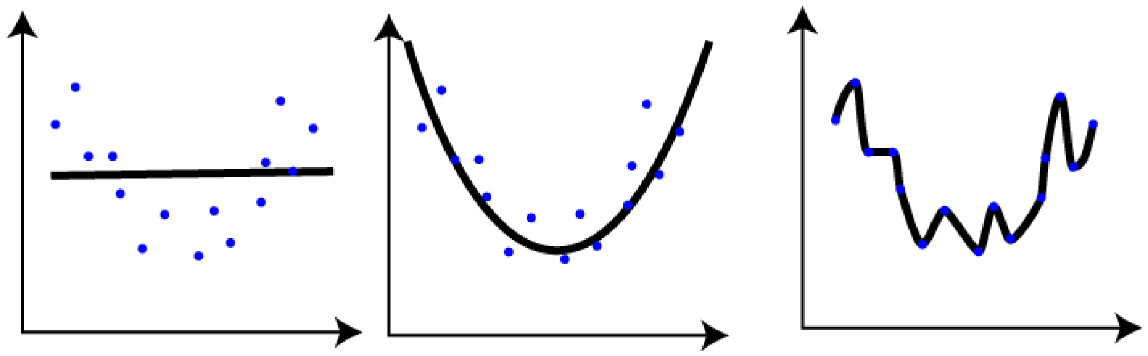


Figure 2.1: The blue dots represent the training data points and the curves are some possible hypothesis functions. The first curve is underfit (does not fit the data at all). The second function describes the trend of the data reasonably well, whereas the third one is overfit. [16]

where θ is the vector of parameters (called *weights*) used to tune the hypothesis function h_θ, m is the number of training samples, x_i is the feature vector of the ith training sample and y_i is the corresponding label.

2.1.2 Gradient Descent

Finding the minimum of a function analytically is often very difficult, as a closed form solution may not exist. But if a function is differentiable, then gradient descent, an iterative algorithm, can be used to find its minimum. It works by picking a starting point and in each iteration, it takes steps in the negative direction of the gradient:

```
$a \leftarrow$ starting point
$\gamma \leftarrow$ learning rate
**repeat**
    $a = a - \gamma \nabla f(a)$
**until** convergence achieved
```

Picking a bad starting point and learning rate can cause it to fail to converge. In non-convex functions, it can converge to a local minimum.

Gradient descent is widely used for minimizing loss functions when training machine learning systems. It is also applicable to other optimization problems, as discussed later.

2.1.3 Overfitting

A common problem in machine learning is overfitting. This happens when the hypothesis function lies too close to the data points, usually as a result of high function complexity and large weights. Hence, it can appear uneven (see Plot 3 in Figure 2.1) and does not *generalize* well on new data.

Some of the techniques discussed later can mitigate this issue.

2.1.4 Linear Classifiers

The simplest way to solve a classification problem is to use a *linear classifier*, which has a linear hypothesis function. Its operation can can be visualized as a hyperplane separating instances of different classes (see Plot A in Figure 2.2).

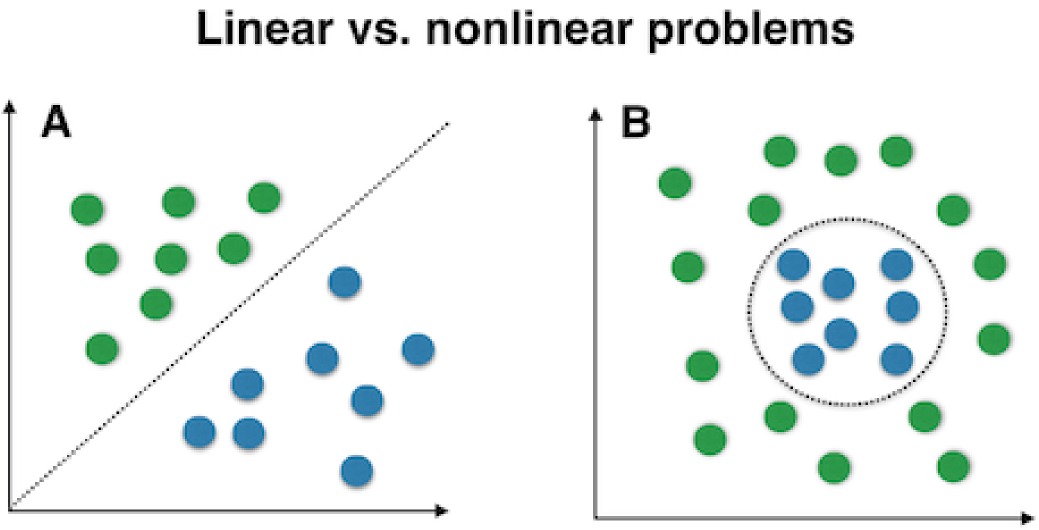


Figure 2.2: The green and blue dots represent instances of two different classes. Plot A shows a scenario where they are *linearly separable*, whereas in plot B, they are not.

In the first part of the project, we will be using the linear classifiers below due to their simplicity and popularity in many applications such as spam filtering.

2.1.4.1 Perceptron

A perceptron (or single-layer perceptron) is the most basic *neural network* (explained in more detail later) and its output can be written as:

$$h(x) = \sigma(w^\intercal x + b)$$

where x the vector of input features, w is the vector of weights, and $\sigma(z) = 1$ if $z > 0$, -1 otherwise. Therefore, a perceptron can separate instances into two classes, where each possible output corresponds to one class. This makes it a *binary* classifier.

2.1.4.2 Logistic Regression

Another very popular binary classifier is logistic regression. Suppose we have classes 0 and 1. The hypothesis function is defined as

$$h(x) = P(y = 1|x) = \frac{1}{1 + e^{-w^\intercal x}}$$

It formulates a classification problem as a regression problem by returning the probability that an instance belongs to class 1. Intuitively, we can say that if $h(x) \geq 0.5$, the predicted class $y = 1$. Otherwise, if $h(x) < 0.5$, $y = 0$.

In order to use a binary classifier such as logistic regression for classification problems involving three or more classes, we can use the *one-vs-all* algorithm: Suppose we have n different classes $1, ..., n$. We train n classifiers $h_1(x), h_2(x), ..., h_n(x)$ such that $h_i(x)$ outputs the confidence of an instance belonging to the ith class [23]. The predicted class is then

$$\arg\max_i h_i(x)$$

2.1.5 Nonlinear Classifiers

Consider the case where instances of different classes are not linearly separable (see Plot B in Figure 2.2). To use a linear classifier for such a classifications task, one would have to introduce "artificial" features that are linear combinations of the original features. This can become intractable when the feature space is large. There is a class of nonlinear classifiers that use only the original feature vector and hence solve such problems far more efficiently.

2.1.5.1 Multilayer Feed-Forward Neural Network

The multilayer feed-forward neural network (aka multilayer perceptron) is an extremely popular machine learning architecture that differs from a single-layer perceptron in that it has additional (*hidden*) layers of nodes (*neurons*) between the input layer and the output layer. One such network with more than one hidden layer is often referred to as a *deep neural network*. Using nonlinear activation functions (for example sigmoid) for neurons in the hidden layers allows a neural network to approximate nonlinear functions.

Each node in layer k is connected to all nodes in layer $k-1$ and $k+1$. Each connection is associated with a weight and the jth node in layer k outputs

$$y_j^{(k)} = \sigma(\sum_i^n y_i^{(k-1)} w_{ji}^{(k-1)})$$

where σ is some *activation function*, $\sum_i^n y_i^{(k-1)} w_{ji}^{(k-1)}$ the *net input function*, $y_i^{(k-1)}$ is the output of the ith node in layer $k-1$ and $w_{ji}^{(k-1)}$ is the weight associated with the connection between this node and the jth node in layer k (see Figure 2.3 for visualization).

The most popular way to solve multiclass classification problems involving n classes with a neural network is to have n output nodes, where the jth output has the softmax activation function [20]

$$\sigma(z_j) = \frac{e^{z_j}}{\sum_i^n e^{z_i}}$$

where z_j is the result of the net input function applied to the inputs of the jth output node. The above function returns the probability of the instance belonging to class j. Clearly, the values of all outputs sum to 1.

We will consider deep neural networks in the second part of this project as they are extremely common but more sophisticated than linear classifiers.

2.2 Dimensionality Reduction

Often, it is possible to reduce the dimensionality of the input space with little impact on the prediction accuracy. Dimensionality reduction techniques can be categorized into *feature selection* and *feature extraction* techniques, which are explained below.

2.2.1 Feature Selection

Feature selection aims to discard a subset of the input features that contribute little to the prediction accuracy. This reduces the complexity of a model, making it

- easier to train,

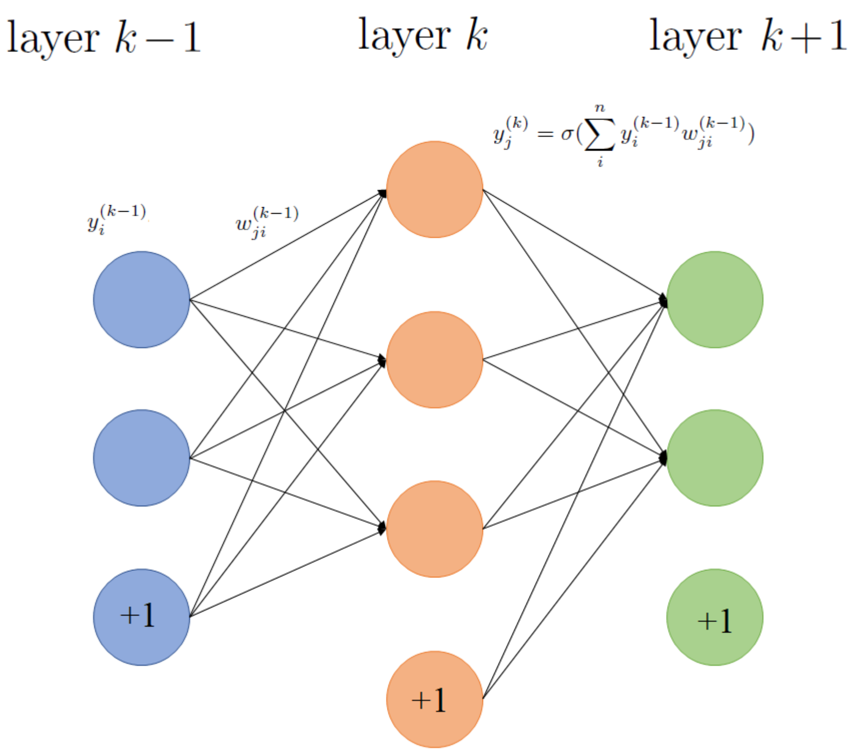


Figure 2.3: 3 middle layers of a deep neural network. Nodes marked with +1 are *bias* nodes, which are always activated, and the weights associated with a bias node represents the constant term in a function (b in $f(x) = ax + b$)

- easier to interpret,
- less prone to overfitting [17].

There are three types of feature selection methods:

- *Filter* methods score each feature using a statistical measure independent of the classifier. The features to be discarded are the ones with the lowest scores.
- *Wrapper* methods create many models with different subsets of features and select the best one based on predictive accuracy.
- *Embedded* methods select the most useful features during the creation process of the model. Two of the most popular examples are LASSO and Ridge regression, both of which will be used in this project due to the fact that they are model specific, which yields better classifiers, and are not overly computationally expensive to tune. Furthermore, both are well understood and the results relatively easy to analyze.

2.2.1.1 Ridge Regression

Ridge regression is primarily a means to mitigate overfitting. Suppose all features are normalized (scaled to be in [0,1]), it performs *L2 regularization* by taking into account the sum of squares of weights in the objective function, such that the objective becomes:

$$\min_w C^{RIDGE}(w) = RSS(w) + \lambda \|w\|_2^2$$

where RSS(w) $= \sum_{i=1}^{n}(y_i - w^\intercal x_i)^2$ is the residual sum of squares and $\lambda \geq 0$ is the 'tuning parameter', expressing how much we wish to penalize the magnitude of the weights, i.e. the weights decrease as lambda increases. In practice, multiple models are generated with various values of λ and then compared with one another to find the one that best balances low training error and good generalization ability.

One way to use ridge regression for feature selection is to view features with the lowest associated weights as the least significant, since the features are normalized. We can then select a weight threshold such that features with weights below the threshold are removed.

It should be noted that the weights can never become zero unless $\lambda = \infty$. Furthermore, if two or more features are highly correlated, then it makes sense to keep only one of these features. However, ridge regression will distribute weights somewhat equally among them, which could result in all these features being removed or kept. In contrast, LASSO does not suffer from this problem and automatically selects the best features to keep.

2.2.1.2 LASSO

Similar to ridge regression, LASSO adds a penalty term to the objective function, performing *L1 regularization*:

$$\min_{w} C^{LASSO}(w) = RSS(w) + \lambda \|w\|_1$$

Since the penalty term is the sum of the absolute value of weights, the optimal solution might see some of the weights set to 0 (sparse model) with large enough λ. The sparsity of the model increases with the magnitude of λ. Features associated with 0 weights are clearly good candidates for removal. Hence, LASSO automatically performs feature selection.

2.2.2 Feature Extraction

Rather than simply discarding a subset of features, feature extraction methods map input variables into lower dimensional space. One such extremely well-known technique is principal component analysis (PCA).

2.2.2.1 Principal Component Analysis (PCA)

Given a set of samples such that each sample $x_i \in \mathbb{R}^d$, the goal of PCA is to linearly project these samples onto a new basis such that each project sample $y_i \in \mathbb{R}^k, k \leq d$ and the variance of the projections on the new basis are maximized.

For convenience sake, we first center the data: $x_i = x'_i - m$, where m is the mean of the features. The optimization problem can be formulated as [8]:

$$\arg\max_{W} tr(W^\intercal S_t W)$$

$$\text{subject to } W^\intercal W = I$$

where W is the projection matrix such that $y_i = W^\intercal x_i$, tr(A) is the trace of matrix A and $S_t = \frac{1}{n} X X^\intercal$ is the covariance matrix.

In the solution to the above problem, W contains as columns the eigenvectors of S_t. Sorted by their associated eigenvalues, the first k eigenvectors capture the most variance and are called the *principal directions*. The projected features are known as *principal components*. To perform dimensionality reduction to k dimensions, we take the first k eigenvectors and construct $U_k = [u_1, ..., u_k]$. $U_d^T X$ gives us the projected data with k principal components.

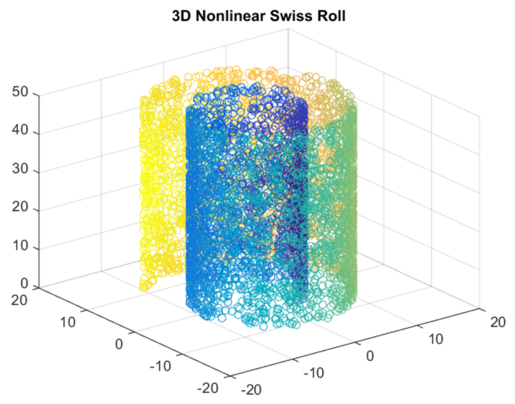


Figure 2.4: A swiss roll in 3D space [28]

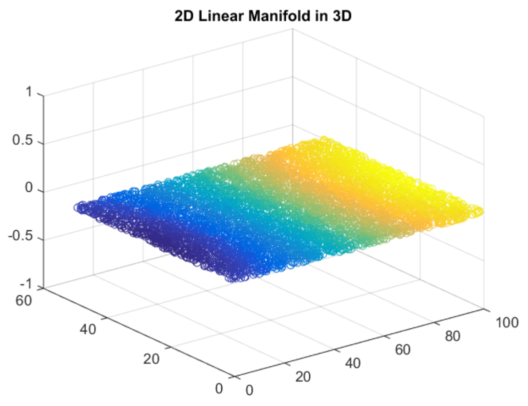


Figure 2.5: Points of swiss roll in Figure 2.4 mapped onto a flat surface [28]

We will be using PCA in the first part of the project, as it is one of the most well-known and effective feature extraction methods, it will allow us to interpret the results more easily than some of the other, more intricate methods.

2.2.3 Nonlinear Dimensionality Reduction: Autoencoders

If data points lie near a nonlinear structure, linear dimensionality reduction methods cannot be used without losing important information. Figure 2.4 shows an example with points lying on a 3D swiss roll, which is a 2D *manifold*, as it can be "unfurled" to produce a flat surface (see Figure 2.5), meaning each point on this manifold is mappable to a point on the unfurled surface (local homeomorphism). Clearly, the manifold is of lower dimensionality compared to its enclosing space. Hence, the goal of nonlinear dimensionality reduction is to learn a manifold onto which we can project the data.

2.2.3.1 Autoencoder (AE)

An autoencoder is a neural network that attempts to approximate the identity function. The network consists of an encoder function

$$f(x) = y = s(Wx + b)$$

where s is a nonlinear function such as the sigmoid function, and a decoder function which reconstructs the input:

$$g(y) = s'(W'y + b')$$

The autoencoder is trained to minimize the reconstruction error:

$$\min L(x, g(f(x)))$$

where L is a loss function such as the mean square error.

In its simplest form (see Figure 2.6), the network has an input and output layer of the same dimension and a single hidden layer whose output is the *code* $(f(x))$. An autoencoder can be used to perform dimensionality reduction if this middle layer has lower dimension than the input layer, creating a bottleneck that forces the network to learn only the most useful properties of the data [13].

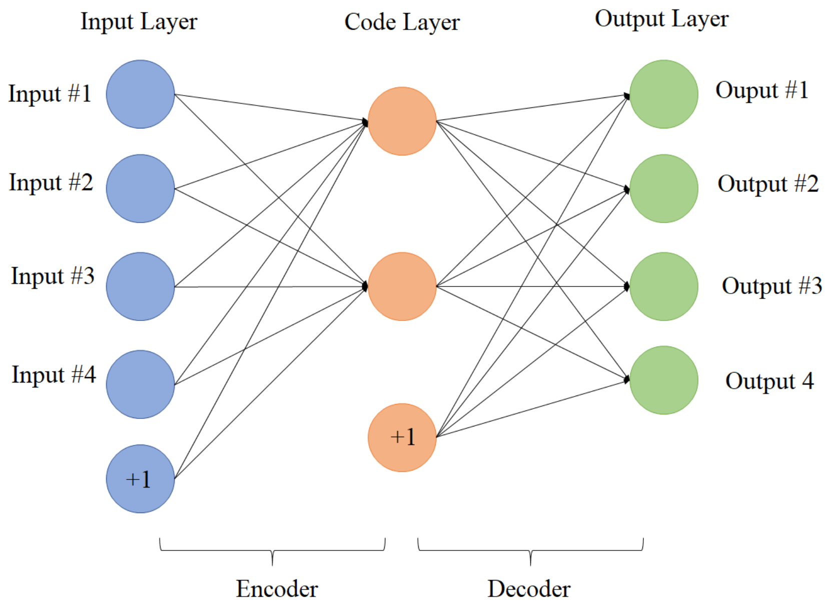


Figure 2.6: An autoencoder with a single hidden layer.

2.2.3.2 Denoising Autoencoder (DAE)

Certain variants of the traditional autoencoder can perform more interesting tasks such as denoising the input. Suppose $\widetilde{x}$ obtained from the distribution $q_d(\widetilde{x}|x)$ is a corrupted version of the initial input x. The task is to reconstruct the clean input x from $\widetilde{x}$. This gives the following training objective:

$$\min L(x, g(f(\widetilde{x})))$$

2.2.3.3 Stacked Denoising Autoencoder (SDA)

Denoising Autoencoders can be "stacked" such that the output code of one DAE serves as input to the one higher on the stack (see Figure 2.7). This has been shown to be an effective pre-training (weight initialization) method [32].

An SDA can be trained by first training the bottom layer as usual to obtain f_1. Then, the next layer gets as clean input $c_1 = f_1(x)$ and has the training objective

$$\min L(c_1, g_2(f_2(q_d(c_1))))$$

where q_d is the corruption function. This process gets repeated with $f_2(c_1)$ as input to the next layer and so on.

Afterwards, the network can be further fine-tuned by using its highest level output as input to a separate supervised learning algorithm such as logistic regression. The whole system can then be trained to minimize the prediction error on a supervised task.

We will experiment with denoising autoencoders in the second part of this project, as they have been proven to be effective at removing noise [32] and one can see adversarial perturbation as noise.

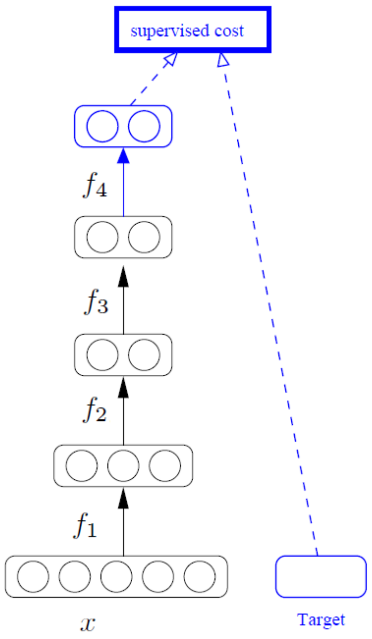


Figure 2.7: A stacked denoising autoencoder [32]

2.3 Adversarial Machine Learning

Adversarial Machine Learning is an emerging field of study that has gained a lot of traction in the recent years. It seeks to strengthen machine learning techniques (most prominently, classifiers) against adversaries who wish to "fool" or weaken them. Numerous papers have been published to reveal vulnerabilities in machine learning algorithms of all sorts including state-of-the-art ones such as convolutional neural networks. Some of these vulnerabilities arise from the assumption that the all data samples are from a *stationary* (unchanging) *distribution*. For example, an adversary can take advantage of this by crafting input data from different distributions. [15]

In order to study adversarial machine learning, it is crucial to provide a classification of the various aspects of adversaries and their attacks. Huang et al. [15] introduced a taxonomy for classifying attacks against online learning algorithms, along with two models for modeling an adversary's capabilities. Some of the terms introduced are found under different names in recent papers, which we will remark on.

2.3.1 Attack Taxonomy

The taxonomy by Huang et al. [15] categorizes any attack using three properties: *influence*, *security violation* and *specificity*.

2.3.1.1 Influence

Influence describes the extent to which the attacker can influence the machine learning system.

- *Causative*: A causative attack (more recently known as *poisoning* attack) influences the training process by inserting specially crafted, malicious training samples into the training set. This is done to degrade the performance of the resulting classifier.

- *Exploratory*: An exploratory attack does not affect the training process. Instead, it aims to discover information about the system, which it can exploit.

2.3.1.2 Security Violation

This property describes the nature of the violation caused by an attacker.

- *Integrity*: An integrity attack results in a classifier misclassifying instances as normal (false negatives). Exploratory integrity attacks are known as *evasion attacks* (explained in detail later), which are the focus of this project as they are highly relevant in real world applications.
- *Availability*: In availability attacks, both malicious and benign instances are misclassified, rendering the classifier useless (denial of service).
- *Privacy*: Privacy violation refers to the act of obtaining sensitive information from the learner such as training data or user information.

2.3.1.3 Specificity

Specificity defines the attackers focus.

- *Targeted*: A targeted attack seeks to degrade a classifier's performance w.r.t. a single or a small subset of instances.
- *Indiscriminate*: Indiscriminate attacks do not have specific targets and have more general goals such as degrading the overall performance of a classifier.

2.3.2 Evasion Attacks

Evasion attacks (or exploratory integrity attacks) are attacks in which malicious samples are altered to evade detection by a classifier (cause misclassification). These altered samples are called *adversarial examples/samples.* Evasion attacks are extremely common in security-related tasks such as spam filtering, malware detection and biometric recognition.

More formally, the adversary tries to find a sample x^* in X, obtained by minimally altering original attack sample x in X, such that x^* evades detection. The optimal evasion can be formulated as [34]:

$$x^* = \arg\min_{x'} d(x', x)$$

$$\text{subject to } g(x') < 0$$

where $d(x', x)$ is the distance between the altered sample and the original sample, and g is the classifier's discriminant function such that the classification function $f(x) = sign(g(x))$ returns -1 and $+1$ for legitimate and malicious samples respectively.

The L_p distance $\|x - x'\|_p$ is widely used as the distance function. Below are the most commonly used L_p distances [6]:

- L_0 distance measures the number of features changed.
- L_1 distance measures the "Manhattan distance". It encourages modifications to be concentrated on a small number of features.

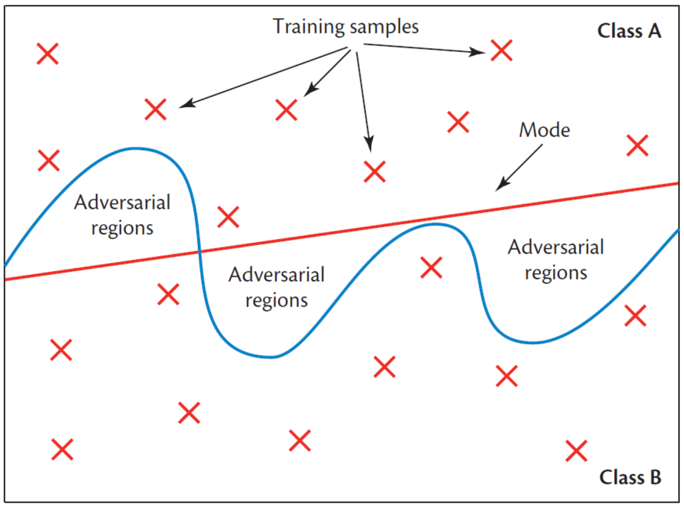


Figure 2.8: The blue curved line represents the *true decision boundary*, whereas the red straight line is the decision boundary learned by the model. The distance between the two boundaries is the space of *adversarial regions* [21].

- L_2 distance measures the Euclidean distance. It can remain small when many features are slightly altered.
- L_∞ distance is the maximum change to any of the features. Hence, it is unaffected by the total number of features changed. Typically, an attack constrained by the L_∞ distance changes all features.

In tasks involving more than two classes such as image recognition, the definition of evasion attacks can be extended to include attacks that aim to make the classifier simply misclassify an instance (*untargeted* attack) or misclassify it as a specific class (*targeted* attack) [2].

The existence of adversarial examples can be explained by the fact that it is almost impossible for a classifier to learn the true decision boundary, and the difficulty increases with the data complexity. The errors around the learned decision boundary constitute *adversarial regions* which an attacker can exploit [21] (see Figure 2.8).

In the literature, attacks are divided into *white-box* and *black-box* attacks. In the former case, the attacker has perfect or near-perfect knowledge of the target model, which can include its architecture, parameters and training data. This allows an attacker to carry out worse-case attacks. In a black-box attack, the adversary has limited to no knowledge about the model. This is the more realistic scenario and requires an attacker to train a surrogate model that approximates the target model, which can still be effective due to the *transferability* property: it has been shown that an adversarial example that evades one classifier can usually also evade another, even one of a different architecture [12].

2.3.2.1 Fast Gradient Sign Method

One efficient white-box attack that serves as a basis for many later proposed attacks is the Fast Gradient Sign Method (FGSM) [12]. It has been shown to be effective in evading image classifiers. The rationale behind the method is that, given large enough input feature space, applying small perturbations to all features should result in sufficiently large change to the output such that the altered instance will be classified differently. This is true in the case of a linear classifier. Goodfellow et al. [12] hypothesize that neural networks are prone to the same weakness, as they often learn relatively linear functions. FGSM maximizes the difference in output by taking advantage of the gradients, producing the adversarial example:

$$x' = x + \epsilon sign(\nabla_x J(\theta, x, y))$$

where ϵ controls the magnitude of the perturbation. This attack is constrained by the $\|\cdot\|_\infty$ norm to ensure that the largest change to a pixel remains unnoticeable.

They find that this technique can trick a variety of models into misclassifying images. For a maxout network and a convolutional maxout network, the error rates are above 85% even at low perturbation rates.

2.3.2.2 C&W L_2 Attack

A state-of-the-art L_2 attack algorithm which generates targeted adversarial examples and can be adapted to defeat many proposed defenses was introduced by Carlini and Wagner [5]. It was derived empirically by experimenting with various loss functions.

Given a neural network with logits Z (inputs to the softmax function). The attack uses gradient descent to solve

$$\min_{x'} \|x' - x\|_2^2 + c \cdot l(x')$$

where l is the loss function and t is the target class:

$$l(x') = \max(\max\{Z(x')_i : i \neq t\} - Z(x')_t, -\kappa)$$

$max\{Z(x')_i : i \neq t\} - Z(x')_t$ measures the difference between t and the next-most-likely class. Hence, the parameter κ controls the confidence of adversarial examples, which increases as κ increases.

By performing binary search on c, one can find an approximation of the adversarial sample with the minimum distance to the original samples.

For evaluation, they adapt the attack to evade 10 techniques used to detect adversarial examples. Each detection method is applied to deep neural networks trained on MNIST (handwritten digits) and CIFAR-10 (color images in 10 classes) datasets. In the case of MNIST, they find that the attack is able to evade most detectors with moderately distorted samples. In contrast, successful adversarial examples for CIFAR-10 are indistinguishable from the original samples.

2.3.3 Defenses against Evasion attacks

Various defense techniques against evasion attacks have been proposed in the recent years and they can be divided into two categories: *robustness enhancement* techniques and *detection* techniques. The former attempt to classify adversarial examples correctly, whereas the latter seek to detect them instead. While early research focused on robustness enhancements, they were ineffective. Hence, the focus shifted towards detection mechanisms. [5].

Some papers [3][33][34][2] analyze the effect of dimensionality reduction on the robustness of classifiers. We describe the relevant parts of each paper below and explain why some of them are flawed.

2.3.3.1 On Sparse Feature Attacks in Adversarial Learning

Wang et al. [33] devise a game between a regularized classifier and an white-box attacker in which they take turns responding to each other's actions: the attacker starts by applying worst-case perturbations to positive samples. Then the classifier responds by re-adjusting its weights to minimize loss when classifying the altered dataset. The attacker generates new adversarial examples based on the changed weights and so on.

For their experiments, they use both ℓ_1 (LASSO) and ℓ_2 regularized classifiers. They expect that the former will perform better. To prove this, they compare the robustness of both against an adversary in a non-game setting. The classifiers are trained with the Spambase database. For each positive sample, the adversary randomly selects 20% of the features and alters them based on the weights of the classifier under attack (see Algorithm 1). Their results in Figure 2.9 show the performance of the ℓ_1 regularized learner to deteriorate more gracefully compared to the ℓ_2 alternative.

There are some critical flaws in this experiment we have to highlight:

- The attacker algorithm is extremely unrealistic for white-box attacks; an attacker with knowledge of the weights would never select random features to modify. Instead, they would focus on features with the largest weights.
- If a feature with weight = 0 is selected, the remaining budget gets decreased although in this case, the attacker does nothing. It is unknown how sparse the tested model is. But if it has a significant portion of features without weight, this should at least partially explain why the ℓ_1 classifier appears to be more robust.
- It would be useful to see how the ℓ_1 model would fare against an unregularized (and hence, unprotected) model. This would show if LASSO leads to models inherently more robust in an adversarial setting.

Algorithm 1 Robustness evaluation under sparse feature attack [33]

Input: $\{x_i, y_i\}_{i+1}^{npos}$: original positive dataset; $w \in \mathbb{R}^{d+1}$: feature weights; $\{c \in \mathbb{N} | (0 < c \leq d)\}$: number of features to be changed; $\{\delta \in \mathbb{R} | (0 < \delta < 1)\}$: attack strength
Output: accuracy of classifier on $\{x_i^*, y_i\}_{i+1}^{npos}$

```
1:  // Randomly select $c$ features of the data and index in vector $\mathbf{a} = \{0 < a_k \leq d\}_{k=1}^{c}$
2:  for $i : npos$ do
3:      $x_i^* \leftarrow x_i, k \leftarrow 1$
4:      while $k \leq c$ do
5:          if $w_{a_k} > 0$ then
6:              $x_{a_k}^* \leftarrow x_{a_k}^*(1 - \delta)$
7:          else
8:              if $w_k == 0$ then
9:                  do nothing
10:             else
11:                 $x_{a_k}^* \leftarrow x_{a_k}^*(1 + \delta)$
12:         $k \leftarrow k + 1$
13: Evaluate classifier using w on changed dataset $\{x_i^*, y_i\}_{i+1}^{npos}$
```

2.3.3.2 Security Evaluation of Pattern Classifiers under Attack

Biggio et al. [3] evaluate empirically the security of pattern classifiers at design phase. They pose a few design scenarios and evaluate the robustness of classifiers under different designs. One of the scenarios involves choosing between a support vector machine (SVM)

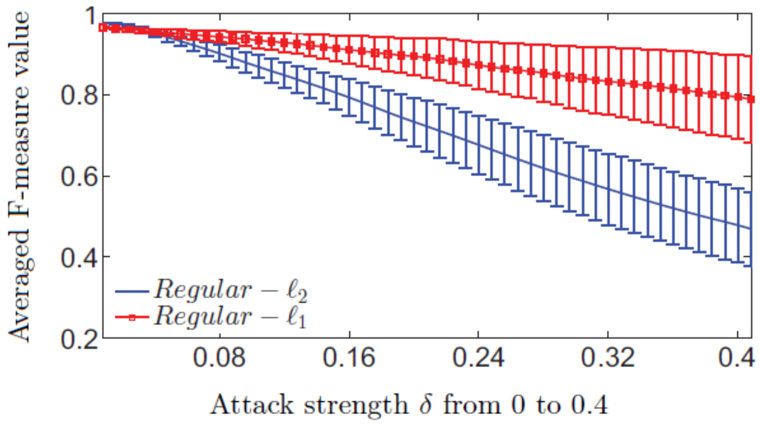


Figure 2.9: ℓ_1 classifier measured against ℓ_2 classifier on F-measure (accuracy) at varying attack strengths [33]

with a linear kernel and logistic regression (LR) classifier for spam filtering. Additionally, feature selection is considered.

For the experiment, they use the TREC 2007 dataset containing both legitimate and spam emails. SpamAssassin tokenization is used extract binary features from the training set, denoting the occurrence of specific words. They utilize a filter feature selection method based on information gain and select four feature subsets with size 1000, 2000, 10000 and 20000. One SVM and one SVM are trained on each feature subset. The attacker has perfect knowledge and commits indiscriminate integrity violation by maximizing the percentage of spam misclassified as legitimate under a given budget. As performance measure, they use the area under the receiver operating characteristic curve (AUC) in the range [0,0.1] (AUC can be interpreted as "The expectation that a uniformly drawn random positive is ranked before a uniformly drawn random negative."): $AUC_{10\%} = \int_0^{0.1} TP(FP)dFP$, where TP is the true positive error rate and FP the false positive rate. They justify this choice by stating that false positives (legitimate emails misclassified as spam) are more harmful than false negatives (spam misclassified as legitimate).

The results of this experiment are in shown Figure 2.10. It can be seen that for the same feature subset, the LR classifier always outperforms the SVM. More importantly, $AUC_{10\%}$ generally decreases as the size of the feature subset decreases. At first glance, it appears that feature selection has a negative impact on the classifier robustness. Below we list reasons why this might not, in general, be the case:

- Biggio et al. [3] use a filter feature selection method, which often produces less optimal models than wrapper and embedded methods.

- It is unclear when robustness of the classifiers are maximized, as we do not know the original number of features and the experiments are not sufficiently granular. For example, if there are in total 100,000 features and it can be shown that a model with all 100,000 features is less robust than a model with only 30,000 features, then we would have proof that the feature selection method they used in fact enhances robustness.

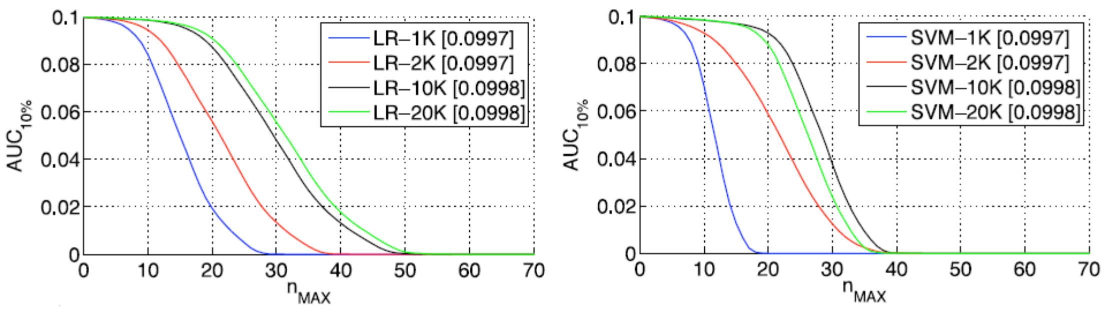


Figure 2.10: $AUC_{10\%}$ achieved on the corrupted testing dataset as a function of n_{max}, the maximum number of features the attacker is allowed to alter [3]

2.3.3.3 Adversarial Feature Selection Against Evasion Attacks

Zhang et al. [34] propose a method to improve the security of binary classifiers (e.g. malware detectors) by exploiting wrapper-based feature selection. They pose an optimization problem that seeks to maximize both the generalization capability and security of a classifier:

$$\theta^* = \arg\max_{\theta} G(\theta) + \lambda S(\theta)$$

$$\text{subject to} \sum_{k=1}^{d} \theta_k = m$$

where $\theta \in \{0,1\}^d$ is a binary valued vector representing whether each feature has been selected. $G(\theta)$ is the classifier's generalization capability which can be estimated with its classification accuracy. $S(\theta)$ is the security term that measures the average minimum number of modifications required for a malicious sample to evade detection. λ controls the tradeoff between generalization capability and security. m specifies the number of features modified by the adversary.

The optimization problem can be approximately solved using wrapper-based feature selection methods such as forward selection and backward elimination. Zhang et al. [34] outline an algorithm for either approach (Algorithm 2).

They evaluate the defense on spam filtering and PDF malware detection examples. For spam filtering, they compare the traditional forward feature selection algorithm with the adversarial version, using a linear SVM. For malware detection, they use traditional and adversarial backward feature elimination with an SVM with RBF kernel.

Both experiments yield similar results (Figure 2.11 shows results for spam filtering): The traditional feature selection methods decrease the classifier's security as m decreases. In contrast, the adversarial methods achieve maximum security when m is relatively small. However, in the case of a perfect knowledge attacker, the maximum security achieved by adversarial methods is only slightly higher than that of an unprotected system (1).

Below we list some potential flaws in their experiments:

- They use greedy wrapper feature selection algorithms which are known to produce less optimal models compared to embedded methods.
- In preparation for the experiment involving spam filtering, they reduced the feature set from more than 20,000 to 500 features using a filter feature selection method. However, Biggio et al. [3] have applied the same filter method to the same dataset

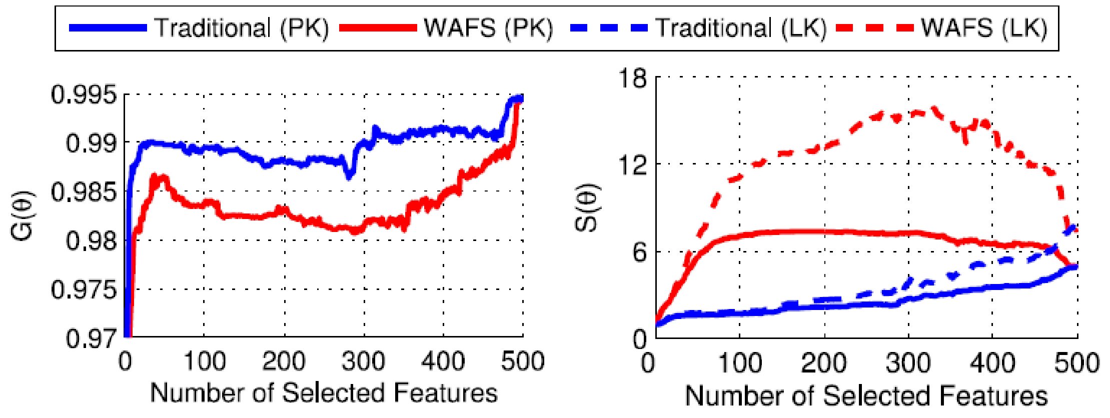


Figure 2.11: Left plot: $G(\theta)$ in the absence of attack; Right plot: $S(\theta)$ under attack for both perfect knowledge (PK) and limited knowledge (LK) attack scenarios as a function of feature subset size [34]

and shown that it negatively impacts the classifier's robustness as the number of features drops below 10,000 (see "Security Evaluation of Pattern Classifiers under Attack").

- They use number of words changed as the security term, which may not be a suitable measure, as the total number of features vary across models. Instead, they should use some normalized measure (e.g. (# features modified) / (# total features)).
- They only use datasets with binary features. Using complex datasets (e.g. images) would allow more nuanced analysis, such as *how much* a feature is changed, rather than only *if* it is changed.

Even if their evaluation is reliable, (1) shows that their adversarial feature selection methods are insufficient as primary means of defense. Rather, they should only be considered when one intends to apply feature selection to a classifier in the first place, but does not wish to diminish its robustness.

2.3.3.4 Enhancing Robustness of Machine Learning Systems via Data Transformations

Bhagoji et al. [2] propose a straightforward robustness enhancement method that uses linear transformation techniques (Algorithm 3).

In their research, they chose PCA for Select() with the rationale that principal components with low variances are usually associated with higher weights, and since the optimal attack perturbation for a linear classifier is a multiple of the weight vector, the principal components with the largest coefficients are the ones to be exploited. By removing these components, the attacker is limited to less optimal perturbations.

They additionally experiment with "anti-whitening" transformation which is based on the principal components of the training data. Let

$$XX^\intercal = U\Lambda U^\intercal$$

, then the transformation matrix is

$$B = \Lambda^{\frac{c}{2}} U \text{ for some } c > 0$$

Algorithm 2 Wrapper-Based Adversarial Feature Selection (WAFS), With Forward Selection (FS) and Backward Elimination (BE) [34]

```
Input: $\mathcal{D} = \{x^i, y^i\}_{i=1}^n$: the training set; $\lambda$: the trade-off parameter; $m$: the number of selected features
Output: $\boldsymbol{\theta} \in \{0,1\}^d$: the set of selected features
 1: $\mathcal{S} \leftarrow \emptyset, \mathcal{U} \leftarrow \{1, ..., d\}$
 2: repeat
 3:     for each feature $k \in \mathcal{U}$ do
 4:         $\mathcal{F} \leftarrow \mathcal{S} \cup \{k\}$ for FS ($\mathcal{F} \leftarrow \mathcal{U} \backslash \{k\}$ for BE)
 5:         $\boldsymbol{\theta} \leftarrow 0$, and then $\theta_j \leftarrow 1$ for $j \in \mathcal{F}$
 6:         Estimate $G_k(\boldsymbol{\theta})$ and $S_k(\boldsymbol{\theta})$ using cross-validation on $\mathcal{D}_{\boldsymbol{\theta}} = \{x^i_{\boldsymbol{\theta}}, y^i\}_{i=1}^n$
 7:     $\lambda' \leftarrow \lambda(\max_k S_k)^{-1}$ (i.e., rescale $\lambda$)
 8:     $k^* = \arg\max_k(G_k(\boldsymbol{\theta}) + \lambda' S_k(\boldsymbol{\theta}))$
 9:     $\mathcal{S} \leftarrow \mathcal{S} \cup \{k^*\}, \mathcal{U} \leftarrow \mathcal{U} \backslash \{k^*\}$
10: until $|\mathcal{S}| = m$ for FS ($|\mathcal{U}| = m$ for BE)
11: $\mathcal{F} \leftarrow \mathcal{S}$ for FS ($\mathcal{F} \leftarrow \mathcal{U}$ for BE)
12: $\boldsymbol{\theta} \leftarrow 0$, and then $\theta_j \leftarrow 1$ for $j \in \mathcal{F}$
13: return $\boldsymbol{\theta}$
```

Anti-whitening serves to exaggerate the disparity between the variances of the components. Instead of cutting off low variance components, it increases the price of accessing them.

They conduct seven experiments with the traditional PCA and one using anti-whitening. For PCA experiments, they compare models with different reduced dimensions, and for anti-whitening, models with different c values are compared. The "best" model is selected in each experiment and their results are summarized in Figure 2.12. A 2x reduction in adversarial success rates can be observed across different attack strategies with only a modest reduction in the classification accuracy.

Algorithm 3 LTrain [2]

```
Input: $\boldsymbol{X}$: centered training data; Train: training algorithm; Select: selects a linear transformation of data based on some properties (e.g. Select = TopPrincipalComponents$(k)$=
Output: $f$: new classifier trained on the transformed training set
 1: Select the linear transformation $\boldsymbol{B}$ = Select$(\boldsymbol{X})$
 2: Compute the transformed training set $\boldsymbol{BX}$
 3: Train classifier $f^{aux}$ = Train$(\boldsymbol{BX})$
 4: Let $f = (x \mapsto f^{aux}(\boldsymbol{BX}))$
```

Data set	Classifier	Attack type	Defense type and parameter	Robustness improvement	Accuracy reduction
MNIST	Linear SVM	Classifier mismatch	PCA (80)	25×	0.22%
MNIST	Linear SVM	White-box (optimal)	PCA (80)	6×	0.22%
MNIST	FC100-100-10	White-box (FG)	PCA (40)	1.5×	0.76%
MNIST	FC100-100-10	White-box (FGS)	PCA (40)	2.2×	0.76%
MNIST	FC100-100-10	White-box (Opt.)	PCA (40)	1.7×	0.76%
MNIST	FC200-200-200-10	Arch. mismatch (FC100-100-10)	PCA (40)	2.4×	0.85%
MNIST	FC100-100-10	White-box (FG)	Anti-whiten (2)	1.7×	0.15%
HAR	Linear SVM	White-box (optimal)	PCA (70)	1.5×	2.3%

Figure 2.12: Results at a misclassification rate of 60%. *MNIST* is an image dataset of handwritten digits. *HAR* (Human Activity Recognition using Smartphones) is a dataset of measurements obtained from a smartphone's accelerometer and gyroscope while the participants holding it performed one of six activities. *FC-100-100-10* refers to a neural network with an input layer, two hidden layers, each containing 100 neurons and an output layer containing 10 neurons. *Robustness improvement* refers to the relative attack strength required to cause 60% misclassification rate vs. an unprotected learner [2]

Chapter 3

Setup

3.1 Tools

All experiments were written in **Python** and the models trained using **Tensorflow**. The only exceptions are the Support Vector Machines (SVMs) [7] used in the next chapter, which were created using the **scikit-learn** library.

3.2 Datasets

We used 3 different binary-label datasets to simplify the security analysis, avoiding the need to generate attacks for multiple classes:

- **MNIST**: MNIST is an image dataset containing handwritten digits labeled from 0 to 9. Each data sample is a gray scale 28x28 image with 784 features, each feature representing a pixel. We focused on the subset consisting of handwritten 7s and 9s, as they have visually similar components. We label the former as class 1 and the latter class 0. For simplicity, we will be referring to this subset as MNIST throughout this document.
- **Ransomware**: The Ransomware dataset contains a subset of 232 features from the dataset used in [27]. The features are normalized and were selected to achieve a reasonably high accuracy. Class 1 is the positive class, which in the context of malware detection means malicious software, whereas class 0 is assigned to benign samples.
- **PDF Malware**: We use a subset of the PDF Malware dataset used in [4]. There are 114 features which we had to normalize to achieve high accuracy. The interpretation of the class labels are the same as for Ransomware.

Chapter 4

Motivation

Prior to training a classifier, there are many hyperparameters to be selected and design choices to be made in order to achieve optimal performance. This work aims to justify feature selection as part of the design consideration to learn a model with a more accurate decision boundary and increased security against attacks.

The intuition behind using feature selection to improve a classifier's security is that by eliminating unimportant features, the attacker can no longer manipulate them to change the decision output. Furthermore, as the feature space of a model grows, so does its complexity, which makes it more prone to overfitting. Hence, feature selection mitigates overfitting and helps the model learn a 'truer' decision boundary. Finally, by reducing the number of features, the perturbation made by an adversary accounts for a larger proportion of the total possible value change. This could potentially make it easier for statistical anomaly detection techniques to detect an attack.

4.1 SVM Experiment

The effectiveness of feature selection in learning a good decision boundary can be visualized. To this end, we demonstrate some examples with SVMs. An SVM is a simple classifier whose decision boundary is the hyperplane that maximizes the margin between the data points of two different classes. We can easily visualize the hyperplane for 2D data points. To this end, we trained the SVM classifiers on a synthetic 2-dimensional dataset. Each data point was generated randomly in each run; the first feature of class 0 instances was drawn from $\mathcal{N}(2, 0.25)$, whereas the first feature of class 1 instances was drawn from $\mathcal{N}(4, 0.25)$. The second feature of all data points was drawn from $\mathcal{N}(3, 0.25)$. Hence, class 1 and class 2 instances should only be distinguished by the first feature.

Figure 4.1 shows the training data points from one sample run. Additionally, the figure contains the true decision boundary as well as the boundary learned using both features and the one learned using only the first, relevant feature. Clearly, the area between the boundary learned using both features and the true decision boundary is significantly larger than the area between the boundary learned using the first feature and the true decision boundary.

We repeated the experiment 20 times for each training method. Figure 4.2 shows that decision boundaries learned without feature selection vary drastically across different runs, as they fit to the randomness of the second feature. In contrast, the hyperplanes learned using the first feature are much closer to one another and to the true decision boundary, as show in figure 4.3.

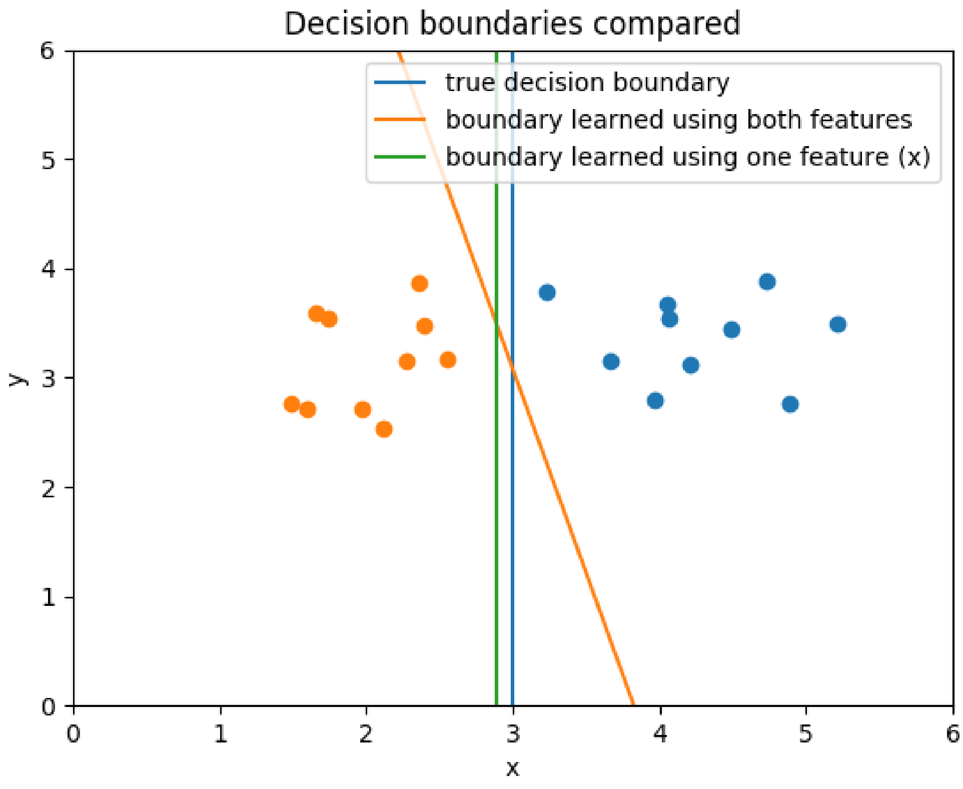


Figure 4.1

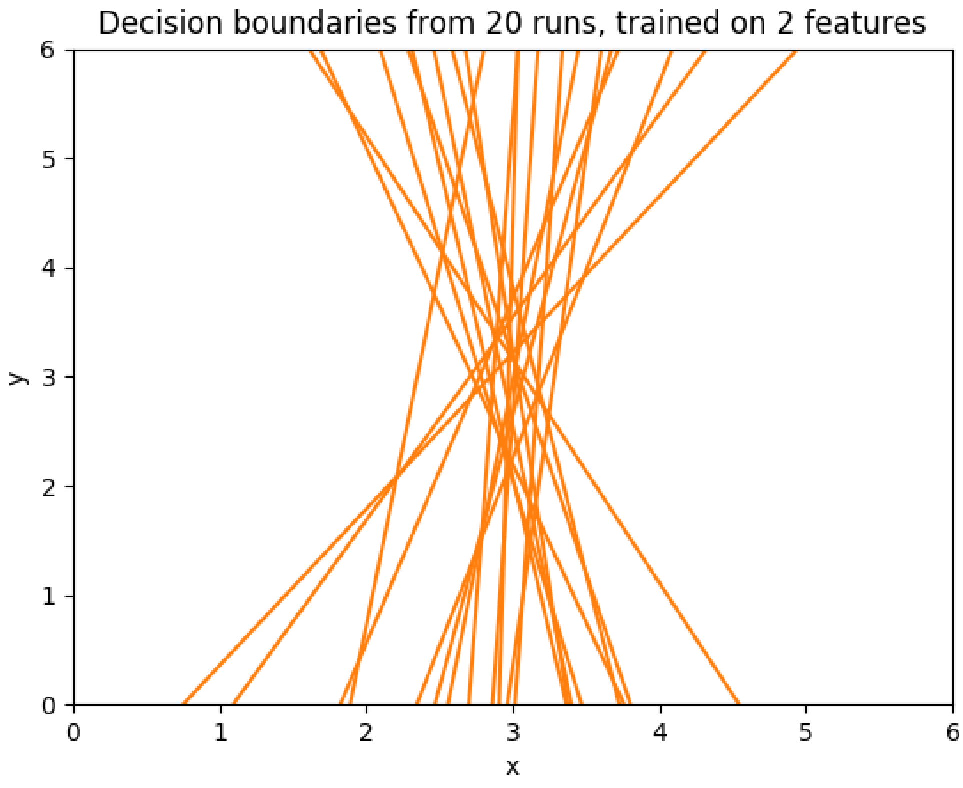


Figure 4.2

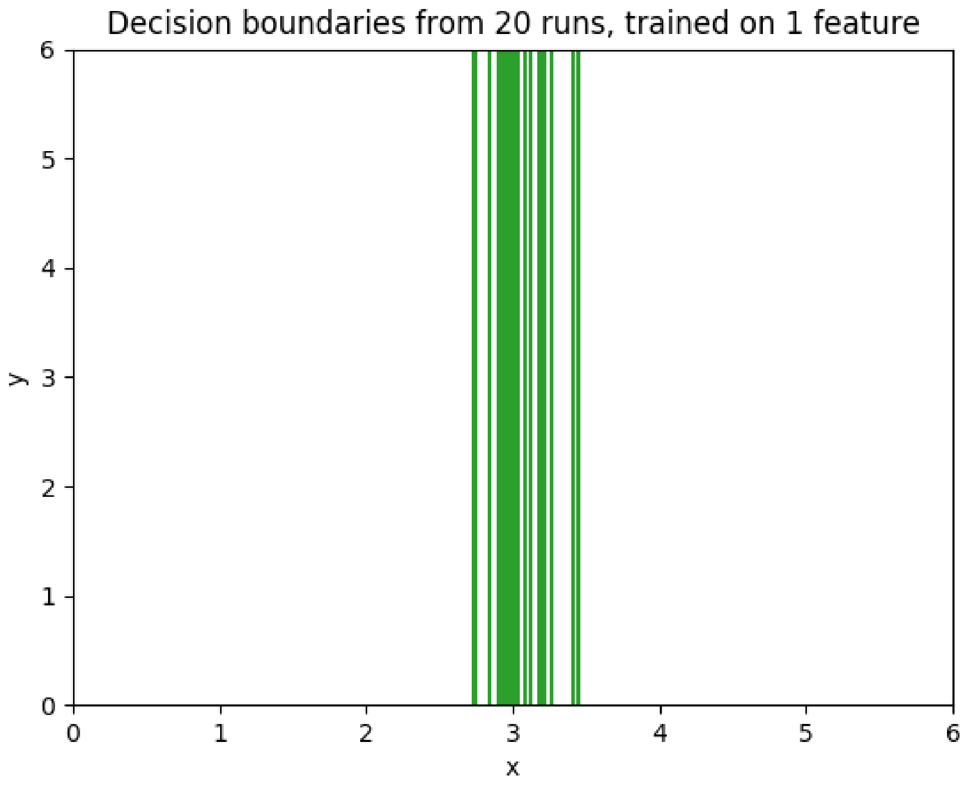

Decision boundaries from 20 runs, trained on 1 feature
0
1
2
3
4
5
6
0
1
2
3
4
5
6
x
y

Figure 4.3

Chapter 5

Effect of Lasso on Security

This chapter details the experiments we performed using Lasso for feature selection and its effect on the security of logistic regression classifiers.

When training machine learning algorithms, ridge regression is often preferred to Lasso for regularization, as it generally gives better results. However, it does not lead to sparse models which we presume to be more secure. By showing that Lasso increases a models security, we can argue for its use instead of or in combination with ridge regression, if security is a concern .

5.1 Experiment

For the experiments we used the three datasets described in chapter 3. For each dataset, we trained a number of logistic regression classifiers, each with a different number of non-zero weights (number of features not ignored by the classifier) by using Lasso constants of different magnitudes.

As class 1 represents the malicious class and the goal of an evasion attack is usually to prevent a malicious sample from being detected, we will only be crafting attacks from class 1 samples in the rest of the work. In the context of our MNIST subset, it means that the attacker is always trying to disguise 7s as 9s.

Simon-Gabriel et al.[29] described a way to scale attack strength according to the attack norm and input dimension. To preserve the average signal-to-noise ratio $\|x\|_2/\|\delta\|_2$, they suggested to compute the attack strength used with L_p-attacks as

$$d_p = c\ f^{1/p}$$

where c is a dimension-independent constant and f is the input dimension. This scaling is useful for normalizing attack strengths across different models for a fixed c value. However, by specifying a computing a distance value d , we would only be able measure the evasion rate for attacks with this distance. We observed that the trends across different models tended to vary with the choice of c.

Hence, we decided to instead compute the average minimum distance, as it is an aggregate value that better captures the differences between models. This is similar to the measures used in other related work [5]. To normalize the attack distance with respect to the number of features, we computed the ratio between the attack distance and the maximum allowed distance. While related research only considered the absolute attack distance when comparing different models, we argue that it is fallacious to do so, as the same perturbation applied to the input of a smaller model means a bigger proportional change.

Specifically, we crafted optimal minimal attacks for each of the three norms (l1, l2, infinity). Each attack is minimal in the sense that it barely causes the classifier classify it as class 0 (output function returns value just under 0.5), as this represents the minimal effort required. Given a minimum distance d_{min}, we compute the normalized value

$$\frac{d_{min}}{f^{1/p}}$$

which denotes the perturbation to maximum possible perturbation ratio, provided the values of all features range from 0 to 1.

To help select the "best" models, we visualize the trade-off between accuracy and security. We propose to use the average of the three normalized attack distances as a comprehensive security score:

$$s = \frac{1}{3}(\frac{d_{L_1}}{f} + \frac{d_{L_2}}{\sqrt{f}} + d_{L_\infty})$$

We then plot the models' security score against accuracy for easy comparison. Finally, to enforce our findings, we perform statistical analysis on adversarial samples by comparing their distribution with clean samples using the statistical distance metric maximum mean discrepancy (MMD).

5.1.1 Training

We split the data in 10 ways (folds) and performed each experiment on each of the folds. This helps validate the stability of our results.

All models were trained with standard gradient descent using the sigmoid cross entropy loss function

$$L(z) = y \cdot -log(s(z)) + (1-y) \cdot log(s(1-z))$$

where z is the network, y is the target label and s is sigmoid function.

Different feature spaces were achieved by selecting different multipliers for the l1 penalty term. It should be noted that Tensorflow's implementation of gradient descent optimizer doesn't yield truly sparse weights when using L1 regularization, as some of the weights end up with very small magnitudes instead of being zeroed. To get sparse solutions, we set a threshold under which weights are zeroed. Furthermore, it was difficult to establish the relationship between the l1 magnitude and the resulting number of non-zero weights, the latter of which tended to differ across different training runs. Hence, if a classifier's number of non-zero weights was close enough to our target (within 5 features), we removed the k weights with the least absolute values, whilst ensuring that prediction accuracy is only minimally affected by the removal.

5.1.2 Attack algorithm

Papernot et al.[25] introduced an optimal, general attack algorithm against DNNs that, given a source sample, the target label, the function of the targeted network and the maximum distortion, an adversarial sample that causes the targeted network to output the desired label, by distorting the source sample while keeping the distortion level below the specified value. We adapted the algorithm to craft attacks constrained by the L_1, L_2 and L_∞ distances against logistic regression classifiers (Algorithm 4). Formally, we define a minimal attack to be

$$x^* = \arg\min_{x'} \ d(x', x) \text{ s.t. } f(x') = y_t$$

where d is a distance function, x is the image to be perturbed, f is the function learned by the targeted classifier and y_t the target output.

To approximate x^*, we performed binary search on the parameter d in Algorithm 4 to find an x' that barely evades the classifier.

Algorithm 4 Attack algorithm

Input: $\boldsymbol{X}$: source sample; $\boldsymbol{Y^*}$: target output; g: function of the network; d_{max}: maximum distance; p: attack norm; α: step size of gradient descent

Output: $\boldsymbol{X^*}$: adversarial sample

```
1: $\boldsymbol{X^*} \leftarrow \boldsymbol{X}$
2: while $f(\boldsymbol{X^*}) \neq \boldsymbol{Y^*}$ do
3:     $\delta \leftarrow \alpha \cdot \nabla g(\boldsymbol{X^*})$
4:     $\delta' \leftarrow \boldsymbol{X^*} - \boldsymbol{X} + \delta$
5:     $\delta^* \leftarrow project_{L_p}(\delta', d_{max})$
6:     $\boldsymbol{X^*} \leftarrow \boldsymbol{X} + \delta^*$
```

To ensure that the distortion respects the specified distance limit d_{max}, we project the perturbation vector δ^* onto the L_p space in each iteration. The projection algorithms are listed below. L_2 and L_∞ projections are very straightforward, whereas the projection onto the L_1 space is more involved. For this, we use the $\mathcal{O}(n \log n)$ algorithm introduced by Duchi et al.[9]. While they also provided a linear time algorithm in the same paper, we found the $\mathcal{O}(n \log n)$ version to perform 3x times as fast in our Python implementation.

Algorithm 5 $project_{L_1}$ [9]

Input: $\boldsymbol{v} \in \mathbb{R}^n$: vector to be projected; $d \in \mathbb{R} > 0$: maximum L^1-norm of the projected vector

Output: $\boldsymbol{w} \in \mathbb{R}^n$: projected vector which solves $\min_{\boldsymbol{w'}} \|\boldsymbol{w'} - \boldsymbol{v}\|_2^2$ s.t. $\|\boldsymbol{w'}\|_1 \leq d$

```
1: Define $\boldsymbol{u}$ where $u_i = |v_i|$
2: $\boldsymbol{\mu} \leftarrow \boldsymbol{u}$ sorted in descending order
3: $\rho \leftarrow \max\left\{j \in \{1, ..., n\} : \mu_j - \frac{1}{j}\left(\sum_{r=1}^{j} \mu_r - d\right) > 0\right\}$
4: $\theta \leftarrow \frac{1}{\rho}\left(\sum_{i=1}^{\rho} \mu_i - d\right)$
5: Define $\boldsymbol{z}$ where $z_i = \max\{u_i - \theta, 0\}$
6: $\boldsymbol{w} \leftarrow sign(\boldsymbol{v}) \star \boldsymbol{z}$
```

Algorithm 6 $project_{L_2}$

Input: $\boldsymbol{v} \in \mathbb{R}^n$: vector to be projected; $d \in \mathbb{R} > 0$: maximum L^2-norm of the projected vector

Output: $\boldsymbol{w} \in \mathbb{R}^n$: projected vector which solves $\min_{\boldsymbol{w'}} \|\boldsymbol{w'} - \boldsymbol{v}\|_2^2$ s.t. $\|\boldsymbol{w'}\|_2 \leq d$

```
1: $n \leftarrow \|\boldsymbol{v}\|_2$
2: $\boldsymbol{w} \leftarrow \frac{\boldsymbol{v} \cdot d}{n}$
```

Algorithm 7 $project_{L_\infty}$

Input: $\boldsymbol{v} \in \mathbb{R}^n$: vector to be projected; $d \in \mathbb{R} > 0$: maximum L^∞-norm of the projected vector

Output: $\boldsymbol{w} \in \mathbb{R}^n$: projected vector which solves $\min_{\boldsymbol{w'}} \|\boldsymbol{w'} - \boldsymbol{v}\|_2^2$ s.t. $\|\boldsymbol{w'}\|_\infty \leq d$

```
1: Define $\boldsymbol{w}$ where $w_i = \min\{\max\{w_i, -d\}, d\}$
```

5.2 Results

5.2.1 Prediction Accuracy

Figures 5.1, 5.2 and 5.3 show prediction accuracy values on the MNIST, Ransomware and PDF datasets respectively for varying input dimensions. Each figure shows the mean, minimum, maximum accuracy values across 10 folds.

5.2.1.1 MNIST

For MNIST, the accuracy remains stable when f, the number of features is between 300 and 784. It slowly declines when f drops from 300 to 10. Even with only 10 features, an accuracy of more than 91% is achieved. The results are not surprisingly, as MNIST is known for having a large number of unused features corresponding to the pixels that are often black, especially around the border of the image.

5.2.1.2 Ransomware

The prediction accuracy remains remarkably stable for the Ransomware dataset as f declines. This suggests that it also contains a large portion of useless features.

5.2.1.3 PDF Malware

PDF Malware classifiers retain extremely high accuracy with 10 features or more. Even with only a single feature, an accuracy of around 93% is achieved. This shows that most features are completely redundant.

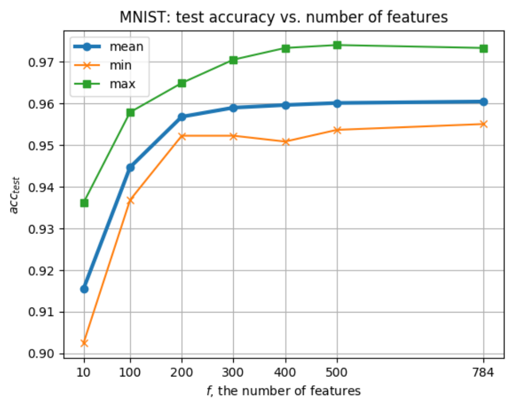


Figure 5.1: Prediction accuracy on MNIST test data.

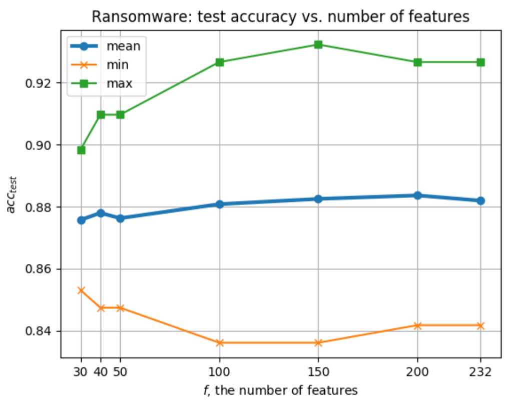


Figure 5.2: Prediction accuracy on Ransomware test data.

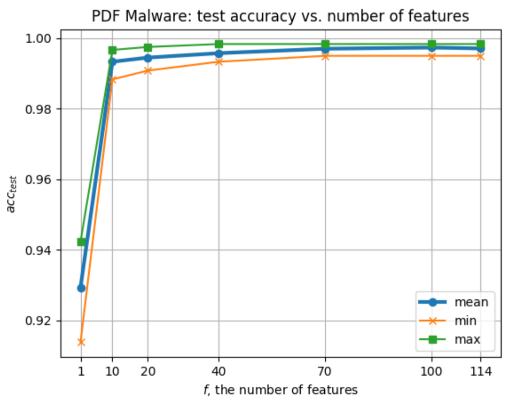


Figure 5.3: Prediction accuracy on PDF Malware test data.

5.2.2 Attack Resistance

5.2.2.1 MNIST

Figures 5.4 and 5.6 show the raw attack distance values for L_1 and L_2 attacks. The fact that d_{avg} drops with the number of features would normally lead one to believe that feature selection with Lasso makes classifiers less secure. However, going by normalized metrics (Figures 5.5, 5.7 and 5.8), Lasso actually multiplies the required distortion ratio going from the largest model to the smallest.

For L_1 attacks, the proportional effort sees more than a ten-fold increase; for L_2 attacks, the effort is tripled, and for L_∞ attacks the effort rises by about 40%.

5.2.2.2 Ransomware

The same trend can be seen for the Ransomware dataset. While d_{avg} decreases as f gets smaller (Figures 5.9 and 5.11), the distortion ratio of L_1 attacks (Figure 5.10) is three times higher for the smallest model compared to the biggest model. A doubling in the relative distortion is seen for L_2 5.7 attacks. The measure sees a 67% increase when it comes to L_∞ attacks.

5.2.2.3 PDF Malware

The average unnormalized attack strength against PDF Malware models are shown in Figures 5.14 and 5.16 with similar results as the other datasets. The relative distortion rises from less than 10% to more than 40% for all three attack norms as f gets smaller (Figures 5.15, 5.16 and 5.18).

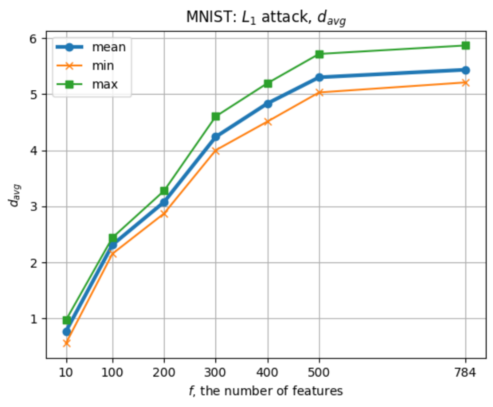


Figure 5.4

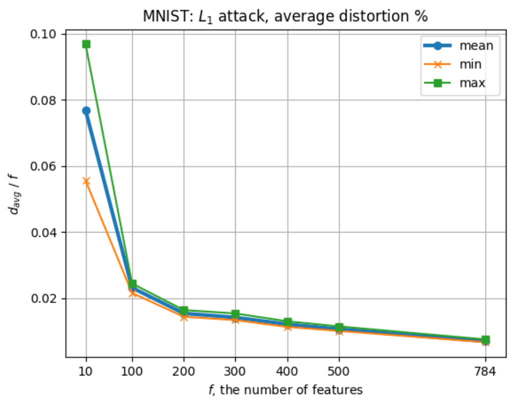


Figure 5.5

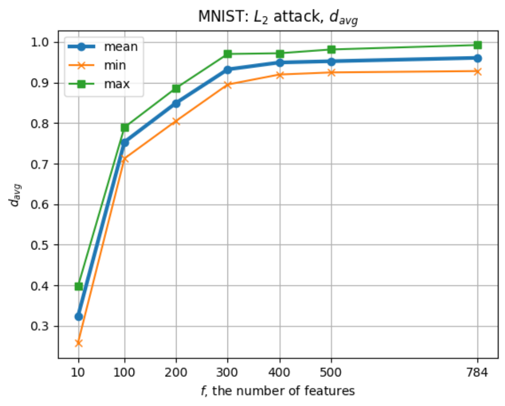


Figure 5.6

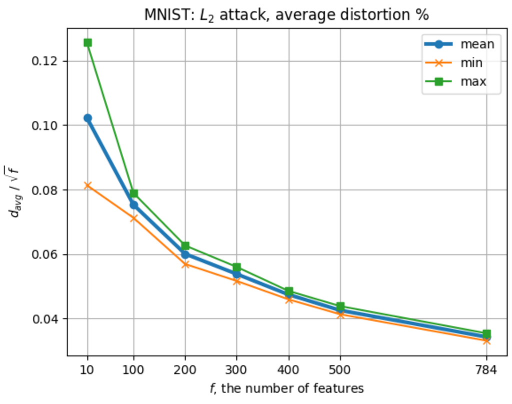


Figure 5.7

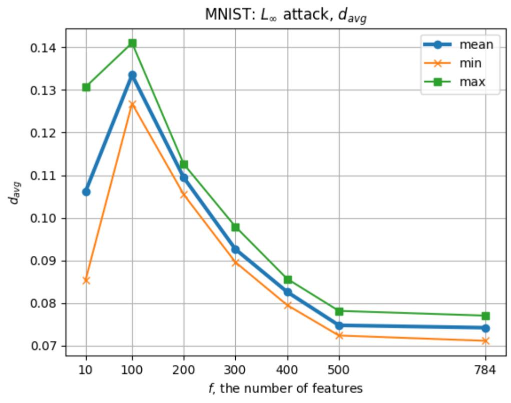


Figure 5.8

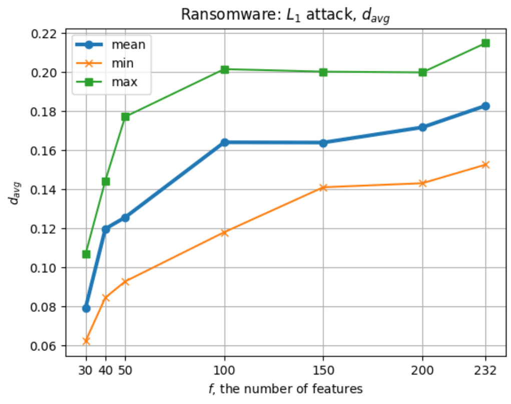


Figure 5.9

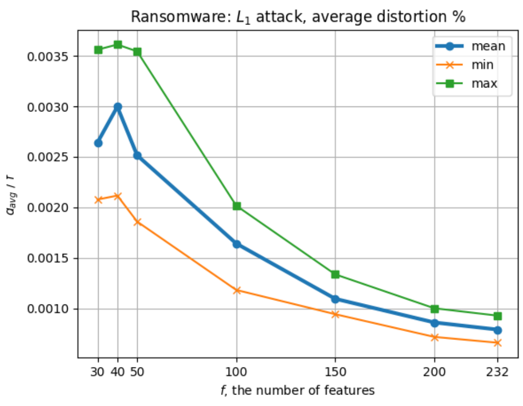


Figure 5.10

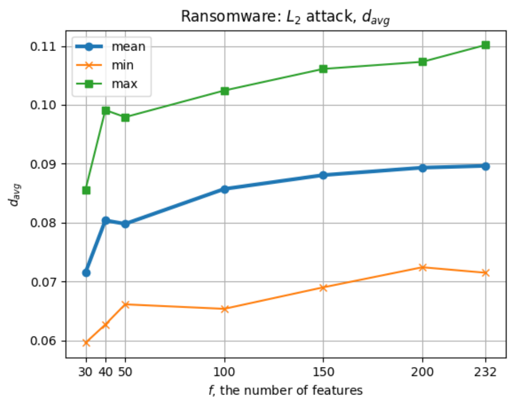


Figure 5.11

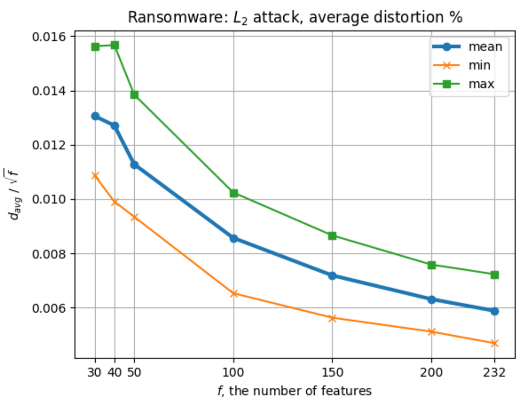


Figure 5.12

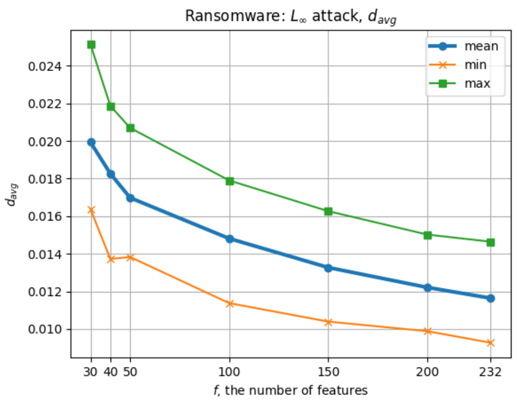


Figure 5.13

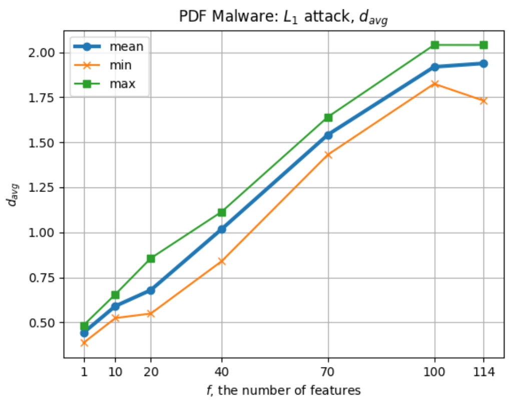


Figure 5.14

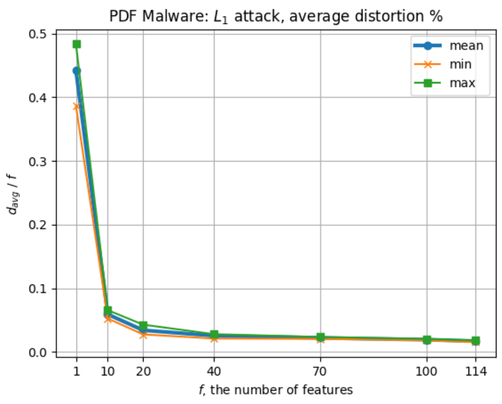


Figure 5.15

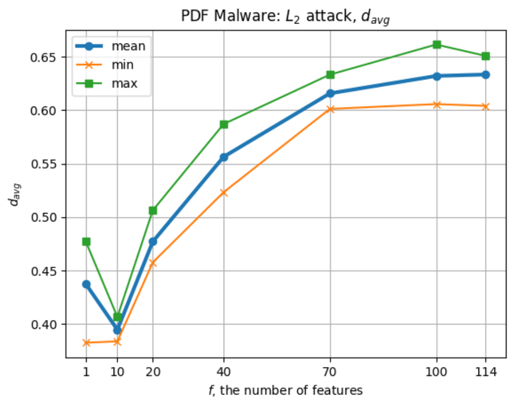


Figure 5.16

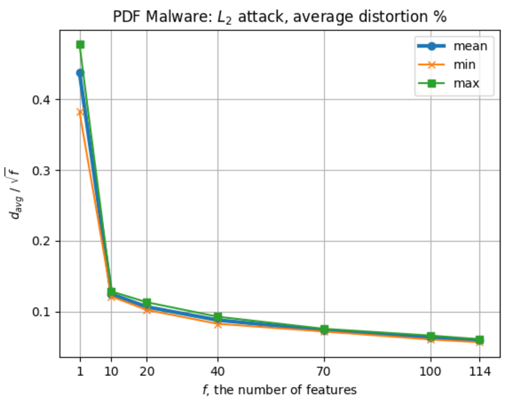


Figure 5.17

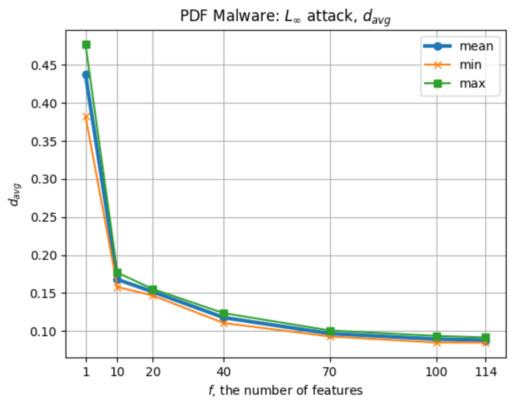


Figure 5.18

5.2.3 Accuracy vs. Security Trade-off

We compute the security score s for all models as described earlier.

5.2.3.1 MNIST

Figure 5.19 plots s against the prediction accuracies of MNIST models. The classifier with 10 features, which has the highest security score, is far less accurate compare to all other models. We can observe that the model with 200 features offers good trade-off between accuracy and security, as it increases the latter by nearly 50% whilst sacrificing less than 0.5% accuracy in comparison with the full model.

5.2.3.2 Ransomware

Regarding Ransomware models (Figure 5.20), the plot clearly shows that all models have similar accuracy levels, but differ in s. The smallest classifier with 30 features is almost twice as secure as the full model while only being around 0.1% less accurate.

5.2.3.3 PDF Malware

As with MNIST, most models are clustered together, except for one outlier, the smallest model. The model with 1 feature is far more secure compared to all others, but is also significantly less accurate. The classifier with 10 features represents a good compromise, as it doubles the security with less than 0.1% decrease in accuracy compared with the full model (Figure 5.21).

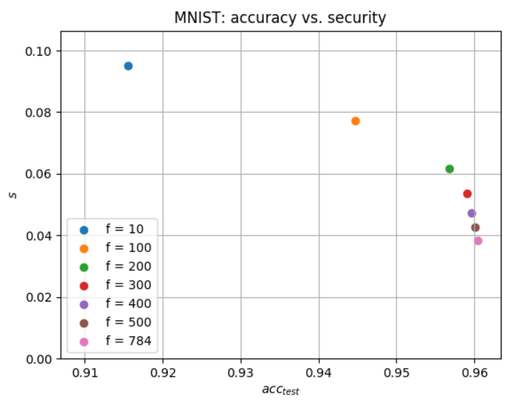


Figure 5.19

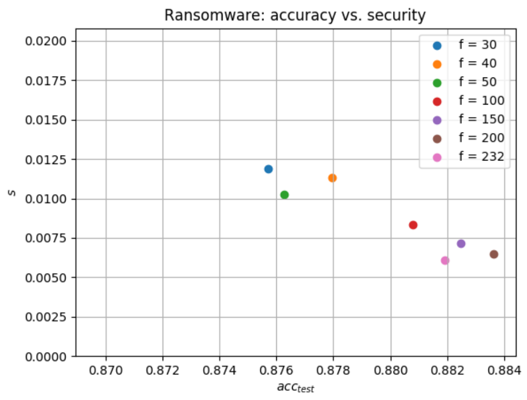


Figure 5.20

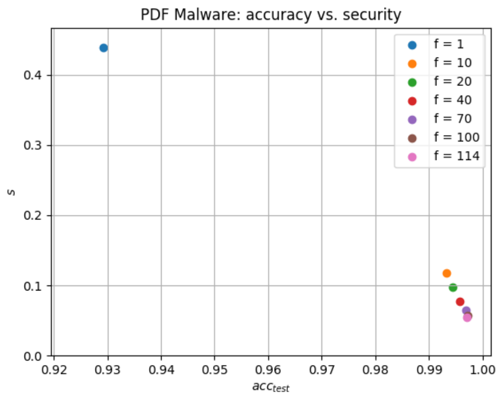


Figure 5.21

5.2.4 Statistical Analysis of Adversarial Samples

Grosse et al.[14] discovered that adversarial samples show measurable statistical differences from clean samples. They found two statistical metrics that revealed discrepancies between adversarial and clean samples: *Maximum Mean Discrepancy (MMD)* [11] and *Energy Distance*. We used the former, as it allowed us to choose a kernel. Given $X = \{x_1, ..., x_n\} \sim p$ and $Y = \{y_1, ..., y_n\} \sim q$, the MMD is defined as

$$D(X, Y, \mathcal{F}) = \sup_{f \in \mathcal{F}} \mathbf{E}_p [f(x)] - \mathbf{E}_q [f(y)]$$

where $\mathcal{F}$ is a class of functions and $\mathbf{E}$ is the expectation. MMD measures the statistical distance between two distributions. $D(X, Y, \mathcal{F}) = 0$ iff $p = q$.

Its unbiased empirical estimate can be computed as

$$\widehat{D}(X, Y) = \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} K(x_i, x_j) - \frac{2}{nm} \sum_{i=1}^{n} \sum_{j=1}^{m} K(x_i, y_j) + \frac{1}{m^2} \sum_{i=1}^{m} \sum_{j=1}^{m} K(y_i, y_j)$$

where K is a kernel function. While Grosse et al.[14] used a Gaussian kernel, the results of which varies with the choice of σ, we decided to use the normalized linear kernel which does not make any assumptions about the underlying distributions and mitigates discrepancies caused by different input dimensions:

$$K(x, y) = \frac{x^\intercal y}{\|x\|_1 \|y\|_1}$$

For each dataset and each attack norm, we computed the MMD between class 1 training samples X_{train_1} and class 1 test samples X_{test_1} as well as the MMD between class 0 training samples X_{train_0} and class 0 test samples X_{test_0} to serve as baselines. We then computed the MMD between the adversarial samples X^* and training samples of both classes separately. The results shown are averages across all folds.

Figures 5.22, 5.23 and 5.24 show the results for the MNIST dataset. Note that the graphs use logarithmic scales for the y-axis as $\widehat{D}(X_{train_0}, X^*)$ and $\widehat{D}(X_{train_1}, X^*)$ grow exponentially as f decreases. Thus, the differences between models with larger f would be obscured if the graphs had linear scales. The plots show that, for all models, the MMDs between training and adversarial samples are multiple times higher than MMDs between training and test samples.. However, the distances between them grow exponentially as f decreases.

Similar trends can be seen in the results for the Ransomware dataset (Figures 5.25, 5.29 and 5.30) and the PDF Malware dataset (Figures 5.28, 5.29 and 5.30).

In conclusion, minimal attacks targeting smaller models exhibit stronger statistical discrepancies with clean samples compared to those targeting larger models. Hence, feature selection with Lasso could be utilized to facilitate statistical attack detection techniques.

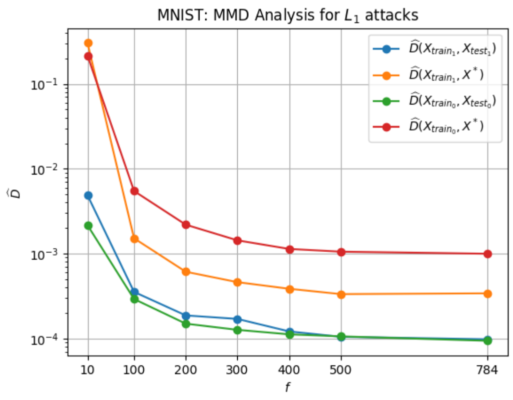


Figure 5.22

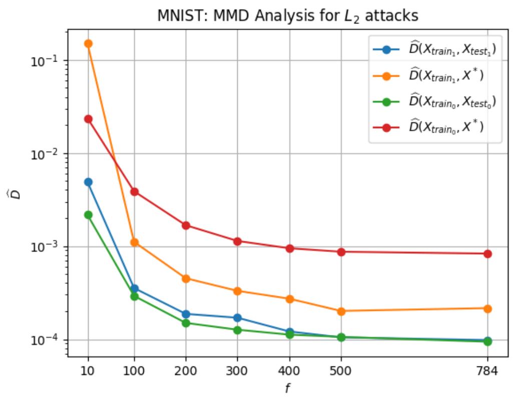


Figure 5.23

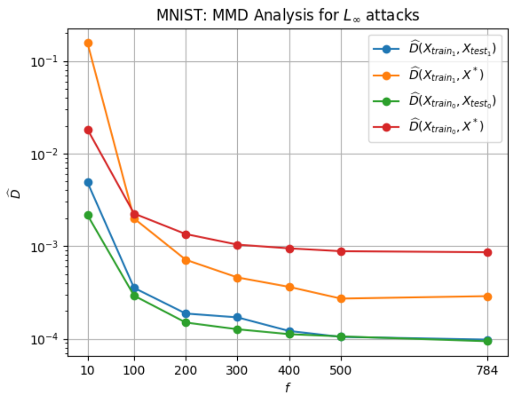


Figure 5.24

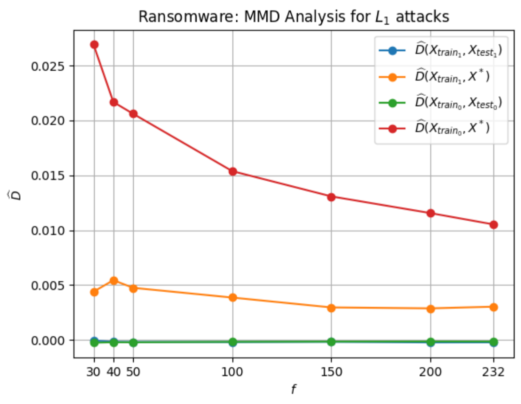


Figure 5.25

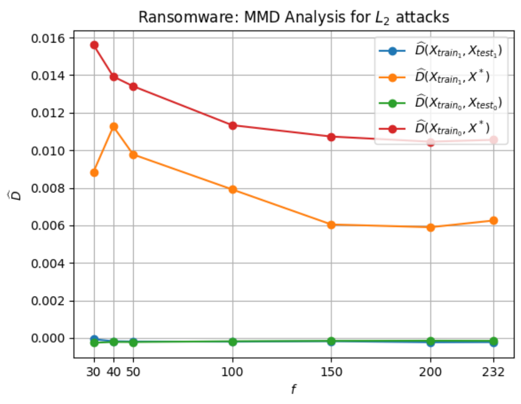


Figure 5.26

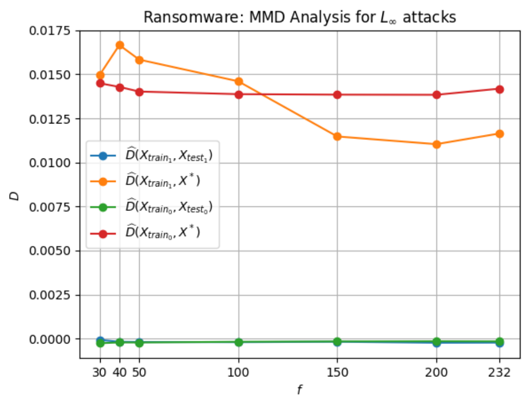


Figure 5.27

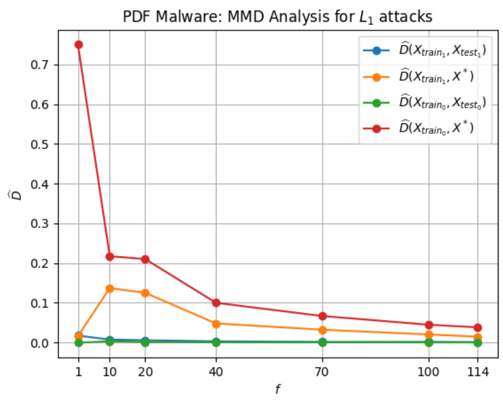


Figure 5.28

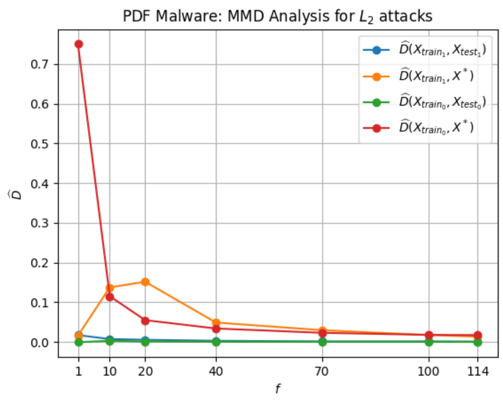


Figure 5.29

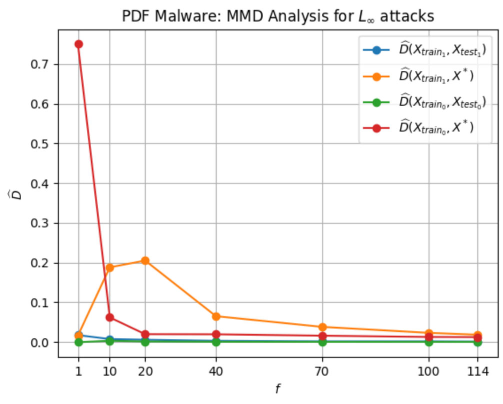


Figure 5.30

5.3 Conclusion

We have shown that by performing dimensionality reduction with Lasso, the required effort proportional to the maximum possible effort increases as the input dimension decreases. Our findings are supported by the fact that adversarial samples differ more strongly from clean samples in reduced space, increasing their detectability by detection methods.

Our results contradict the claims made by [3] and [34] that feature selection has a negative impact on classifier security. If they had normalized their metrics, they would likely have arrived at the same conclusion as we.

Having performed the experiments on three different datasets, we conclude that the results are likely transferable to other classification tasks, making Lasso a secure alternative to Ridge regression when tuning a model.

Chapter 6

One-sided Feature Selection based on Mutual Information and Weight Polarity

Filter-based feature selection methods are popular for their efficiency. As with other traditional feature selection methods, no consideration is given to the security implications of the features chosen. In security critical applications, false negatives are far less desirable than false positives. Hence, we explore in this chapter the possibility of sacrificing true negative rates for improved true positives rates and robustness against positive adversarial samples.

6.1 Feature Selection based on MI and Weight Polarity

An intuitive criteria for feature selection is mutual information (MI). It measures how much information one can obtain from one random variable, given knowledge of another random variable. The mutual information of two discrete random variables X and Y is defined as:

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) \log \left(\frac{p(x,y)}{p(x)\,p(y)} \right)$$

where $p(x)$ is the probability of x and $p(x,y)$ is the joint probability of x and y, $I(X;Y) \geq 0$ and $I(X;Y) = I(Y;X)$
By computing the MI between a set of features and the labels, one can measure how useful these features are in predicting the associated labels. The most established feature selection technique based on MI takes a greedy approach by add the next feature that maximizes the MI in each iteration [1].

Our idea is to remove unimportant features associated with the negative class, precluding the possibility of manipulating these features to disguise a positive sample. Mutual information by itself does not reveal whether the activation of a feature is more indicative of the positive or negative class. Therefore, we propose to first pre-train an LR classifier without feature selection. Features associated with weights that are negative will be indicative of the negative class. All such features are potential candidates for removal. We score them by computing the MI metric between each feature and the output labels. This differs from [1] in that the MI is computed independently for each feature rather than subsets of features. Features with the k lowest scores are then removed. Finally we retrain the classifier with the chosen subset of features. The pseudocode for this method is described in Algorithm 8.

Algorithm 8 Feature selection based on Mutual Information and Weight Polarity

Input: X: training samples; Y: training labels; F_{all}: set of all features; W: weights of pre-trained LR classifier; k: the number of features to remove

Output: F: the set of selected features

```
1: Define $W^-$ where $w_i^-$ is a negative element in $w$
2: Define $F^-$ where $f_i^-$ is a feature associated with an element in $W^-$
3: Define $S$ where $s_i = I(X_{f_i^-}; Y)$
4: Define $F_r$ where $f_r$ is among $k$ features with the lowest $s_i$
5: $F \leftarrow F_{all} \setminus F_r$
```

6.2 Experiment

Similar to the experiments performed on Lasso in Chapter 5, we trained models with different input dimensions for each of the three datasets. However, it was not possible to get very small models this time, since only features associated with negative weights were allowed to be removed.

We evaluated the performance of the classifiers by looking at their test accuracy, true negative and true positive rates to investigate the trade-offs.

With regards to security evaluation, we again generated minimal white-box attacks against the models and plot both raw and normalized attack distances. However, we only used L_2 attacks, since we observed similar trends for attacks of all three norms in Chapter 5.

As a baseline, we also performed all above evaluations with a feature selection algorithm that removes k features completely at random.

6.2.1 Discretization of Feature Values

Notice that the above definition of the mutual information is for discrete random variables, although the datasets used have features with continuous values. While MI is also defined for continuous random variables:

$$I(X;Y) = \int_Y \int_X p(x,y) \log\left(\frac{p(x,y)}{p(x)\,p(y)}\right) dx\,dy,$$

it requires approximating the probability density functions as well as solving double integrals. To simplify the problem, we discretized the features. We explored various "reasonable" bin sizes and found no significant differences in the resulting models. Thus, we settled on rounding the feature values to the first decimal place, meaning there were 10 "bins" for the values.

6.3 Results

6.3.1 Prediction Accuracy

6.3.1.1 MNIST

Figures 6.1, 6.2 and 6.3 show, respectively, the accuracy, TNR and TPR of MNIST classifiers whose features were selected with our MI-based method. Surprisingly, there is only a slight bump in TNR as f, the number of features drops to 500. The TNR starts to decline as f drops lower. The TPR remains relatively stable until f hits 300, where very few of the features originally associated with the negative class are remaining.

We discovered that the reason why the TNR did not change favorably with our feature selection method was that some weights flipped their polarity after retraining. Many weights that were originally indicative of the positive class became indicators for the opposite class. This was potentially the result of an attempt of the optimizer to compensate for the lack of features associated with the negative class and thereby mitigated any potential benefits of removing them.

Compared to models resulting from random feature selection (Figures 6.4, 6.2, 6.3), MI-based models generally score lower on all three performance metrics, especially as f gets small.

6.3.1.2 Ransomware

Unlike MNIST, MI-feature selected classifiers (Figures 6.7, 6.8, 6.9) generally perform better than their random counterparts (Figures 6.10, 6.11, 6.12). Unexpectedly, the TNR (rather than the TPR) of the former see a slight bump for moderately reduced models.

6.3.1.3 PDF Malware

MI-based (Figures 6.13, 6.14 and 6.15) and random (Figures 6.16, 6.17 and 6.18) feature selection methods perform similarly. Our FS technique did not increase the classifiers' TNR, as they were already extremely high, and since the dataset mostly consists of redundant features, removing them did not have a big impact.

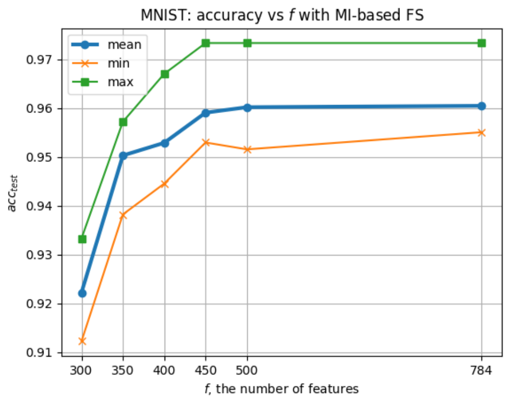


Figure 6.1: Prediction accuracy on MNIST test data. MI-based feature selection.

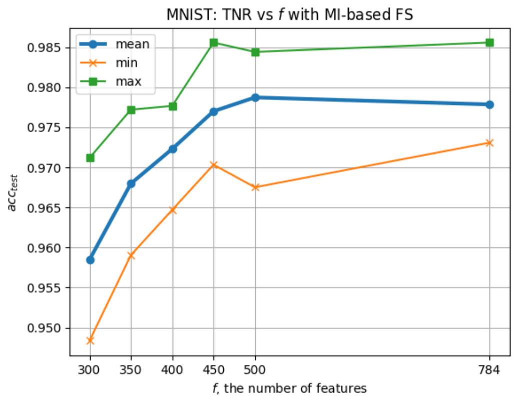


Figure 6.2: True negative rate on MNIST test data. MI-based feature selection.

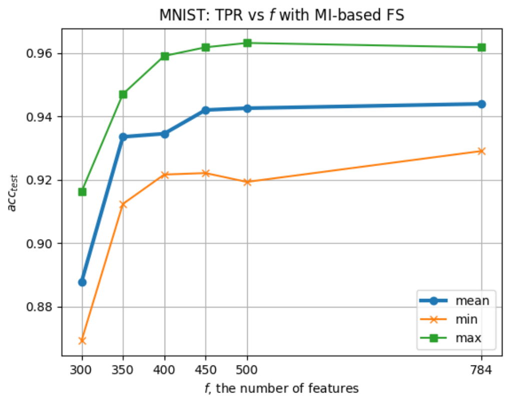


Figure 6.3: True positive rate on MNIST test data. MI-based feature selection.

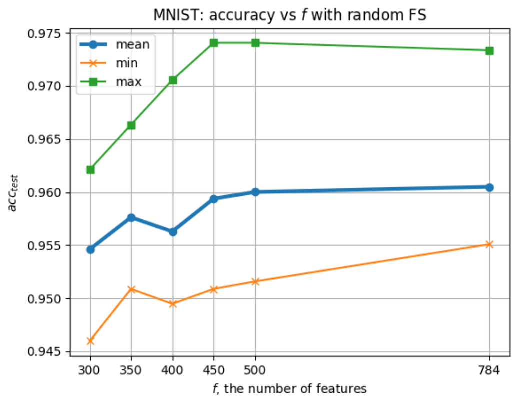


Figure 6.4: Prediction accuracy on MNIST test data. Random feature selection.

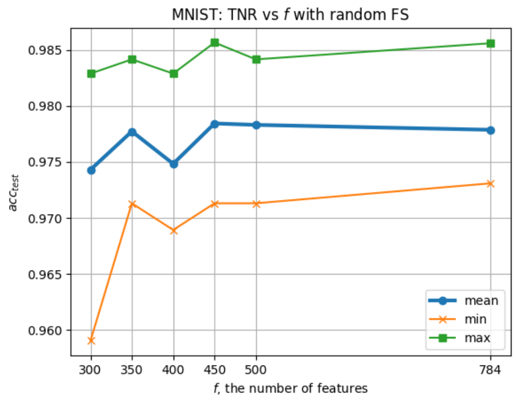


Figure 6.5: True negative rate on MNIST test data. Random feature selection.

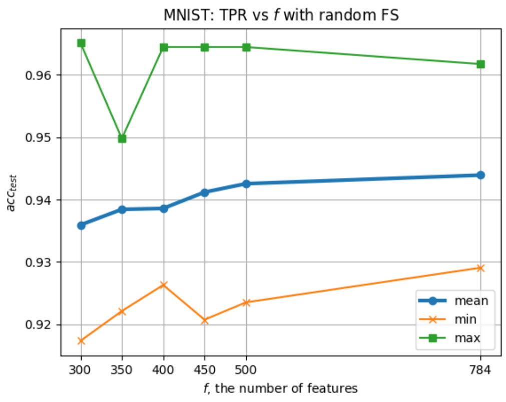


Figure 6.6: True positive rate on MNIST test data. Random feature selection.

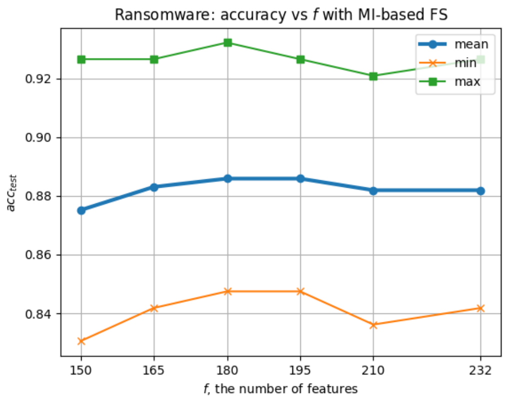


Figure 6.7: Prediction accuracy on Ransomware test data. MI-based feature selection.

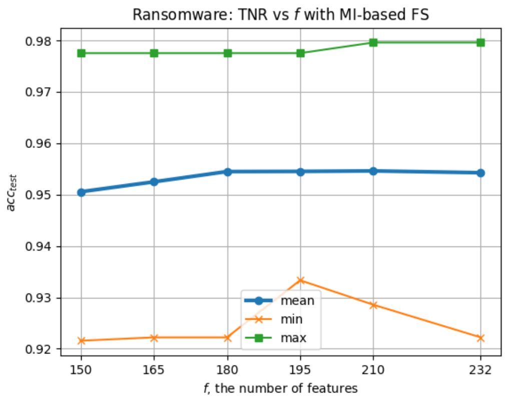


Figure 6.8: True negative rate on Ransomware test data. MI-based feature selection.

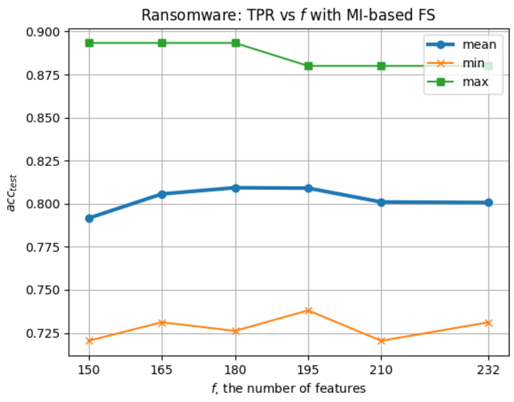


Figure 6.9: True positive rate on Ransomware test data. MI-based feature selection.

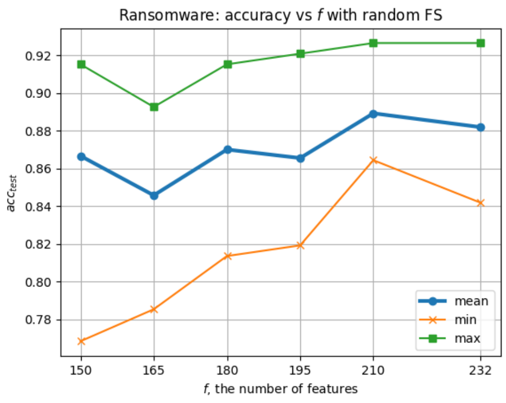


Figure 6.10: Prediction accuracy on Ransomware test data. Random feature selection.

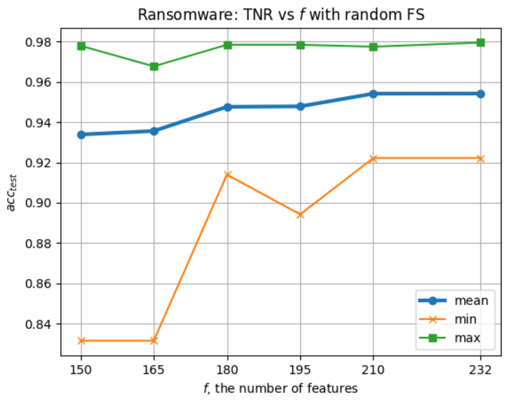


Figure 6.11: True negative rate on Ransomware test data. Random feature selection.

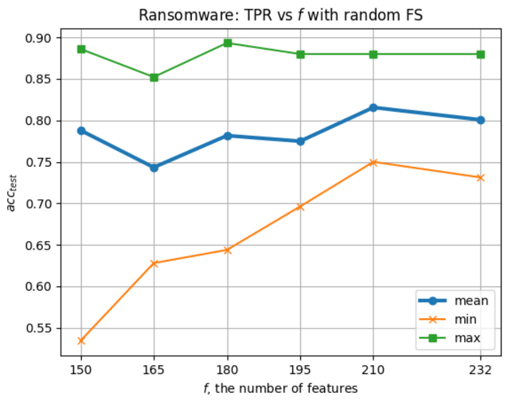


Figure 6.12: True positive rate on Ransomware test data. Random feature selection.

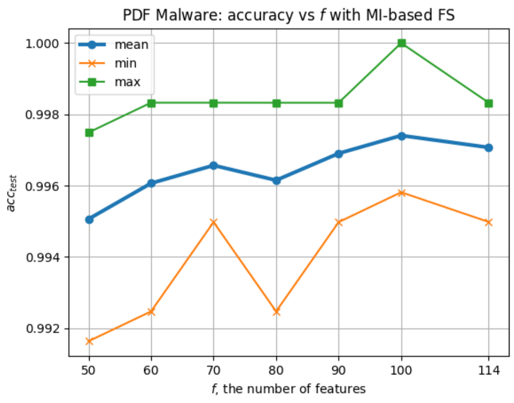


Figure 6.13: Prediction accuracy on PDF Malware test data. MI-based feature selection.

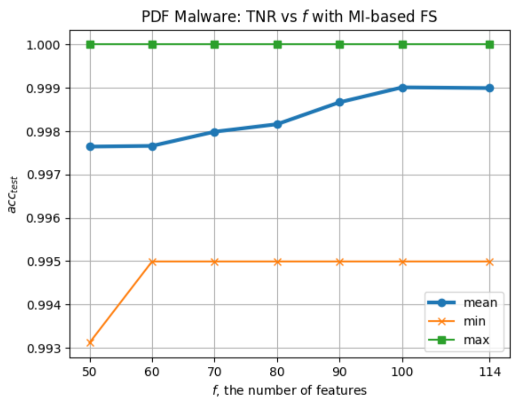


Figure 6.14: True negative rate on PDF Malware test data. MI-based feature selection.

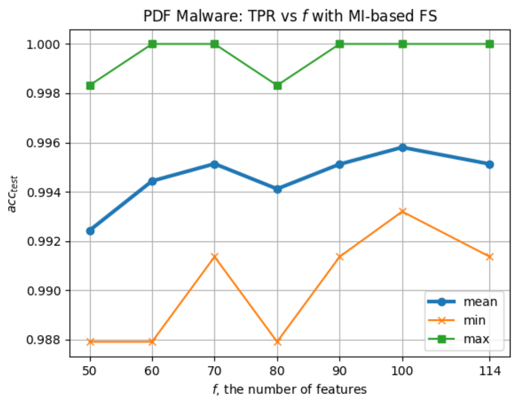


Figure 6.15: True positive rate on PDF Malware test data. MI-based feature selection.

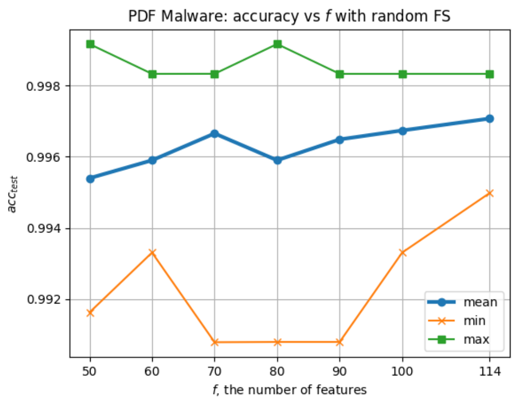


Figure 6.16: Prediction accuracy on PDF Malware test data. Random feature selection.

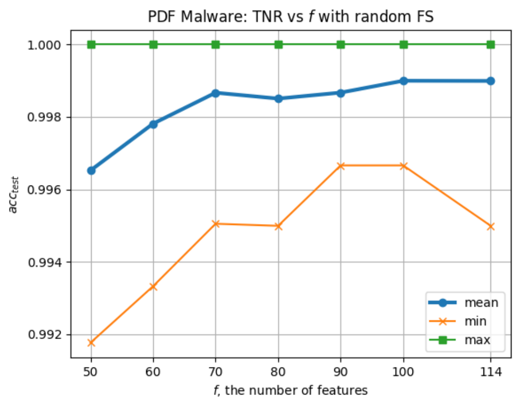


Figure 6.17: True negative rate on PDF Malware test data. Random feature selection.

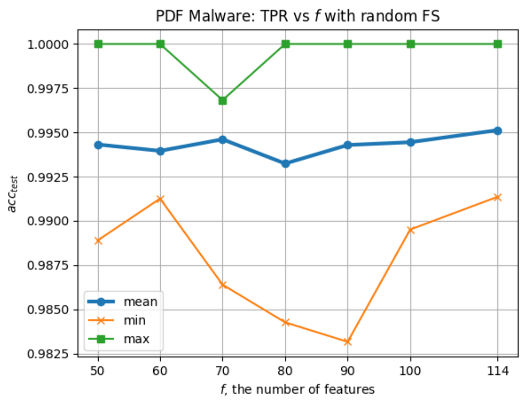


Figure 6.18: True positive rate on PDF Malware test data. Random feature selection.

6.3.2 Attack Resistance

6.3.2.1 MNIST

Figures 6.19 and 6.20 show that both MI-based and random feature selection can increase the classifiers' robustness. In the former case, the security metric peaks at 450 features, whereas in the latter case, the robustness level almost monotonically increases as f decreases. Both feature selection techniques generate models that are less secure compared to Lasso (Figure 5.7).

6.3.2.2 Ransomware

It can be seen that MI-based FS (Figure 6.21) makes Ransomware classifiers more robust compared to those trained resulting from the random techniques (Figure 6.22. In fact, the MI-based technique provides similar robustness improvements compared to Lasso (Figure 5.12).

6.3.2.3 PDF Malware

Figure 6.23 shows that, similar to MNIST, MI-based FS increases PDF Malware classifiers' robustness up to a certain point, which then declines with f. In contrast, the robustness levels are relatively stable for all input sizes with the random FS method (Figure 6.24). Again, both methods are outperformed by Lasso (Figure 5.17).

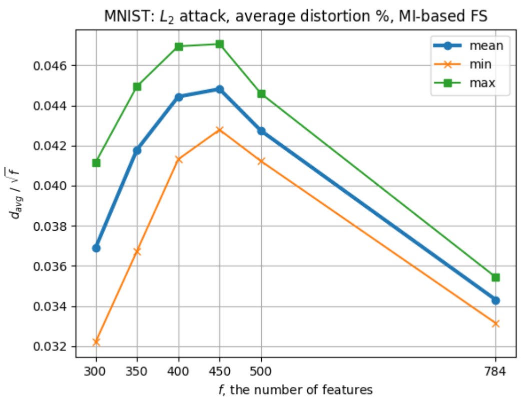


Figure 6.19

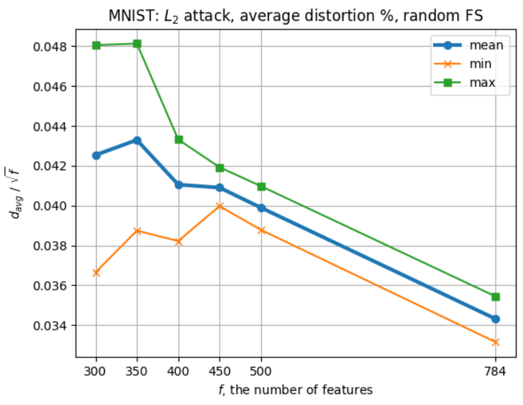


Figure 6.20

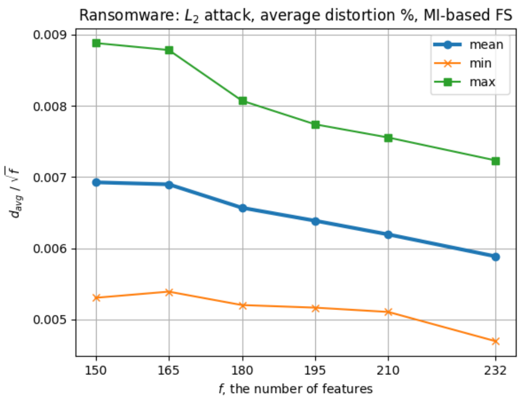


Figure 6.21

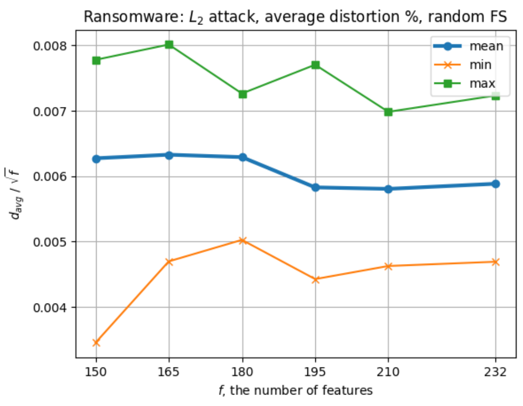


Figure 6.22

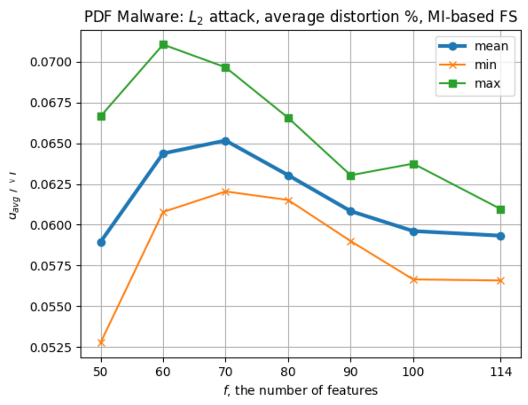


Figure 6.23

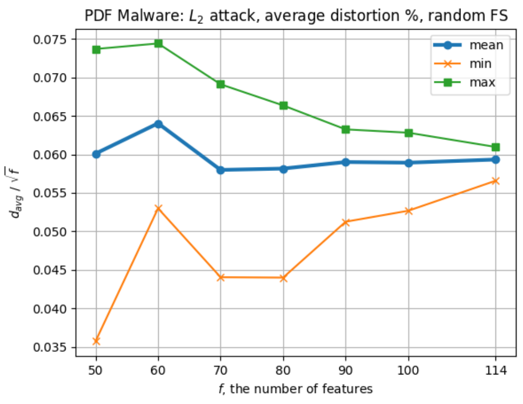


Figure 6.24

6.4 Conclusion

Chapter 7

Effect of Autoencoder on Security

This chapter details experiments investigating whether Autoencoders enhance the robustness against attacks.

While Bhagoji et al. [2] showed that transforming input data via PCA decreases the success rate of adversarial attacks, there exist more powerful feature extraction techniques which can capture non-linear structure in the data. We examined the effect on classifier security and performance by processing the input with traditional Autoencoders and its variants, Denoising and Stack Denoising Autoencoders.

7.1 Experiment

For all experiments described in this chapter, the MNIST dataset was used. We trained Autoencoders, Denoising Autoencoders and Stacked Autoencoders in various configurations. The main variables in the configurations are: the number of layers, the number of hidden nodes and whether the LR classifier is connected to the code layer or to the output layer of the Autoencoder. As in Chapter 5, we measured the robustness of networks by crafting minimal attacks from class 1 and computing the average attack distance over all samples. However, no normalization needed to be applied to the distances as the input dimension is the same for all models tested.

7.1.1 Crafting Adversarial Samples

We found that Algorithm 4 does not work with LR classifiers connected to Autoencoders. Hence, we attempted various strategies based on back-propagation. By combining the Jacobian-Based Saliency Map Attack (JSMA) [25] and Carlini & Wagner's attack [6], we found one that strikes a good balance between optimality and efficiency.

To solve the problem of finding the minimal attack, our algorithm consists of two phases: the *viable attack generation* phase and the *attack distance reduction* phase.

7.1.1.1 Viable attack generation

For viable attack generation (Algorithm 9), we simply solve the problem of finding a successful attack while ensuring that attack distance is bounded by the attack norm. However, the attack need not be minimal:

$$x^* = \arg\min_{x'} \; \left|o(x') - y_t\right| + c \cdot \|x' - x\|_p$$

where $|o(x') - y_t| \in [1, 0]$ is the difference between the output of the network and the target label and c controls the importance of minimizing the attack distance. Solving the above problem is equivalent to solving one iteration of C&W attack's binary search. For a successful attack, c needs to be chosen such that the target label y_t is the closest to the output $o(x^*)$. Empirically, the same c value can be used to find successful attacks for all models we tested. However, a small c value can cause the gradient descent to take more iterations to converge.

Algorithm 9 Attack against AE-enforced LR classifier

Input: $\boldsymbol{X}$: source sample; $\boldsymbol{Y^*}$: target output; p: attack norm; c: scale of the distance penalty term; l: attacker's loss function $l(x, x') = |o(x') - y_t| + c \cdot \|x' - x\|_p$; α: step size of gradient descent; e: minimum L_2-norm of the initial gradient $\nabla l(\boldsymbol{X}, \boldsymbol{X^*})$

Output: $\boldsymbol{X^*}$: adversarial sample

```
1: Initialize $\boldsymbol{X^*}$ where $x_i^* \sim \mathcal{U}(0, 1)$ s.t. $\|\nabla(o(\boldsymbol{X^*}) - \boldsymbol{Y^*})\|_2 \geq e$
2: repeat
3:     $\delta \leftarrow \alpha \cdot \nabla l(\boldsymbol{X}, \boldsymbol{X^*})$
4:     $\boldsymbol{X^*} \leftarrow clip(\boldsymbol{X^*} + \delta)$
5: until convergence
```

We found that we could not use initialize the attack using the source sample x, as the gradient of the prediction loss $o(x) - y_t$ with respect to x is always zero or near zero, rendering gradient descent useless. This phenomenon is known as gradient masking [25] and can result from adversarial training and defensive distillation [24]. It is interesting to observe that gradient masking occurs naturally in LR classifiers reinforced with Autoencoders.

To escape the plateau, Tramèr et al. [31] proposed to initially make a single random step away from the source point. However, the technique was not effective against our networks, as even after 10000 attempts with different random steps, the probability of finding a point where the gradient was non-zero was below 20%. This suggests that the plateau surrounding the source point is quite large. We found that by initializing the initial point completely randomly, it usually took less than 100 attempts to find a viable point.

One problem associated with complex optimization problems is the existence of many local minima. The optimality of the minimum that the gradient descent algorithm converges to depends heavily on the location of the starting point. Hence, to find a good local minimum for each sample, we repeated Algorithm 9 5 times with different random starting points and used the solution with the lowest distance as input to the next phase.

7.1.1.2 Attack distance reduction

In the distance reduction phase, we take the successful attack x^* generated in the first phase and process it in Algorithm 10, where its distance gets slowly decremented. Algorithm 10 works similarly as 4 by projecting the perturbation vector in each iteration of the gradient descent, but instead of specifying a target distance as input, we decrement the projected distance in each iteration until the targeted classifier just barely assigns the target label to the attack. As in Algorithm 4, the gradient descent is performed on the loss function

$$l(x') = \big|o(x') - y_t\big|$$

The distance penalty term is not required here as the algorithm uses projections.

Algorithm 10 Attack distance reduction

Input: $\boldsymbol{X}_a$: successful adversarial sample; $\boldsymbol{Y}^*$: target output;
l: attacker's loss function $l(x') = |o(x') - y_t|$; g: prediction function of the network; p: attack norm; α: step size of gradient descent; Δ_d: decrement in projection distance
Output: $\boldsymbol{X}^*$: minimal adversarial sample

```
1: $\boldsymbol{X}^* \leftarrow \boldsymbol{X}_a$
2: $\boldsymbol{X}' \leftarrow \boldsymbol{X}^*$
3: repeat
4:     $\boldsymbol{X}' \leftarrow \boldsymbol{X}^*$
5:     $d_t \leftarrow \|\boldsymbol{X} - \boldsymbol{X}^*\|_p - \Delta_d$
6:     $\delta \leftarrow \alpha \cdot \nabla g(\boldsymbol{X}^*)$
7:     $\delta \leftarrow \boldsymbol{X}^* - \boldsymbol{X} + \delta$
8:     $\delta \leftarrow project_{L_p}(\delta, d_t)$
9:     $\boldsymbol{X}^* \leftarrow clip(\boldsymbol{X} + \delta)$
10: until $g(\boldsymbol{X}^*) \neq \boldsymbol{Y}^*$
11: $\boldsymbol{X}^* \leftarrow \boldsymbol{X}'$
```

7.2 Evaluation & Results

7.2.1 Traditional Autoencoders

We first evaluate traditional autoencoders connected to LR classifiers in various configurations. The following are the variables in our experiment:

- Number of hidden layers: We compare single-layer AEs with deep AES containing 3 hidden layers (1 extra layer for the encoder and decoder) as deep models are able to model more complex functions and we would like to investigate its effects on performance and security.

- Number of hidden nodes: We vary the number of hidden nodes to control the strength of compression. To establish whether the results stem from the nature of autoencoders or from the compression, we also look at non-uncompleted autoencoders where no compression takes place .

- Representation connected to LR classifier: For each autoencoder trained, we test whether there are differences between using the code layer (compressed representation) as input to the LR and using the output (reconstructed representation) of the decoder. In the latter case, we use the same 'vanilla' LR classifier that also serves as a baseline for performance and security.

7.2.1.1 Training

All single-layer autoencoders were trained using the RMSProp optimizer with a learning rate of 0.02 over mini-batches of size 256 for 100,000 iterations, whereas multi-layer autoencoders were trained for 200,000. We found it crucial to use regularization during the training process, as unregularized networks took more iterations to converge and were vulnerable to attacks that modified unexpected features (e.g. border pixels). Hence, we added an L1 penalty term with magnitude of 0.0000001 to encourage the learning of sparse representations. We chose the sigmoid function for neuron activation, as it can model non-linearity.

To evaluate networks where the code layer is connected to a classifier, we train the classifier on the encoded training set.

7.2.1.2 Results: Classifying code representation

Table 7.1 shows that, contrary to our initial assumption, LR classifiers learned from compressed representations are no more secure and in some cases less secure than a standard LR classifier. For multi-layer encoders, certain models show robustness levels significantly lower than the baseline. We do not believe that this is caused by the particular choice of layer sizes, as the results varied greatly when we retrained these models with the same hyperparameters. However, we only recorded the least robust models. The variance in security can be explained by the added complexity of training extra hidden layers, causing autoencoders to end up with very different weights across different training runs.

In terms of performance, the prediction accuracy decreases with the size of the code layer, similar to using other dimensionality reduction techniques.

7.2.1.3 Results: Classifying reconstructed representation

Table 7.2 shows that preprocessing the input of our independently retrained LR classifier with an autoencoder increases its robustness against L_2 attacks by up to 47%. However, this does not appear to be the result of dimensionality reduction, as both single-layer and multi-layer non-undercomplete autoencoders perform similar to some of their undercomplete counterparts. Again, there are strong variances in the results, with some models slightly less secure than the undefended LR classifier. The increase in robustness could be attributed to a fluke in weight assignments, as the effect is not consistent.

In the case of single-layer autoencoders, accuracy generally drops with the size of the code layer, although the performance hit is less pronounced in comparison with classifiers learned from compressed representations. Using additional hidden layers seems to mitigate the accuracy loss when using smaller autoencoders.

Interestingly, the average reconstruction error for adversarial samples is significantly higher than for clean samples. However, there is much overlap between the error values, which rules out the possibility of using the reconstruction error as an attack detection method.

7.2.2 Denoising Autoencoders

We now evaluate single-layer, deep and stacked denoising autoencoders connected to LR classifiers. We refer to deep denoising autoencoder as a multi-layer autoencoder where all layers are trained at the same time rather than greedy layer-wise. We also compare the DAE-enhanced classifiers with an LR classifier trained on noisy data to see if using DAEs is more effective than simply adding noise to the training data of an LR classifier.

7.2.2.1 Training

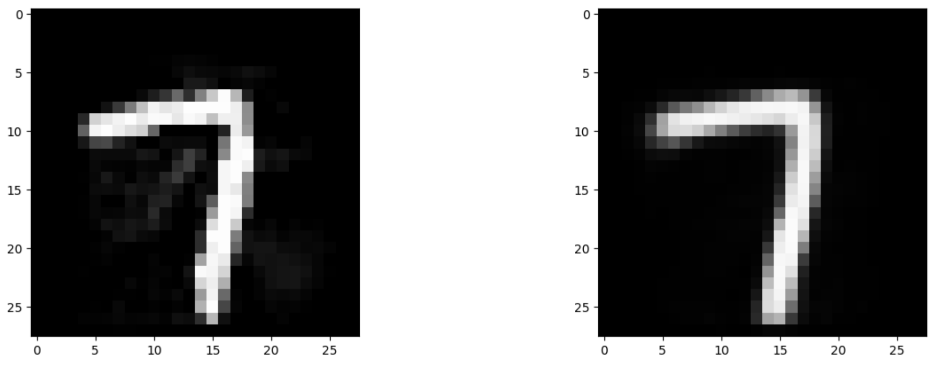
The single-layer and deep denoising autoencoders were trained for 200,000 iterations each with the same settings used in the previous subsection. As for SDAEs, we trained each encoder-decoder pair for 100,000 iterations, totaling 200,000 iterations. We experimented training with both additive uniform and Gaussian noise at various magnitudes and found no empirical difference in the resulting accuracy and security for appropriate parameters.

Table 7.1: Evaluation of single-layer and two-layer encoders connected to LR classifiers. "Enc-784 + LR" means the code layer with 784 nodes of an encoder serves as input to an LR classifier. "Enc-784-392 + LR" means the encoder has 784 nodes in the first hidden layer and 392 nodes in the code layer and its code layer serves as input to an LR classifier.

Model	Accuracy (%)	TNR (%)	TPR (%)	Average L_2 attack distance
LR	96.61	98.61	94.65	0.97
Enc-784 + LR	96.71	98.32	95.14	0.95
Enc-392 + LR	97.10	98.81	95.43	0.95
Enc-196 + LR	95.93	98.02	93.87	0.93
Enc-98 + LR	96.07	98.02	94.16	0.84
Enc-49 + LR	95.24	97.52	93.00	0.89
Enc-24 + LR	93.32	96.04	90.66	0.99
Enc-12 + LR	91.21	96.63	85.89	0.76
Enc-784-784 + LR	96.76	98.32	95.23	1.01
Enc-784-392 + LR	96.56	98.32	94.84	0.94
Enc-392-196 + LR	96.56	98.51	94.65	0.89
Enc-196-98 + LR	95.24	98.02	92.51	0.64
Enc-98-49 + LR	94.55	97.72	91.44	0.64
Enc-49-24 + LR	91.85	96.83	86.96	0.78
Enc-24-12 + LR	92.54	97.52	87.65	0.95

Table 7.2: Evaluation of full single-layer and deep autoencoders connected to LR classifiers. "AE-784 + LR" means the output layer of an autoencoder serves as input to an LR classifier. "AE-784-392 + LR" means the autoencoder has 784 nodes in the first and third hidden layer and 392 nodes in the code layer and the output layer serves as input to an LR classifier.

Model	Accuracy (%)	TNR (%)	TPR (%)	Average L_2 attack distance
LR	96.61	98.61	94.65	0.97
AE-784 + LR	96.22	98.61	93.87	1.32
AE-392 + LR	95.93	98.61	93.29	1.26
AE-196 + LR	95.73	98.41	93.09	1.15
AE-98 + LR	95.39	94.85	95.91	1.37
AE-49 + LR	96.61	97.72	95.53	1.09
AE-24 + LR	94.60	91.87	97.28	1.06
AE-12 + LR	94.75	98.81	90.76	0.93
AE-784-784 + LR	96.02	97.92	94.16	1.43
AE-784-392 + LR	95.97	98.71	93.29	1.13
AE-392-196 + LR	96.07	98.51	93.68	0.93
AE-196-98 + LR	96.22	98.91	93.58	1.08
AE-98-49 + LR	95.93	96.23	95.62	1.31
AE-49-24 + LR	96.07	94.85	97.28	1.26
AE-24-12 + LR	96.32	98.32	94.36	1.37



(a) successful attack against regular LR (b) attack denoised by DAE-784-784

Figure 7.1: Successful denoising of attack

Hence, we will solely be adding uniform noise with the noise vector $\tilde{v} \sim [\mathcal{U}(-1,1)]^f$ to all training samples.

7.2.2.2 Results: Classifying code representation

LR classifiers learned from the code of all three types of autoencoders see an increase in robustness against L_2 attacks (Table 7.3). There appears to be a positive correlation between the code layer size, accuracy and security. Encoder-augmented classifiers with code layer size of 98 or higher are more accurate than the standard LR model. Most augmented classifiers score both higher accuracy and security compared to the LR model trained on noisy data.

The largest deep encoders provide the highest robustness, more than doubling the average minimum attack distance. Classifiers strengthened with single-layer encoders perform similarly by both metrics to those with stacked encoders.

7.2.2.3 Results: Classifying reconstructed representation

In the case of single-layer and stacked autoencoders, classifiers are more robust when connected to the decoder output than to the code layer. For deep autoencoders, the robustness levels do not see much change. All DAE- and SDAE-augmented LR classifiers with more than 24 nodes in the code layer score similar or slightly higher accuracies compared to the standard LR classifier but lower than some denoising encoder-augmented models.

Figure 7.1 confirms that the increased robustness indeed comes from the fact that a DAE can remove adversarial perturbation from the input.

7.3 Conclusion

Contrary to our initial assumption, dimensionality reduction by undercomplete autoencoders does not increase a classifier's robustness to attacks. While some of the autoencoders tested show improved accuracy and security, we have proven that the same gains can be achieved by their non-undercomplete counterparts. In fact, performance and security generally drops with the size of the code layer. This has severe security implications,

Table 7.3: Evaluation of denoising encoders connected to LR classifiers. "DEnc" and "SDEnc" refer to the encoder of a DAE and an SDAE respectively.

Model	Accuracy (%)	TNR (%)	TPR (%)	Average L_2 attack distance
LR	96.61	98.61	94.65	0.97
Noisy LR	94.60	98.22	91.05	1.42
DEnc-784 + LR	98.18	98.61	97.76	1.87
DEnc-392 + LR	98.92	99.21	98.64	1.59
DEnc-196 + LR	98.28	99.01	97.57	1.61
DEnc-98 + LR	97.64	98.32	96.98	1.59
DEnc-49 + LR	95.97	97.52	94.46	1.58
DEnc-24 + LR	95.43	98.32	92.61	1.42
DEnc-12 + LR	94.01	95.34	92.70	1.41
DEnc-784-784 + LR	99.07	99.11	99.03	1.91
DEnc-784-392 + LR	98.97	99.21	98.74	2.04
DEnc-392-196 + LR	98.82	99.01	98.64	1.89
DEnc-196-98 + LR	97.74	98.51	96.98	1.58
DEnc-98-49 + LR	96.96	98.41	95.53	1.64
DEnc-49-24 + LR	95.34	97.42	93.29	1.34
DEnc-24-12 + LR	96.91	97.52	96.30	1.54
SDEnc-784-784 + LR	98.33	98.22	98.44	1.81
SDEnc-784-392 + LR	98.97	99.01	98.93	1.83
SDEnc-392-196 + LR	98.63	98.91	98.35	1.82
SDEnc-196-98 + LR	98.33	98.81	97.86	1.71
SDEnc-98-49 + LR	97.25	98.32	96.21	1.67
SDEnc-49-24 + LR	95.48	96.83	94.16	1.49
SDEnc-24-12 + LR	94.45	96.43	92.51	1.44

Table 7.4: Evaluation of full DAEs and SDAEs connected to LR classifiers.

Model	Accuracy (%)	TNR (%)	TPR (%)	Average L_2 attack distance
LR	96.61	98.61	94.65	0.97
Noisy LR	94.60	98.22	91.05	1.42
DAE-784 + LR	96.96	97.52	96.40	2.17
DAE-392 + LR	97.10	98.22	96.01	2.06
DAE-196 + LR	97.01	98.12	95.91	2.02
DAE-98 + LR	96.71	97.42	96.01	2.00
DAE-49 + LR	96.56	97.22	95.91	1.88
DAE-24 + LR	96.51	97.32	95.72	1.62
DAE-12 + LR	94.80	93.56	96.01	1.78
DAE-784-784 + LR	96.96	97.52	96.40	1.82
DAE-784-392 + LR	97.10	98.22	96.01	2.24
DAE-392-196 + LR	97.01	98.12	95.91	2.06
DAE-196-98 + LR	96.71	97.42	96.01	1.49
DAE-98-49 + LR	96.56	97.22	95.91	1.80
DAE-49-24 + LR	96.51	97.32	95.72	1.55
DAE-24-12 + LR	94.80	93.56	96.01	1.53
SDAE-784-784 + LR	98.04	98.71	97.37	1.96
SDAE-784-392 + LR	97.69	98.81	96.60	1.91
SDAE-392-196 + LR	97.40	99.21	95.62	1.93
SDAE-196-98 + LR	98.04	98.41	97.67	2.03
SDAE-98-49 + LR	97.50	97.72	97.28	1.92
SDAE-49-24 + LR	97.05	97.03	97.08	1.82
SDAE-24-12 + LR	95.29	96.53	94.07	1.62

since autoencoders are popular for squashing the input to machine learning systems.

Secondary to our main objective, we have demonstrated that learning from the code layer of a denoising autoencoder significantly improves the prediction accuracy and doubles the robustness of a classifier through denoising. This suggests that it enables the learning of a decision boundary that is closer to the true one (Figure 7.2), making it an effective way to boost a model's defense without sacrificing accuracy.

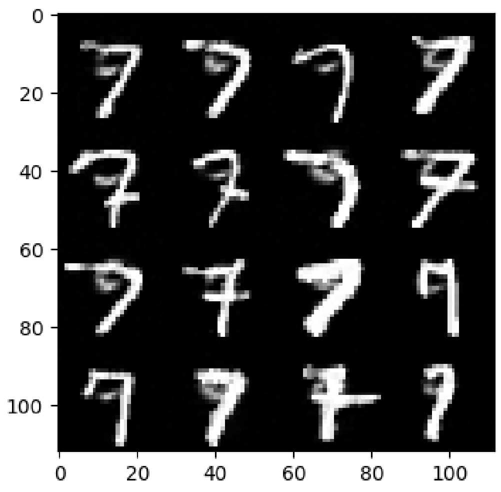


Figure 7.2: Successful attacks against DEnc-784-784

Chapter 8

Effect of Noising Input to DAE

Previously, we saw that LR classifiers augmented with denoising encoders experienced dramatic increases in prediction accuracy as well as doubled robustness against white-box L_2 attacks. In this chapter, we explore the possibility of further improving the robustness of such networks while retaining high accuracy by exploiting the fact that they are trained to remove noise from the input.

8.1 Noising Input to DAE at Inference Phase

We propose the counter-intuitive defense of adding random noise to the input of a denoising encoder-augmented classifier at inference stage. The idea is that since a DAE is trained to identify the original image in the presence of interference (noise), it should be able to handle the same task during inference. The addition of noise means that the DAE has to work with limited information. We theorize that, as a side effect, adversarial perturbations may become obscured under the noise, thus enabling a denoiser to treat it as regular noise (Figure 8.1). In order to prevent the perturbation from being obfuscated, the attacker would have to increase its magnitude in addition to solving a more difficult optimization problem that takes into account the randomness of the noise.

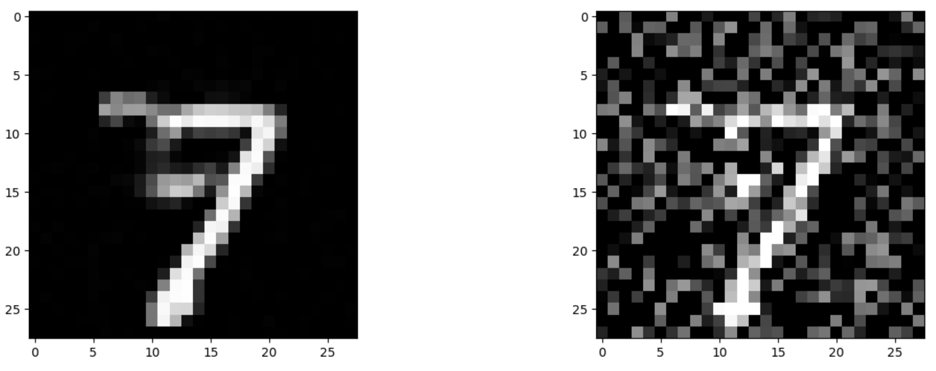
8.2 White-box Attack

We devised a white-box attack that assumes access to the random noise generation function q used by the targeted network g in addition to knowledge of all its parameters. While the attacker cannot predict the exact noise vector that will be added to their next sample, they can ensure, during crafting, that their attack x^* evades g when added separately to k different noise vectors $q_1, q_2, ..., q_k$ generated by q. If the attack is successful when a large number of different noise vectors are added to it, then the attack will likely be successful when added to an unseen noise vector from the same distribution. We theorize that

$$\lim_{k \to \infty} p_e(x^*) = 1$$

where p_e is the probability of evasion. Hence, k directly controls the level of confidence of the attack's success. Algorithm 11 describes the attack generation in full. Notice that the only change compared to Algorithm 9 is that the attacker's prediction loss is now the average of the absolute differences between all noised inputs $x' + q_i$ and the target label y_t.

The attack distance reduction can be done decrementally in the same way as in Algorithm 10 with the new prediction loss function.



(a) successful attack against a deep DAE (b) attack in the presence of uniform noise

Figure 8.1: Obscuring adversarial perturbation with noise. Attack more like to be seen as its original class '7' after adding noise.

Algorithm 11 Attack against classifier with noised input

Input: $\boldsymbol{X}$: source sample; $\boldsymbol{Y^*}$: target output; p: attack norm; c: scale of the distance penalty term; o: output of target network; q: noise generator of target network; k: number of noise vectors added to attack; α: step size of gradient descent; e: minimum L_2-norm of the initial gradient $\nabla l(\boldsymbol{X}, \boldsymbol{X^*})$

Output: $\boldsymbol{X^*}$: adversarial sample

```
1: Generate $q_1, q_2, ..., q_k$
2: Define attacker's prediction loss $u(x) = \frac{1}{k}\sum_i^k |\boldsymbol{Y^*} - o(clip(x' + q_i))|$
3: Define attacker's full loss function $l(x, x') = u(x') + c \cdot \|x' - x\|_p$
4: Initialize $\boldsymbol{X^*}$ where $x_i^* \sim \mathcal{U}(0, 1)$ s.t. $\|\nabla u(\boldsymbol{X^*})\|_2 \geq e$
5: repeat
6:     $\delta \leftarrow \alpha \cdot \nabla l(\boldsymbol{X}, \boldsymbol{X^*})$
7:     $\boldsymbol{X^*} \leftarrow clip(\boldsymbol{X^*} + \delta)$
8: until convergence
```

8.3 Experiment

For this experiment, we focused solely on the LR classifier augmented with a two-layer denoising encoder, each layer with 784 nodes ("DEnc-784-784 + LR"), from the previous experiment, as the network has the highest accuracy and nearly highest robustness.

We evaluated its performance and security by noising the input with uniform noise of magnitudes, 0.2, 0.4, 0.6, 0.8 and 1.0. While the DAE was trained with $\tilde{v} \sim [\mathcal{U}(-1,1)]^f$, we wished to test whether it operated better at lower noise levels. Unlike in the previous chapters, we did not craft minimal attacks, as we found, empirically, that attacks too close to the decision boundary had high chances of failing. Instead, we crafted the attacks as described in Algorithm 11 with $k = 30$ for high confidence and then tested them against the target model at various stages of reduction. We classified each batch of attacks 100 times, as the results would vary in each run, and recorded the average evasion rate as well as the standard deviation thereof. Instead of crafting attacks from all 1028 class 1 test samples, we had to limit ourselves to a randomly chosen subset thereof with 100 samples, as computing the new gradient was extremely slow for $k = 30$. As a baseline, we crafted minimal attacks from this subset that target the same network without noised input.

For accuracy evaluation, we classified the set of clean test samples 100 times and recorded the same metrics as above.

8.4 Results

Figures 8.2, 8.3, 8.4, 8.5 and 8.6 show the evasion rates of white-box attacks against denoising encoder-augmented LR classifiers with different levels of noise added to the input at inference stage. We can observe that, in general, as the noise strength increases, so does the effort required for the attacker to achieve high evasion rates. However, noise levels of 0.8 or higher also result in noticeably higher evasion rates for low-effort attacks. They are also associated with high variances in evasion rate across different runs, implying they cause great uncertain in the decision boundary.

Comparing the results numerically (Table 8.1) , the evasion rate drops to as low as 73.34 (%) for the noise level of 1.0, despite the fact that each attack was crafted to evade the classifier when added to 30 different noise patterns from the same distribution. This means that the attackers needs significantly higher k to achieve a near perfect evasion rate. It stands to reason that as k increases, so does the attack distance, as the attack has to be very close to the target class for it be identified as such under many noise patterns.

In terms of performance, noise levels under and including 0.6 sacrifice very little accuracy for significant gains in security. The highest noise level causes the accuracy to drop significantly. Presumably, the noise is so strong that samples of one class often gets mistaken for the other class.

Overall, adding either $[\mathcal{U}(-0.4,0.4)]^f$ or $[\mathcal{U}(-0.6,0.6)]^f$ to the input seems to be the best compromise in terms of accuracy loss and security gain.

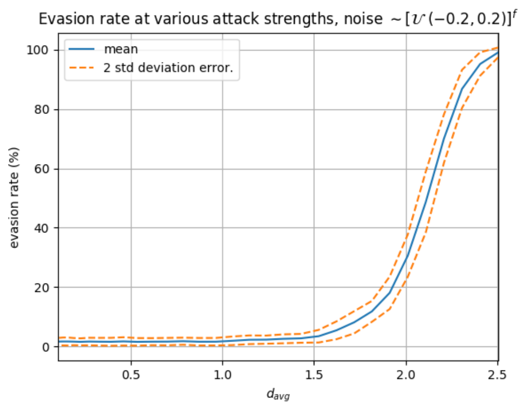


Figure 8.2

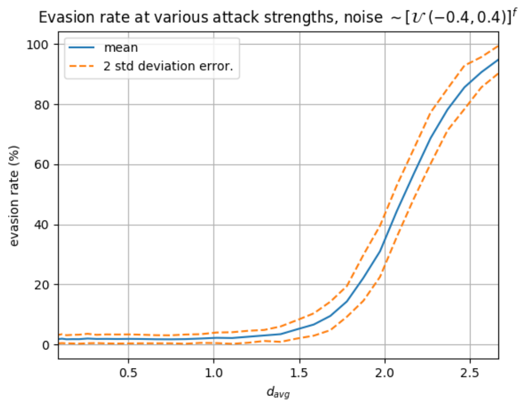


Figure 8.3

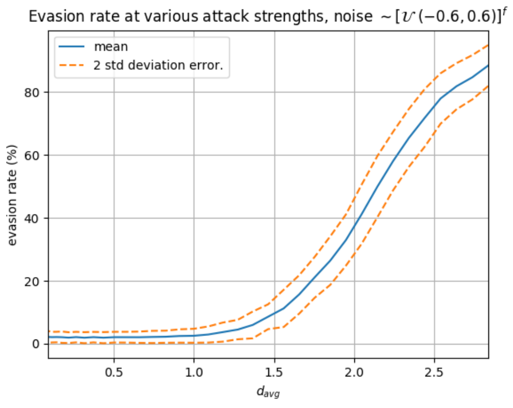


Figure 8.4

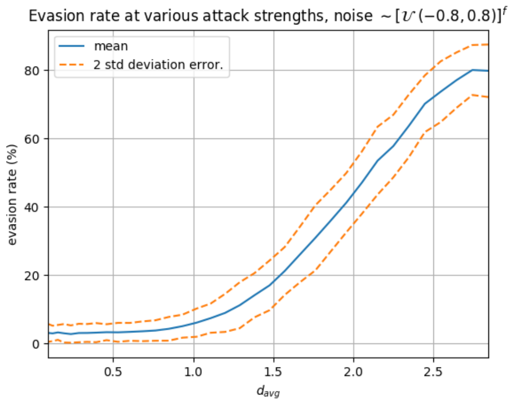


Figure 8.5

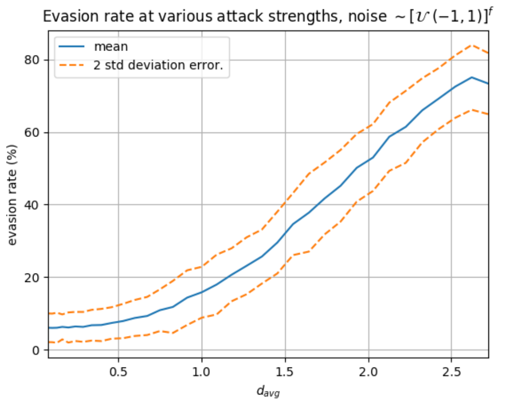


Figure 8.6

Table 8.1: Accuracy, mean attack distance and evasion rate of L_2 attacks at different uniform noise levels

Noise level	Mean accuracy (%)	Mean L_2 attack distance	Evasion rate (%)
0	99.07	1.93	100
0.2	98.99 ± 0.21	2.51	99.10 ± 0.82
0.4	98.88 ± 0.28	2.67	94.78 ± 2.30
0.6	98.66 ± 0.32	2.84	88.45 ± 3.26
0.8	98.15 ± 0.42	2.84	79.76 ± 3.84
1	96.29 ± 0.71	2.72	73.34 ± 4.21

8.5 Conclusion

We confirmed our initial assumption by showing that adding random noise to the input of a denoising encoder-augmented MNIST classifier can increase its robustness by 50% without sacrificing much prediction accuracy. Considering that the classifier by itself is already quite robust, this improvement is significant and the perturbation level is close to fooling human perception. Furthermore, our technique is extremely easy to implement and can greatly increase the effort required to find high confidence attacks.

Using randomness to defend against adversarial samples is an idea that was also proposed by Feinman et al. [10]. Their technique detects adversarial samples by randomly dropping out (deactivating) a portion of the weights. By repeating this process multiple times on an input, they can compute an uncertainty value associated with the sample. The idea is that the uncertainty level will be higher for adversarial samples than non-adversarial ones. Carlini et al. [5] showed it to be the most effective detection method among the 10 that they evaluated.

Our findings reinforce the idea that randomness is effective in preventing evasion attacks, as it prevents the adversary from taking advantage of flaws in the weight assignments.

Chapter 9

Conclusion

Below we summarize the results of this paper:

- We showed that by fairly evaluating different models trained with Lasso, smaller models are generally more secure.
- We proved that it is possible to gain 50% or more security by using Lasso without sacrificing much accuracy.
- We proved that attacks against smaller models trained with Lasso are easier to detect through statistical means compared to larger models.
- We disproved our assumption that using weight polarity to guide feature selection allows trade-off between true positive and true negative rates.
- We disproved our assumption that using traditional autoencoders to perform dimensionality reduction on the input increases classifier security. In fact, the compression effect can sometimes make the system more vulnerable to attacks.
- We verified our assumption that denoising autoencoders mitigate attacks by partially removing adversarial noise.
- We showed that it is possible to further enhance the denoising simply by applying noise to the input at inference phase. It is possible to increase the attack distance by 50% with little loss in accuracy. The findings reinforce the value of randomness in defending against attacks.

9.1 Limitations and Future work

- All experiments were performed on datasets with binary labels. It would be valuable to explore how the findings scale to multiclass classification tasks, where the decision boundaries are less clear.
- The three datasets used in our experiments had linear structures. It is worth experimenting with datasets containing more complex data such as CIFAR-10.
- We only looked at one type of classifier throughout the experiments. We could investigate how the results transfer to other classifiers.
- We only examined one popular feature selection technique, Lasso. Future work could explore how other widely used techniques affect security using our methodology.

- It would be interesting to check how our noising defense fares against other randomized defense methods.
- Apart from randomization, adversarial training [19] have shown to be amongst the most promising defenses against evasion attacks. We could explore the possibility of training denoising autoencoders to remove adversarial noise specifically.

Bibliography

[1] Roberto Battiti. “Using mutual information for selecting features in supervised neural net learning”. In: *IEEE Transactions on neural networks* 5.4 (1994), pp. 537–550.

[2] Arjun Nitin Bhagoji, Daniel Cullina, and Prateek Mittal. “Dimensionality Reduction as a Defense against Evasion Attacks on Machine Learning Classifiers”. In: *arXiv preprint arXiv:1704.02654* (2017).

[3] Battista Biggio, Giorgio Fumera, and Fabio Roli. “Security evaluation of pattern classifiers under attack”. In: *IEEE transactions on knowledge and data engineering* 26.4 (2014), pp. 984–996.

[4] Battista Biggio et al. “Evasion attacks against machine learning at test time”. In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases.* Springer. 2013, pp. 387–402.

[5] Nicholas Carlini and David Wagner. “Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods”. In: *arXiv preprint arXiv:1705.07263* (2017).

[6] Nicholas Carlini and David Wagner. “Towards evaluating the robustness of neural networks”. In: *Security and Privacy (SP), 2017 IEEE Symposium on.* IEEE. 2017, pp. 39–57.

[7] Corinna Cortes and Vladimir Vapnik. “Support-vector networks”. In: *Machine learning* 20.3 (1995), pp. 273–297.

[8] Marc Deisenroth and Stavros Petridis. *Lecture Notes. Course C495: Mathematics for Machine Learning.* 2017.

[9] John Duchi et al. “Efficient projections onto the l 1-ball for learning in high dimensions”. In: *Proceedings of the 25th international conference on Machine learning.* ACM. 2008, pp. 272–279.

[10] Reuben Feinman et al. “Detecting adversarial samples from artifacts”. In: *arXiv preprint arXiv:1703.00410* (2017).

[11] Robert Fortet and E Mourier. “Convergence de la répartition empirique vers la répartition théorique”. In: *Annales scientifiques de l’École Normale Supérieure.* Vol. 70. 3. Elsevier. 1953, pp. 267–285.

[12] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. “Explaining and harnessing adversarial examples”. In: *arXiv preprint arXiv:1412.6572* (2014).

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* http://www.deeplearningbook.org. MIT Press, 2016.

[14] Kathrin Grosse et al. “On the (statistical) detection of adversarial examples”. In: *arXiv preprint arXiv:1702.06280* (2017).

[15] Ling Huang et al. “Adversarial machine learning”. In: *Proceedings of the 4th ACM workshop on Security and artificial intelligence.* ACM. 2011, pp. 43–58.

[16] Jesse Johnson. *General regression and over fitting*. Mar. 2013. URL: https://shapeofdata.wordpress.com/2013/03/26/general-regression-and-over-fitting/ (visited on 01/25/2018).

[17] Saurav Kaushik. *Introduction to Feature Selection methods with an example*. Dec. 2016. URL: https://www.analyticsvidhya.com/blog/2016/12/introduction-to-feature-selection-methods-with-an-example-or-how-to-select-the-right-variables/ (visited on 01/25/2018).

[18] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. "Adversarial examples in the physical world". In: *arXiv preprint arXiv:1607.02533* (2016).

[19] Aleksander Madry et al. "Towards deep learning models resistant to adversarial attacks". In: *arXiv preprint arXiv:1706.06083* (2017).

[20] James D. McCaffrey. *The Max Trick when Computing Softmax*. Mar. 2016. URL: https://jamesmccaffrey.wordpress.com/2016/03/04/the-max-trick-when-computing-softmax/ (visited on 01/25/2018).

[21] Patrick McDaniel, Nicolas Papernot, and Z Berkay Celik. "Machine learning in adversarial settings". In: *IEEE Security & Privacy* 14.3 (2016), pp. 68–72.

[22] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997, p. 2.

[23] Andrew Ng. *Machine Learning Online Course*.

[24] Nicolas Papernot et al. "Distillation as a defense to adversarial perturbations against deep neural networks". In: *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE. 2016, pp. 582–597.

[25] Nicolas Papernot et al. "The limitations of deep learning in adversarial settings". In: *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*. IEEE. 2016, pp. 372–387.

[26] Arthur L. Samuel. "Some Studies in Machine Learning Using the Game of Checkers". In: *IBM Journal of Research and Development* 3.3 (July 1959), pp. 210–229.

[27] Daniele Sgandurra et al. "Automated dynamic analysis of ransomware: Benefits, limitations and use for detection". In: *arXiv preprint arXiv:1609.03020* (2016).

[28] Cencheng Shen. *The Swiss Roll Matching Example*. URL: http://www.cis.jhu.edu/~cshen/html/PublishSwissRoll.html (visited on 01/25/2018).

[29] Carl-Johann Simon-Gabriel et al. "Adversarial Vulnerability of Neural Networks Increases With Input Dimension". In: *arXiv preprint arXiv:1802.01421* (2018).

[30] Christian Szegedy et al. "Intriguing properties of neural networks". In: *arXiv preprint arXiv:1312.6199* (2013).

[31] Florian Tramèr et al. "Ensemble adversarial training: Attacks and defenses". In: *arXiv preprint arXiv:1705.07204* (2017).

[32] Pascal Vincent et al. "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion". In: *Journal of Machine Learning Research* 11.Dec (2010), pp. 3371–3408.

[33] Fei Wang, Wei Liu, and Sanjay Chawla. "On sparse feature attacks in adversarial learning". In: *Data Mining (ICDM), 2014 IEEE International Conference on*. IEEE. 2014, pp. 1013–1018.

[34] Fei Zhang et al. "Adversarial feature selection against evasion attacks". In: *IEEE transactions on cybernetics* 46.3 (2016), pp. 766–777.