Adversarial Machine Learning: A review of the "Adversarial Robustness Toolbox (ART)"

Ca' Foscari University of Venice
Department of Environmental Sciences, Informatics and Statistics

Computer Science Master's Thesis
Academic Year 2021-2022

Graduand Habtamu Desalegn Woldeyohannes - 877159
Supervisor Claudio Lucchese

Acknowledgments

First of all, Praise to God! Everything in my life happened because of your blessing. Then, I would like to thank all my beloved families especially my mother Zimam W/Gaber, Tizazu Desalegn (brother), and to my love Addisalem Haile for her patience and love.

Next, I would like to give special thanks my thesis supervisor, Dr. Claudio Lucchese for his support and guidance throughout during this study. I wish I work with you in future too. Then, I would like to thank all our professors for the best training, encourage and support in these last two and half years. At last to my best classmates batch 2018/19 for good memories and supports, Thank you so much.

Abstract

Adversarial Robustness Toolbox (ART) is an open-source project for machine learning security by IBM research. ART implements many novel adversarial attacks and defenses, it can be used by researchers as a standard benchmark for novel adversarial attack and defense techniques, and it is considered as a tool for developers to build and deploy secure machine learning systems that are resilient against adversarial attacks. In this thesis, we review a recent version of ART Python library.

We cover data poisoning and evasion attacks supported in current version of "Adversarial Robustness Toolbox v1.5.1". We evaluate the performance results of those attacks against machine learning models for classification tasks on image and tabular data in adversarial setting. Specifically, we evaluate those attack methods ART supports against four supervised machine learning algorithms: support vector machines (SVM), decision trees (DT) and random forest (RF) trained with scikit-learn; and gradient-boosted decision trees (GBDT) trained with lightGBM, and two publicly available datasets (i.e. census income dataset and MNIST handwritten digit database for tabular and image data respectively).

Keywords Adversarial Robustness Toolbox, Adversarial Attacks, Evasion Attacks, Poisoning Attacks, Adversarial Examples

Contents

List of Figures

List of Tables

Abbreviations

ART	Adversarial Robustness Toolbox
AI	Artificial Intelligence
ML	Machine Learning
AML	Adversarial Machine Learning
DNN	Deep Neural Networks
SVM	Support Vector Machines
DT	Decision trees
RF	Random forest
GBDT	Gradient-boosted decision tree
TP	True Positive
FP	False Positive
TN	True Negative
FN	False Negative
TPR	True Positive Rate
FPR	False Positive Rate
MCC	Matthews Correlation Coefficient
ROC	Receiver Operating Characteristic
ROC AUC	Area Under the ROC Curve
FGM	Fast Gradient Method
FGSM	Fast Gradient Sign Method
PGD	Projected Gradient Descent
BIM	Basic Iterative Method
UP	Universal Perturbation
UAPs	Targeted Universal Adversarial Perturbations
HSJA	HopSkipJump Attack
BA	Boundary Attack
ZOO	Zeroth-order optimization
EAD	Elastic-Net Attack
C&W	Carlini & Wagner
JSMA	Jacobian Saliency Map Attack
VAT	Virtual Adversarial Method

Chapter 1

Introduction

Despite the success of Machine Learning (ML)/AI systems, ML models have been shown to be vulnerable to adversarial examples i.e, maliciously perturbed examples, which are undetectable to the human, but mislead models to make incorrect classification ([19], [13], [27], [34] [11]). The vulnerability of ML models to adversarial examples exposes a security risk in ML/AI systems such as computer vision, image classification, fraud detection, spam detection, malware detection, and beyond ([3], [23], [31], [25], [12], [10], [39], [34], [11], [13], [7]).

Nicolae et al., (2018) introduce Adversarial Robustness Toolbox (ART)[1], which is an open source machine learning security library developed at IBM research using Python programming language. ART used as a tool to test adversarial robustness of ML models with many state-of-the-art adversarial attacks and defense methods, and supports most known machine learning frameworks.

ART contains Attacks i.e security attacks (i.e. poisoning attacks and evasion attacks) and privacy attacks (i.e. inference attacks and extraction attacks), Defenses (i.e adversarial training, preprocessing, post-processing, detectors, and transformer), and Model Robustness (i.e certificator, metrics, and verification) tools (Nicolae et al., 2018).

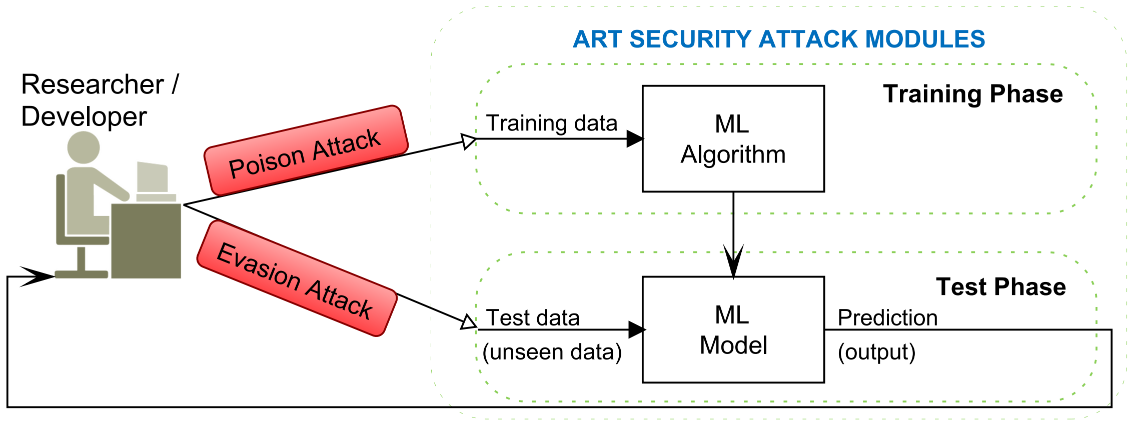


Figure 1.1: ART evasion and poison attack on ML pipeline

[1]https://github.com/Trusted-AI/adversarial-robustness-toolbox

Adversarial attacks are a type of attacks performed to fool the targeted classifier in ML, in which the classifier assigns the example to the wrong class. However, researcher or developer wants to understand the problem and under what situation the attack methods fails, and how well can defend in those attacks to the ML models using ART library is illustrated in Figure 1.1.

Many researchers proposed adversarial attacks and defense mechanisms for Deep Neural Networks (DNN). However adversarial attack is not limited to DNN. Emphasize that, in this paper we aim at review on adversarial attacks on traditional machine learning algorithms. Particularly, we study the ART evasion and poison attacks against support vector machines, decision trees, random forest, gradient-boosting decision trees approaching classification tasks on tabular and image data types.

The contributions of this thesis are as follows:

- We experiment evasion and data poisonous attacks on four machine learning models for tabular and image data type and highlights the weakness and strength of those attack algorithms implemented in ART.
- We assess the effectiveness of adversarial examples produced using traditional machine learning models rather than deep neural network models.

1.1 Overview

The rest of the document is organized as follows: Chapter 2 introduces supervised machine learning, particularly classification models; then discuss key concepts on adversarial attacks; Chapter 3 reviews the adversarial attacks (i.e., evasion and poisoning attacks) implemented in ART; Chapter 4 discusses the experimental result and analysis of the study; Chapter 5 summary and conclude this document.

Chapter 2

Background

This chapter has two sections. Section 1, gives some theoretical background for supervised learning problems, specifically, the classification models followed by classification algorithms and metrics used in our experiments. In section 2, we discuss basic adversarial machine learning concepts, in which our focus is on adversarial attacks. Specifically, the security attacks on ML models, in which an attacker generates adversarial examples against ML models aiming to fool the target model.

2.1 Machine Learning

Mitchell (1997) provides the founding definition of Machine Learning "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E". For example, consider the Census Income dataset from the UCI ML Repository, the task is to predict yearly income of a person be greater than \$50k or not:

- Task T: classifying the yearly income of a person be greater than \$50k or not from tabular data.
- Performance measure P: percentage of yearly income of persons classified correctly.
- Training experience E: yearly income dataset, i.e., Census Income dataset from the UCI ML Repository.

Machine learning can be categorized into the following:

- **Supervised Learning** is a learning task in which examples of labeled data are used in learning classification or regression models. Classification models used to classify discrete class label. whereas, Regression models used to predict continuous values.
- **Unsupervised Learning** is unlike supervised learning tasks, we use unlabeled data to find or cluster the hidden structure of the data by using the clustering, dimensional reductions, and others when labeled training data are not available.

- **Semi-supervised learning** is the combination of supervised and unsupervised machine learning tasks.
- **Reinforcement Learning** is a learning task in which an agent performs continuous actions while observing the result (or rewards).

2.1.1 Supervised Learning

Given a training set $\mathcal{D}_{train} = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}$ of correctly labeled examples and set of hypotheses $\mathcal{H}$, the learning stage of ML algorithm is to find the function $f^* \in \mathcal{H}$ while minimizing a loss function. We define a hypothesis function $f(\mathbf{x})$ that estimate some unknown target function, and if there exists a target function, we can define as:

$$f^* : \mathcal{X} \rightarrow \mathcal{Y} \quad \Rightarrow \quad y_i \approx f^*(\mathbf{x}_i) \; for \; i = 1, ..., n \tag{2.1}$$

where $\mathbf{x}_i$ is i-th input feature vector in the dataset and y_i is i-th class label in classification problems.

However, finding the target function to predict y requires to search all $f \in \mathcal{H}$ and it is computationally infeasible. Loss functions measures the error difference between a predicted label using hypothesis f and a true label. When the loss function is defined as L^2 loss (mean squared error):

$$\ell(f(\mathbf{x}), y) = \frac{1}{2}(f(\mathbf{x}) - y)^2 \tag{2.2}$$

and, the empirical risk or training loss for a set of labeled examples $\mathcal{D}_{train}$ is defined as:

$$L_n = \frac{1}{n}\sum_{i=1}^{n} \ell(f(\mathbf{x}_i), y_i) \tag{2.3}$$

Classical machine learning algorithms are based on empirical risk minimization (Vapnik, 1992) approach to search the best hypothesis $f^* \in \mathcal{H}$ that minimizes the loss function on the given training set $\mathcal{D}_{train}$. In addition to that, the identified hypothesis function f^* will generalize to test (unseen) data.

$$f^* = \arg\,\min_{f\in\mathcal{H}} \sum_{i=1}^{n} \ell(f(\mathbf{x}_i), y_i) \tag{2.4}$$

Before starting the implementation of the support vector machines, decision trees, random forest and gradient-boosted decision trees classifiers using Scikit-learn and LightGBM, some basic concepts on those classifiers will be presented below.

Support Vector Machines (SVM): Given a training set $\mathcal{D}_{train} = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}$, the learning stage of an SVM algorithm is to separate the training data into classes using a hyperplane by choosing the maximum margin while minimizing the loss function. SVM also know as ”a maximum margin classifier”. Figure 2.3(a) shows an example of decision boundary separates the two classes with maximum margin by SVM with Linear kernel model. Sometimes, the robustness of the model depends on the generalization performance, meaning when the classifier is best (or have wide margin), then it is not easily affected by small perturbations which leads to misclassification on the decision boundary (Biggio et al., 2012).

Decision trees (DT): Given a training set $\mathcal{D}_{train} = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}$, decision trees are constructed by recursively splitting the training set. Each split is based on a single feature with a threshold condition, in which successive binary splits of training set into smaller and smaller pieces. However, finding the best sequence of split rules is NP-Complete problem (Murthy, 1998). Hence, Breiman et al., (1984) introduce a heuristic algorithm called Classification And Regression Tree (CART) algorithm, which incrementally adds nodes to a decision tree, starting from the root. At test time, an instance $\mathbf{x}$ is classified by traverses the decision tree from its root until it reaches a leaf, i.e., the class assigned in according to conditions specified in the tree internal nodes .

Random forest (RF) is an ensemble learning methods for decision trees. RF is an ensemble predictor consists of set of trees which are trained independently and combined by a majority voting to assign the class label. Each tree constructed with bagging and per-node feature sampling over the training set (Breiman, 2001).

Gradient-boosted decision trees (GBDT) are an ensemble learning methods for decision trees, which uses an iterative procedure to generate one single stronger model by combining the decisions of the weak learners produced i.e. decision tree models.

Given a training set $\mathcal{D}_{train} = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}$ and set of hypotheses $\mathcal{H}$ is the set of decision trees, GBDT algoithm identifies the set of functions $f_t \in \mathcal{H}$ while minimizing the loss function.

GBDT uses classification and regression trees (CART) as base learners and gradient boosting aims to minimize the loss function in every iteration. The approximation of $\hat{y}_i$ w.r.t the i-th labeled example $\mathbf{x}_i$ in additive form is defined as:

$$\hat{y} = \sum_{t=1}^{K} f_t(\mathbf{x}_i), \; f_t \in \mathcal{H} \tag{2.5}$$

where K is the number of trees.

Then the base function in case of CART algorithm is defined as:

$$f_t(\mathbf{x}_i) = w_{c(\mathbf{x}_i)} \tag{2.6}$$

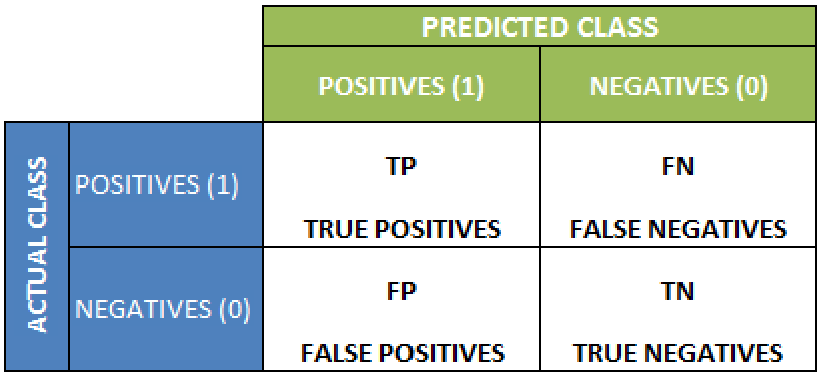
where $w \in \mathbf{R}^T, c : \mathbf{R}^d \implies \{1, 2, .., T\}$. T is the number of leaves. For input $\mathbf{x}_i \in \mathbf{R}^d$, $c(\mathbf{x}_i)$ function outputs the i-th estimation value of the leaf node of the decision tree.

Then the loss function is defined as:

$$\arg\ \min_{f \in \mathcal{H}} \sum_{i=1}^{n} \ell(f_t(\mathbf{x}_i), y_i) \tag{2.7}$$

where n is the number of examples in the training (Lin et al., 2020).

Classification metrics are evaluation metrics used to evaluate the performance of classification models. these metrics are built based on the confusion matrix (see Figure 2.1), in which the columns represent the number of examples in the predicted class and the rows represents the number of examples in the actual class. TP and TN indicate both actual and predicted values are 1 and 0 respectively. FN indicates actual and predicted values are 1 and 0 respectively. Whereas, FP indicates actual and predicted values are 0 and 1 respectively. In case of binary classification problem, The classifier's wrong prediction will be recorded in FP or FN.



		PREDICTED CLASS	
		POSITIVES (1)	NEGATIVES (0)
ACTUAL CLASS	POSITIVES (1)	TP TRUE POSITIVES	FN FALSE NEGATIVES
	NEGATIVES (0)	FP FALSE POSITIVES	TN TRUE NEGATIVES

Figure 2.1: Confusion matrix.

Metrics used to measures the performance of our ML models are as follows:

- Accuracy is defined as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + TN} \tag{2.8}$$

  Whereas (TP+TN) is the number of correct predictions and (TP+TN+FP+TN) is total number of predictions. The above formula works to evaluate binary classification, and generally for multiclass classification, the accuracy will be calculated as the fraction of all correctly predicted (on the diagonal of the confusion matrix) over all examples.

- Macro Accuracy (Balanced Accuracy) is defined as:

$$\text{Macro Accuracy} = \sum_{i=1}^{k} \frac{\frac{TP_i+TN_i}{TP_i+TN_i+FP_i+TN_i}}{k} \tag{2.9}$$

  where k is the number of class labels and the fraction of correctly predicts is computed based on one-vs-all confusion matrix.

- Precision is defined as:

$$Precision = \frac{TP}{TP + FP} \tag{2.10}$$

Where TP is the number of correctly predicted positive values, and (TP + FP) will be all the positive predictions. In the census case, precision indicates the proportion of persons that the model guesses with \$50k and above income and actually have mentioned income.

- Recall is defined as:
$$Recall = \frac{TP}{TP + FN} \tag{2.11}$$
Where TP is the number of correctly predicted positive values, and (TP + FN) will be all actually positive values. In the census case, recall indicates the proportion of persons having \$50k and above income that are correctly guessed by the model as having income \$50k and above.

- F_1 score is defined as:
$$F_\beta = (1 + \beta^2)\frac{Precision \cdot Recall}{(\beta^2 \cdot Precision) + Recall} \tag{2.12}$$
$$F_1 = 2\frac{Precision \cdot Recall}{Precision + Recall} \tag{2.13}$$
Where F_β , a parameter β controls the importance of each assigned term and popular setting is β=1 which is known as F_1 score, precision is a positive predictive value, and recall is a true positive rate. As in the formula specified above, F_1 score measures a weighted harmonic mean between precision and recall (Fernández et al., 2018).

- Matthews correlation coefficient (MCC) defined as:
$$MCC = \frac{TP + TN - FP + FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{2.14}$$
Where TP,TN,FP and FN are all 4 categories of the confusion matrix, which takes advantage over F_1 score because of considering errors and correct classification in both classes. The result of MCC ranges between -1 and 1 (similar to correlation), where MCC=1 indicates the classification is always correct, MCC=0 indicates the classification used is a random guess, and MCC=-1 when the classification is wrong. Note that, MCC is useful where unbalanced class data occurs (Fernández et al., 2018).

- Area Under the ROC Curve (ROC AUC) defined on the basis of
$$\begin{aligned} TPR &= \frac{TP}{TP + FN} \\ FPR &= \frac{FP}{FP + TN} \end{aligned} \tag{2.15}$$
Where TPR is True Positive Rate, FPR is False Positive Rate, and the ROC Curve plots constructed based on the two parameters: TPR versus FPR at different model thresholds. ROC curve used to determine the right threshold value and AUC (Area under the ROC curve) indicates model quality by

considering the area under the curve, the larger the area indicates the model quality. ROC AUC computes an aggregated single score from the ROC curve based on the overall performance of the model thresholds, ranging between 0.5 and 1, A model whose guesses are 100% correct has ROC AUC of 1.0; one whose model guesses are 50% correct has ROC AUC of 0.5, which will be achieved by any randomly chosen model. ROC AUC is the preferred evaluation metric for class imbalanced datasets (Fernández et al., 2018).

2.2 Adversarial Machine Learning

There are two main categories of adversarial attacks on ML systems. One category focuses on the privacy attacks on ML/AI systems, which extracts ML models and collects sensitive data/attributes. The other category is about security attacks on ML/AI systems, which generates adversarial examples, i.e., evasion and poisoning attacks. The wide use of such kinds of attacks due to ML models are vulnerable to adversarial examples, Adversarial Machine Learning (AML) becomes the study of ML that deals with protecting ML pipeline at training, test and inference time (Nicolae et al., 2018).

2.2.1 Adversarial Attacks

Adversarial attacks are algorithms for seeking adversarial perturbations. For example, attack algorithm uses ℓ_p norm distance changes to create an adversarial example, i.e., input to a ML models that is designed on purpose to fool the targeted classifier and are often imperceptible (or valid input) to humans (Brendel et al., 2018; Goodfellow et al., 2014). Here, the goal of adversarial attacks is to minimize the perturbation magnitude while creating adversarial example under ℓ_p distance metric, for grayscale MNIST images with 784 pixel values, the ℓ_0 distance between original and adversarial image is the number of pixels where the intensity value is different. whereas, the ℓ_∞ distance is the maximum of intensity differences among pixels of the two images. Formally, the distance metric d($\cdot$) between the original **x** and adversarial example **x'** is less than the attack budget (denoted as ϵ), while fooling the targeted classifier C to assign the example to the wrong class.

$$\begin{aligned} &d(\mathbf{x}, \mathbf{x'}) < \epsilon \\ &where \;\; C(\mathbf{x}) \neq C(\mathbf{x'}) \end{aligned} \tag{2.16}$$

According to the attacker's knowledge and accessible of the targeted ML system, we can categorize adversarial attacks into three category such as:

1. **White-box attacks:** the attacker has full access to the classifier's architecture, parameters and evaluate loss/class gradients on the training data (Chen et al., 2017).

2. **Black-box attacks:** the attacker doesn't know any information about the targeted model except output (class labels) produced to the given input. Black-box attacks (a.k.a query-based attacks) are performed considering the feedback provided by querying the classifier with adversary manipulated data to learn the true labels based on the predicted class label or confidence score (Chen et al., 2017).

3. **Gray-box attacks:** the attacker execute attacks having knowledge of the targeted classifier architecture, algorithm or training data only (Nicolae et al., 2018).

In white-box attacks, attackers able to calculate gradient w.r.t input example but in black-box (or gray-box) attacks doesn't. In the case of our review, we look at 13 white-box and 3 black-box attack methods implemented in ART.

When we consider attack objectives, there are two objectives of the attacker:

1. **Untargeted attack**: Given an original example with correct labeling, by untargeted attack, the attacker aims to create an adversarial example that lead to misclassification (or different from the corresponding original example class label).

2. **Targeted attack**: Given an original example with correct labeling, the goal of the targeted attack is to modify the original example aiming to be classified as the specified target class (Nicolae et al., 2018; Chen et al., 2017).

Evasion Attack

Evasion attack (a.k.a. adversarial examples) is carefully perturbing the input to a classifier such that it is misclassfied at test time. Like DNN, this attack applies to gradient-based models, such as SVM using gradient descent optimization techniques (Biggio et al., 2013). However, for models without gradient, for example, random forest uses input binarization, which transforms input into an array of {0,1} based on given threshold (Chen and Jordan, 2019).

Based on Brendel et al. (2018), adversarial attack algorithms are categorized into four categories: gradient-based, score-based, decision-based, and transfer-based (see Table 3.1). However, the Decision Tree attack implemented in ART is categorized according to the method used, so we categorized it under decision tree-based attack.

1. **Gradient-based attack** uses model gradient of the training data to create adversarial example in white-box setting.

2. **Score-based attack** works by accessing the predicted confidence scores of the model, i.e., logit or output probabilities used to create adversarial example in black-box setting.

3. **Decision-based attack** uses final decision of the model, i.e., class label to create adversarial example in black-box setting. In contrast to the score-based attack, it uses the final model prediction. In our work, all ART attack algorithms in this category supports SVM, DT, RF, and GBDT models (see Table 3.1).

4. **Transfer-based attack** uses information of the training data, i.e., with out model information to create adversarial example.

5. **Decision tree-based attack** uses decision tree structure to create adversarial example for decision tree model only. Example, ART Decision Tree attack.

Poisoning Attack

In contrast to evasion attacks where attacks against ML model at test time, poisoning attack is injecting poisonous data during model training and data collection phase of ML pipeline (Nicolae et al., 2018; Biggio et al., 2012).

Biggio et al., (2012) proposed an attack on support vector machines algorithm by injecting carefully crafted attack points into training data that will maximize the classification errors on the unseen test data, as a result the model generalization performance is decreased.

The below example shows data poisoning attack on SVM scikit-learn model, which can slip the decision boundary such that the classifier makes wrong prediction to the examples given at test time (see Figure in 2.3(b)).

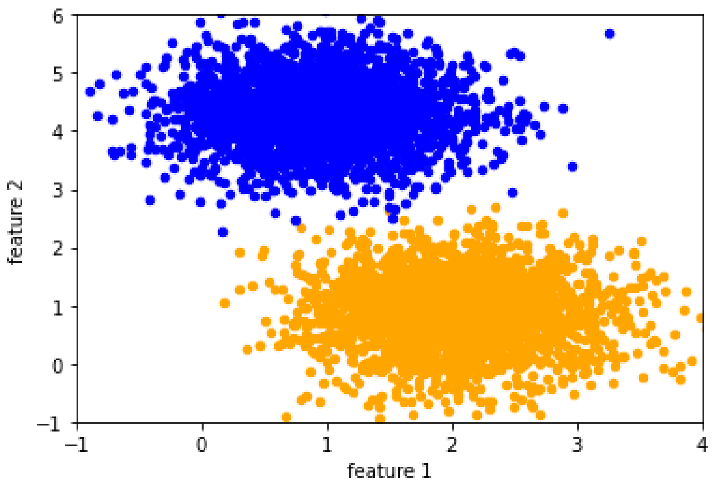


Figure 2.2: Sample data points representing in two-dimensional space.

Let consider synthetic data, which has a set of 5k data points with 2 class labels generated by make_blobs[2], which are linearly separable as shown in Figure 2.2. Now for simplicity fit the model using the linear kernel of a SVM classifier on training data points will be obtained in Figure 2.3(a).

On model training, the best decision boundary for the classifier is represented in two-dimensional space. This decision boundary is selected to best separate the data points by their class labels, either 0 (blue) or 1 (yellow). In Figure 2.3(a) the support vectors are shown in circles, and the classifier used only those support vectors to make decision boundary for separation between the two class labels, whereas other data points are not considered to maximize the margin of the decision boundary.

Now we will consider the scenario of using specially crafted attack points in model training by choosing randomly 50 data points with "1" class and alter their class

[2]https://scikit-learn.org/

labels. As a result, we can notice in Figure 2.3(b) that the decision boundary of the targeted classifier has slightly shifted, changes in the number of support vectors and increased margin cause of the contaminated data (noise). The support vectors are shown in circles and the attack point in red. In this example, the attacker provides 50 poisonous data points in the targeted setting, As a result, the attack influence the change in the decision boundary. Since the decision boundary is changed, the model generalization performance will be decreased for unseen test examples as seen in Figure 2.4.

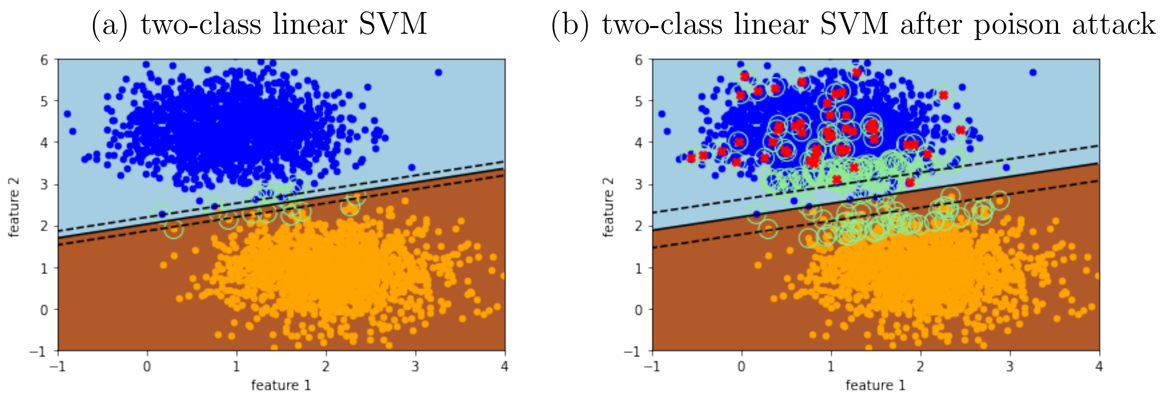


Figure 2.3: Training data (70%): ART Poisoning attack on SVM

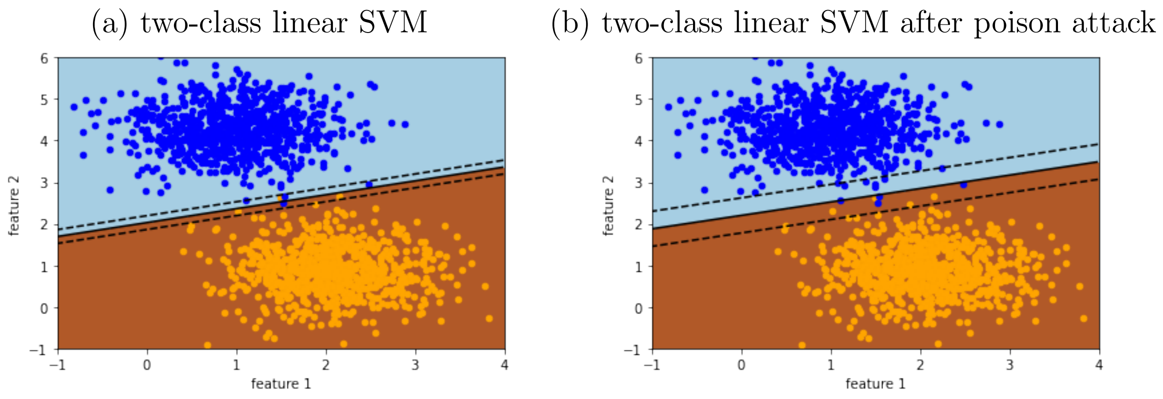


Figure 2.4: Test data (30%): Clean and Poison SVM model

Chapter 3

ART Attacks

In this chapter, we consider adversarial threats to ML/AI systems implemented in ART, i.e., evasion and data poisoning attacks only, which are adversarial attacks against ML models during testing and training time, respectively. Adversarial examples created by ART adversarial attack methods are evaluated on the following supervised machine learning classes: support vector machines (SVM), decision trees (DT), random forest (RF), and gradient-boosting decision trees (GBDT) trained on handwritten digit images from MNIST and census income dataset.

ART library implements many advanced adversarial attack algorithms, which are used to create adversarial examples for most well known ML/DNN models and its applications. In the benchmark experiments or development of a secured ML system, ART provides framework-agnostic library modules, i.e., an implementation of Python code for the state-of-the-art adversarial attacks, defenses, and robustness metrics. ART 1.5.1 library has a total of 37 security attacks on ML/AI systems, including 32 evasion attacks and 5 data poisoning attacks.

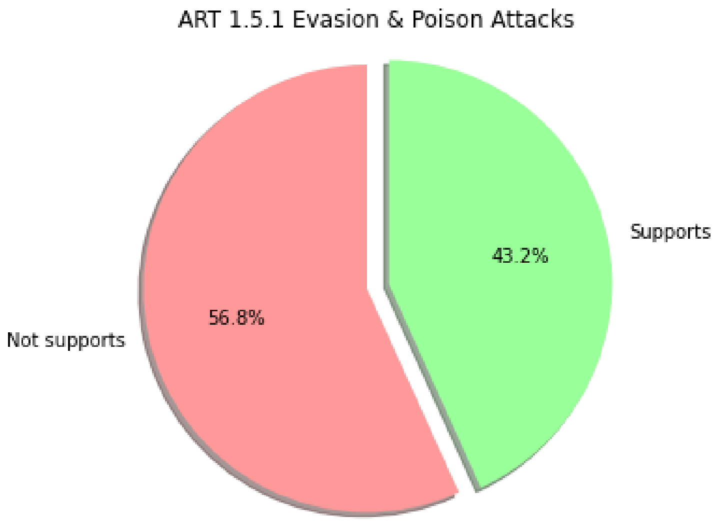


Figure 3.1: ART evasion and poison attacks supports scikit-learn or LightGBM ML frameworks.

Adversarial attack methods implemented in ART are considered to be ML framework independent and work with a number of ML/DNN models. But as shown in Fig. 3.1 more than half of the attacks unable to poisoned or create adversarial examples from input data on ML algorithms: SVM, DT, RF trained with scikit-learn; GBDT trained with LightGBM ML framework (Appendix I: Table 5.1 contains list of ART attack algorithms excluded from evaluation). Since, some attack algorithms are designed and implemented to support only a specific ML/DNN model, library, algorithm and application. For example, ART ShapeShifter attacks against deep learning models to object detection application. ART Decision Tree attack supports only scikit-learn decision tree classifier, But ART evasion attacks which are listed in Table 3.1, except Decision Tree attack creates adversarial example to fool the target SVM model. With these considerations, Our work focuses on ART attack methods against supervised ML algorithms are: SVM, DT, RF trained with scikit-learn and GBDT trained with lightGBM on image and tabular data for classification tasks.

This chapter is organized as follows. we review ART evasion attack algorithms in Section 3.1 and ART poisoning attack algorithm in Section 3.2. In evasion attack section, we review 15 ART evasion attacks, for instance, 14/15 of these evasion attacks against SVM model (as shown in Table 3.1). and in the data poison attacks section, we review only 1 ART poisoning attack on SVM.

3.1 ART Evasion Attacks

In ART evasion attacks, we generate adversarial examples in the targeted and untargeted settings. Note that the following two approaches will be used to perform framework-agnostic evasion attacks against ML models in ART are:

Approach #1: using pretrained models

1. Load a dataset (if needed do preprocessing).
2. Load pretrained model.
3. Create the ART classifier and wrap a model.
4. Generate adversarial examples using ART evasion attack algorithm.
5. To evaluate the classifier, predictions are made on the adversarial examples.

Approach #2: using untrained models

1. Load a dataset (if needed do preprocessing)
2. Create/build a model
3. Create ART classifier and fit a model.
4. Generate adversarial examples using ART evasion attack algorithm.
5. To evaluate the classifier, predictions are made on the adversarial examples.

Table 3.1: ART evasion attacks sorted by submission year (recent on the top).

ART Evasion Attacks					
No.	**Attack Algorithms**	**Attack Method**	**Attack Type**	**Objective**	**Supports on scikit-learn / LightGBM**
1	Targeted Universal Adversarial Perturbations (UAPs, Hirano and Takemoto, 2019), Section 3.1.1	White-box	Gradient-based	Targeted	SVM
2	HopSkipJump Attack (HSJA, Chen and Jordan, 2019), Section 3.1.3	Black-box	Decision-based	Both	SVM DT RF GBDT
3	Boundary Attack (BA, Brendel et al., 2018), Section 3.1.3	Black-box	Decision-based	Both	SVM DT RF GBDT
4	NewtonFool (Jang et al., 2017), Section 3.1.1	White-box	Gradient-based	Untargeted	SVM
5	Zeroth-order optimization Attack (ZOO, Chen et al., 2017), Section 3.1.2	Black-box	Score-based	Both	SVM DT RF GBDT
6	Elastic-Net Attack (EAD, Chen et al., 2017), Section 3.1.1	White-box	Gradient-based	Both	SVM
7	Projected Gradient Descent (PGD, Madry et al., 2017), Section 3.1.1	White-box	Gradient-based	Both	SVM
8	Carlini & Wagner Attack (C&W,Carlini and Wagner, 2017), Section 3.1.1	White-box	Gradient-based	Both	SVM
9	Universal Perturbation (UP, Moosavi-Dezfooli et al., 2016), Section 3.1.1	White-box	Gradient-based	Untargeted	SVM DT RF
10	Basic Iterative Method (BIM, Kurakin et al., 2016)	White-box	Gradient-based	Both	SVM
11	DecisionTree Attack (Papernot et al., 2016), Section 3.1.4	-	Decision tree-based	Untargeted	DT
12	Jacobian Saliency Map Attack (JSMA, Papernot et al., 2016) , Section 3.1.1	White-box	Gradient-based	Untargeted	SVM

13	DeepFool (Moosavi-Dezfooli et al., 2015), Section 3.1.1	White-box	Gradient-based	Untargeted	SVM
14	Virtual Adversarial Method (VAT, Miyato et al., 2015), Section 3.1.1	-	Gradient-based	Untargeted	SVM DT RF GBDT
15	Fast Gradient Sign Method (FGSM, Goodfellow et al., 2014) , Section 3.1.1	White-box	Gradient-based	Both	SVM

3.1.1 Gradient-based attacks

In this section, we review the gradient-based (belonging to white-box) attack algorithms to produce adversarial examples.

Fast Gradient Method (FGM)

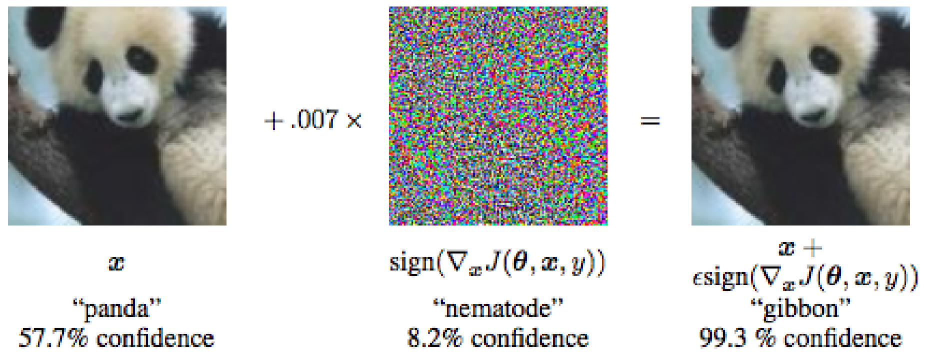
The Fast Gradient Method (FGM) (Nicolae et al., 2018) is a gradient-based attack, which designed to find a perturbation for a given original example, such that the classifier might forced to assigns the example to the wrong class in the targeted and untargeted settings (see 2.2, Annex 5). This attack works by computing the gradient of the loss with respect to the input once, and creating a small perturbation by multiplying a small adversarial budget (denoted as ϵ) by the vector of the gradients, i.e. a fast attack method. This attack has two versions based on distance metric used to create an adversarial example:

1. Fast Gradient Method attack, which uses ℓ_∞ norm to create adversarial example is known as "Fast Gradient Sign Method (FGSM)", which is originally implemented by Goodfellow et al. (2014). Figure 3.3 shows the images generated using ART FGSM attack, in which an adversarial image were created by changing to relatively gray color of some background (black) and foreground (white) pixel of the original image.

   In FGSM attack, a small perturbation (denoted as δ) is generated by computing the gradient of the classifier loss function w.r.t the input $\mathbf{x}$ is given as:

   $$\delta = -\epsilon \cdot sign(\nabla_{\mathbf{x}} \boldsymbol{L}(\mathbf{x}, y)) \tag{3.1}$$

   where $\epsilon > 0$ is the size of the adversarial perturbation (adversarial budget) with in ℓ_∞ norm, $\mathbf{x}$ is the input data and normalized into [0,1], y is either the class labels associated with $\mathbf{x}$ or specified by the attacker in untargeted and targeted settings, respectively. The gradient of the loss function of the targeted classifier is computed with respect to input $\mathbf{x}$ (Backpropagation algorithm used for DNN) and the sign of the gradient indicates the perturbation direction. the value of attack input variation ϵ controls the perturbation strength. Increasing ϵ value will increase misclassification, but the adversarial image is very different from the original image. Instead using small ϵ for which $\mathbf{x} + \delta$ is remains adversarial and maintains the similarity



+ .007 ×
=
x
"panda"
57.7% confidence
$\text{sign}(\nabla_x J(\theta, x, y))$
"nematode"
8.2% confidence
$x + \epsilon\text{sign}(\nabla_x J(\theta, x, y))$
"gibbon"
99.3 % confidence

Figure 3.2: Generating perturbation and adversarial example by FGSM (From Goodfellow et al., 2014)

between the original and adversarial example (see Figure 3.2 for illustration of FGSM attack on MNIST).

2. Fast Gradient Method (FGM) is an attack implementation based on ℓ_1 and ℓ_2 norm to create an adversarial example. In contrast to FGSM, the perturbation δ is computed as:

$$\delta = \epsilon \cdot \frac{\nabla_{\mathbf{x}} \boldsymbol{L}(\mathbf{x}, y)}{\|\nabla_{\mathbf{x}} \boldsymbol{L}(\mathbf{x}, y)\|_p} \tag{3.2}$$

where p=1 and 2 for ℓ_1,ℓ_2 norm, respectively. In ℓ_2 norm, we create the adversarial examples by adding the perturbation value computed in Equation (3.2) in the direction of the gradient.

In ART, the adversarial examples for all attack methods are constructed using clip function, which brings the value of adversarial example within allocated range during constructing of the adversarial examples, i.e., in the range of [0,1]. In FGM/FGSM attack, the adversarial example **x**' is created using Equation (3.3).

$$\mathbf{x}' = clip(\mathbf{x} + \sigma, \mathbf{x}_{min}, \mathbf{x}_{max}) \tag{3.3}$$

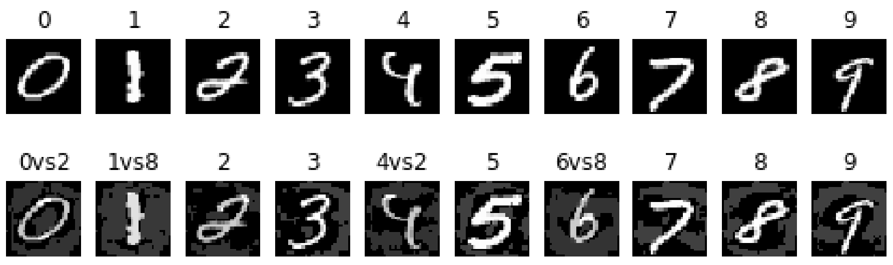


Figure 3.3: The top-side of the figure shows image of original MNIST and the bottom-side shows the perturbed images generated by ART FGSM under ℓ_∞ norm bounded by $\epsilon = 0.3$ and predicted class labels by SVM model.

In ART FGM/FGSM attack, the minimal perturbation parameter is used to attack in constraint of maximum input variation (denoted as ϵ_{max}) and step size (denoted as ϵ_{step}). When minimal perturbation applied, the attack performed iteratively with given step size until the attack succeed or failed (if $\epsilon_{step} > \epsilon_{max}$). This computing with minimal perturbation is shown in Figure 4.9.

Basic Iterative Method (BIM)

The Basic Iterative Method (BIM) (Nicolae et al., 2018) is an extension of FGSM attack in the targeted and untargeted settings (see 2.4, Annex 5), in which an adversarial example is created based on iterative procedure to modify the input within the ℓ_∞ norm bound of adversarial budget ϵ via chosen step size ϵ_{step} at each iteration, where $0 < \epsilon_{step} < \epsilon$.

Projected Gradient Descent (PGD)

The Projected Gradient Descent (PGD) (Nicolae et al., 2018) is an extension of FGSM attack, which modifies the input multiple times iteratively in the targeted and untargeted settings. In contrast with BIM attack, PGD perturbs the input example using ℓ_1, ℓ_2 and ℓ_∞ norm. Figure 3.4 shows the images generated using ART FGSM, BIM, PGD attacks.

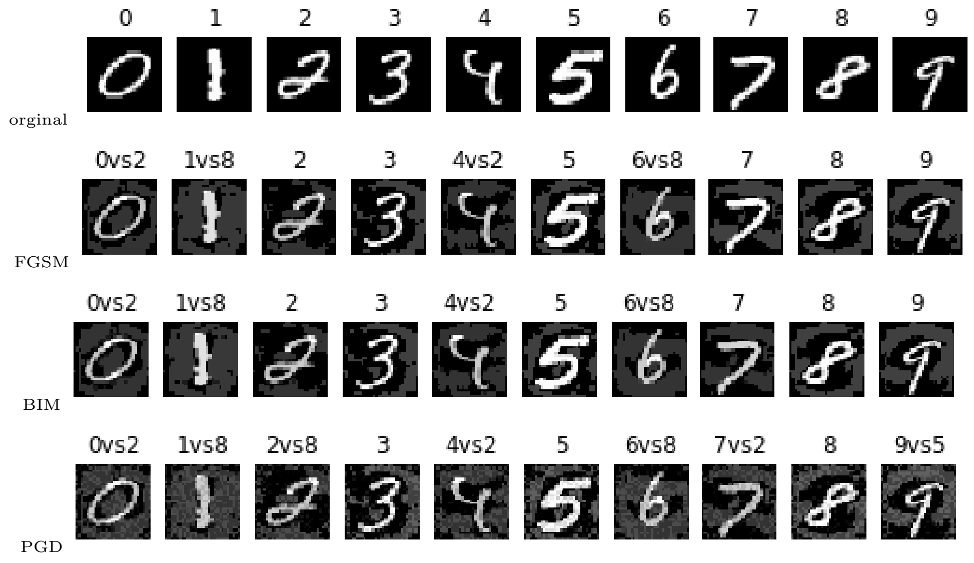


Figure 3.4: Image of original and perturbed images generated by ART FGSM, BIM, and PGD attacks under ℓ_∞ norm bounded by $\epsilon = 0.3$ on MNIST and predicted class labels by SVM model.

Carlini & Wagner (C&W) Attack

The Carlini & Wagner (C&W) ℓ_p attack (Nicolae et al., 2018; Carlini and Wagner, 2017) is a gradient-based attack aims at finding adversarial example using minimum ℓ_p norm of adversarial perturbations in the targeted and untargeted settings. The targeted version of the C&W attack solves the optimization problem in Equation (3.4) to find the smallest c for which the $\ell(\mathbf{x}')$ minimizing the objective function such that $\ell(\mathbf{x}') = 0$. In the equation, $\|\mathbf{x}' - \mathbf{x}\|_p$ is the distance between the original example $\mathbf{x}$ and adversarial example $\mathbf{x}$', $\ell(\mathbf{x}')$ is the approximation function, y is the target label, $k \geq 0$ is a confidence parameter, c is a size of perturbation. Therefore, we have the constraint $\|\mathbf{x}' - \mathbf{x}\|_p \leq \epsilon$ for a given $\epsilon > 0$.

$$\begin{aligned} &\mathit{minimize} \quad \|\mathbf{x}' - \mathbf{x}\|_p + c \cdot \ell(\mathbf{x}') \\ &\mathit{Such\ that} \quad \|\mathbf{x}' - \mathbf{x}\|_p \leq \epsilon \\ \mathit{where} \quad \ell(\mathbf{x}') = \mathit{max}(&\mathit{max}\ \{Z_i(\mathbf{x}') : i \in \mathcal{Y} \setminus \{y\}\} - Z_y(\mathbf{x}') + k, 0). \end{aligned} \tag{3.4}$$

Where Z is the logit for input $\mathbf{x}$', in the targeted setting $\ell(\mathbf{x}')$=0 if and only if the clasifier C($\mathbf{x}$')=y.
In contrast, the untargeted version of the C&W attack solves the objective function in Equation (3.5).

$$\ell(\mathbf{x}') = \mathit{max}(Z_y(\mathbf{x}') - \mathit{max}\{Z_i(\mathbf{x}') : i \in \mathcal{Y} \setminus \{y\}\} + k, 0) \tag{3.5}$$

In the ART implementation, the distance metric are ℓ_2 and ℓ_∞ norm. and, binary search method is used to find c. Figure 3.5 shows the images generated using ART C&W ℓ_∞ attack (more in 2.7, Annex 5).

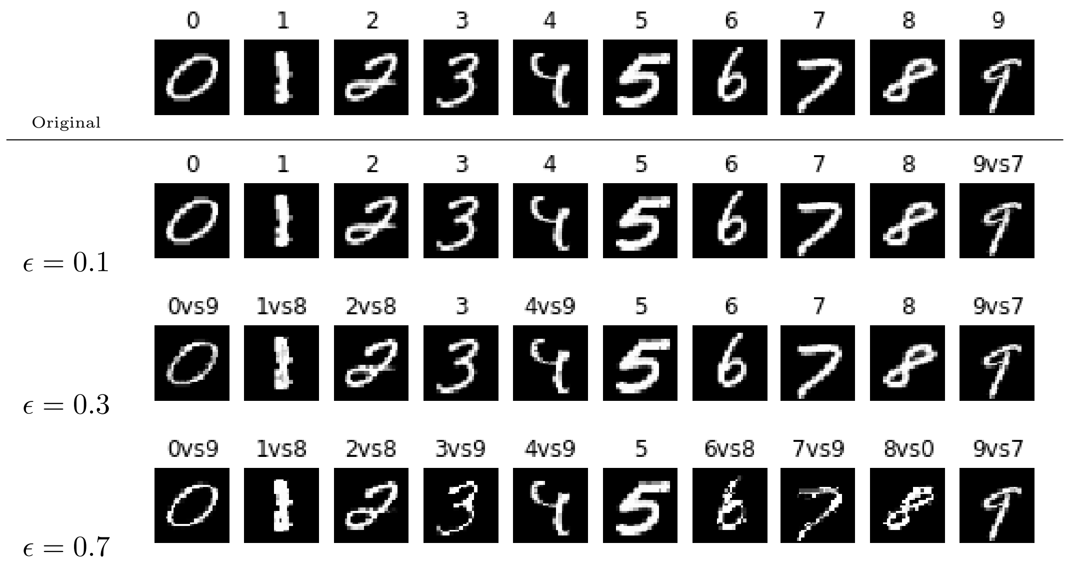


Figure 3.5: Image of original and perturbed images generated by ART C&W(ℓ_∞) attack on MNIST. As a result, the predicted class labels by SVM model.

Elastic-Net (EAD) Attack

The Elastic-Net (EAD) attack (Nicolae et al., 2018; Chen et al., 2017) is a gradient-based attack aims to minimize ℓ_1 norm of adversarial perturbations in the targeted and untargeted settings. Figure 3.6 shows the images generated using ART EAD attack. In addition, EAD uses elastic-net regularization method (see 2.8, Annex 5).
Chen et al., (2017) proposed variant of C&W attack with ℓ_1 norm of adversarial perturbations. ℓ_1 norm is a convex function which measures the number of modified pixels (when the input is image), i.e., sparsity by the perturbation and small number of perturbation is enough for attack.

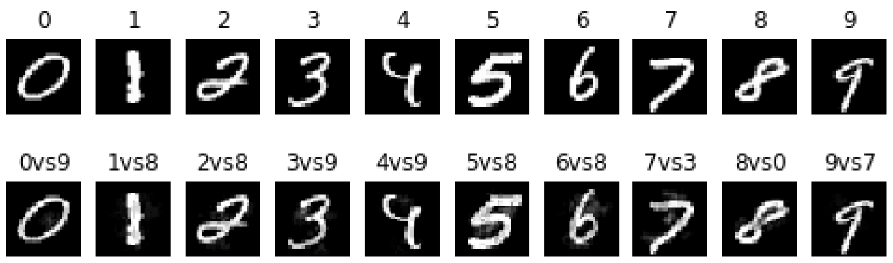


Figure 3.6: The top-side of the figure shows image of original MNIST and the bottom-side shows the perturbed images generated by ART Elastic-Net EAD(EN) attack with 100 iterations. As a result, the predicted class labels by SVM model.

Universal Perturbation (UP)

The Universal perturbations (UP) (Nicolae et al., 2018) aims at finding a single adversarial perturbation ρ, i.e., used to generate input agnostic perturbations using the following algorithms: FGSM, C&W, DeepFool, BIM, PGD, EAD, NewtonFool, JSMA, and VAT in the untargeted setting. Figure 3.7 shows the images generated using ART FGSM and UP attacks.
Moosavi-Dezfooli et al., (2016) proposed a universal perturbation computing algorithm and demonstrate a single perturbation, which is image-agnostic can causes high probability classification errors on a natural images using DNN model.

The Universal Adversarial Perturbation implementation in (Nicolae et al., 2018) is an iterative procedure to accumulate the universal perturbation on random inputs, and its refined universal perturbation is projected into the ℓ_p norm with the maximum bound ϵ for attack strength. When iteration ends and the attack failure tolerance satisfies, the final universal perturbation ρ is added to examples to create adversarial examples (see Equation (3.6)).

$$\mathbf{x}\text{'} = \mathbf{x} + \rho \tag{3.6}$$

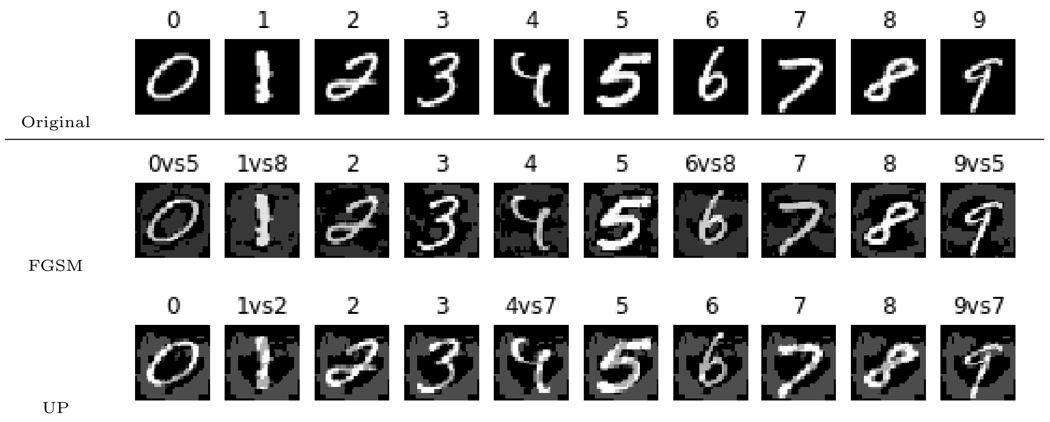


Figure 3.7: Image of original and perturbed images generated by ART FGSM and UP attack with ℓ_∞ norm bounded by $\epsilon = 0.3$ on MNIST. As a result, the predicted class labels by SVM model.

Targeted Universal Adversarial Perturbations (UAPs)

The Targeted Universal Adversarial Perturbations (UAPs) (Nicolae et al., 2018) is similar with UP attack, but the main difference is UAPs attack works in the targeted setting (Hirano and Takemoto, 2019), and with two algorithm: Fast Gradient Sign Method (FGSM) and Simple Black-box Adversarial (SimBA) algorithms. Figure 3.8 shows the images generated using ART FGSM and UP attacks.

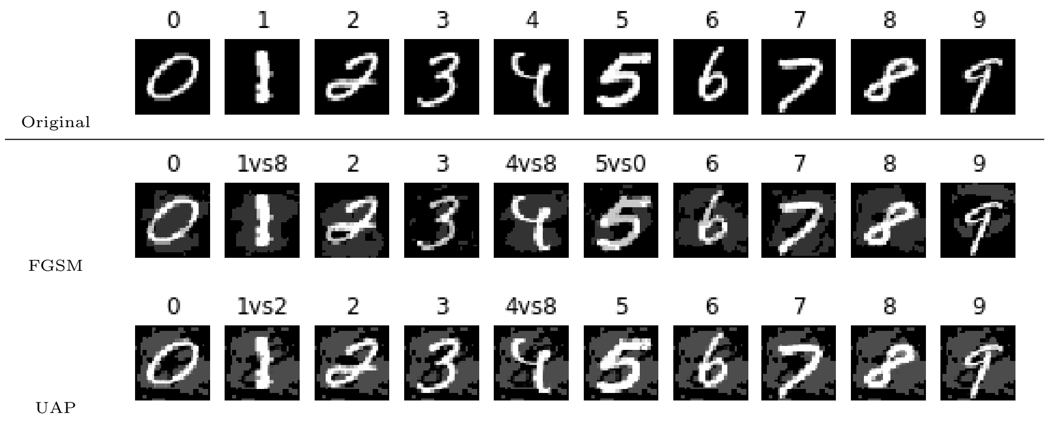


Figure 3.8: Image of original and perturbed images generated by ART FGSM and UAP attacks with ℓ_∞ norm bounded by $\epsilon = 0.3$ and prediction of the classifier on MNIST. As a result, the predicted class labels by SVM model.

Jacobian Saliency Map Attack (JSMA)

The Jacobian-based Saliency Map Attack (JSMA) (Nicolae et al., 2018) is a gradient-based attack, which aims at perturbing a small set of input features i.e. controlling by ℓ_0 norm (sparse perturbations), instead of the whole input features, and based on the saliency map to create an adversarial examples in the untargeted setting Papernot et al., (2016). Figure 3.9 shows the images generated using ART JSMA attack.

The JSMA method implementation in (Nicolae et al., 2018) is an iterative procedure, in which for a given input example with target label, the algorithm computes a saliency map using saliency_map function, then based on the saliency map, the algorithm iteratively chose a small set of input features to change at each step to construct an adversarial example that will increase the likelihood of the target label (see 2.4, Annex 5).

DeepFool

The DeepFool (Nicolae et al., 2018) is a gradient-based attack, its goal is to find a minimum adversarial perturbation direction that push the original example $\mathbf{x}$ over the separating hyperplane of the classifier C($\mathbf{x}$) with ℓ_2 norm to assigns the example to the different class in the untargeted setting. Note that, the perturbation applied iteratively in case of nonlinear problems. Figure 3.10 shows the images generated using ART DeepFool attack.

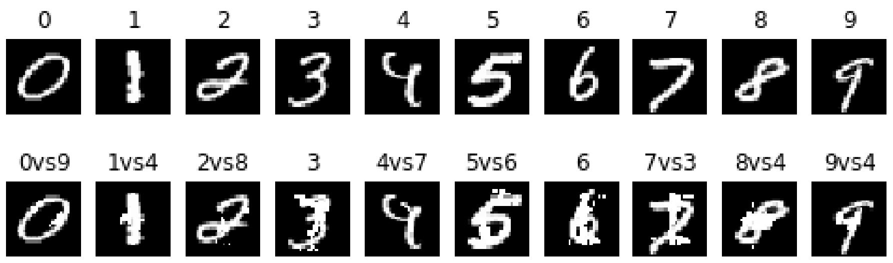


Figure 3.9: The top-side of the figure shows image of original MNIST and the bottom-side shows the perturbed images generated by ART JSMA(ℓ_0) attack and predicted class labels by SVM model.

The DeepFool attack implementation in (Nicolae et al., 2018) is based on an iterative procedure to find the nearest decision boundary in ℓ_2 distance metric and push the original example over the decision boundaries until a different target label obtained or maximum iteration reached, and the results of adversarial perturbation δ is computed using multiplied by a factor 1 plus overshoot parameter (termed epsilon $\epsilon \geq 0$) in Equation (3.7) to push the example over the decision boundary, then adversarial example constructed as in Equation (3.8):

$$\delta = (1 + \epsilon) \cdot (\mathbf{x}\text{'} - \mathbf{x}) \tag{3.7}$$

$$\mathbf{x}\text{'} = clip(\mathbf{x} + \delta, \mathbf{x}_{min}, \mathbf{x}_{max}) \tag{3.8}$$

Moosavi-Dezfooli et al., (2015) shows that the computed perturbation using this algorithm is faster than FGSM (see 2.5, Annex 5).

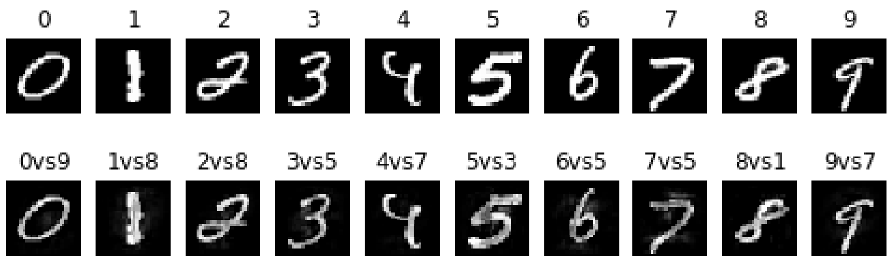


Figure 3.10: The top-side of the figure shows image of original MNIST and the bottom-side shows the perturbed images generated by ART DeepFool(ℓ_2) attack with $\epsilon = 1e-6$ and 100 iterations. As a result, the predicted class labels by SVM model.

NewtonFool

The NewtonFool (Nicolae et al., 2018) is a gradient-based attack, in which the attack aims to decrease class probabilities (softmax output for Neural Networks) to zero by applying gradient descent in the untargeted setting (Jang et al., 2017). Figure 3.11 shows the images generated using ART NewtonFool attack.

The NewtonFool attack implementation in (Nicolae et al., 2018; Jang et al., 2017) is based on iterative procedure to find small perturbation d such that the probability $F_y(x+d) \approx 0$. In each iteration, compute using Newton's method for solving the step size (denoted as δ), which changes over time based on step 6 in NewtonFool attack algorithm and tuning parameter $\eta \in (0, 1)$ to control how small the perturbation does, the perturbation d is computed as based on Equation (3.9), and results of adversarial example constructed (see 2.6, Annex 5) using Equation (3.10).

$$d = -\frac{\delta \cdot \nabla F_y(\mathbf{x}')}{\|\nabla F_y(\mathbf{x}')\|_2} \tag{3.9}$$

$$\mathbf{x}' = clip(\mathbf{x} + d, \mathbf{x}_{min}, \mathbf{x}_{max}) \tag{3.10}$$

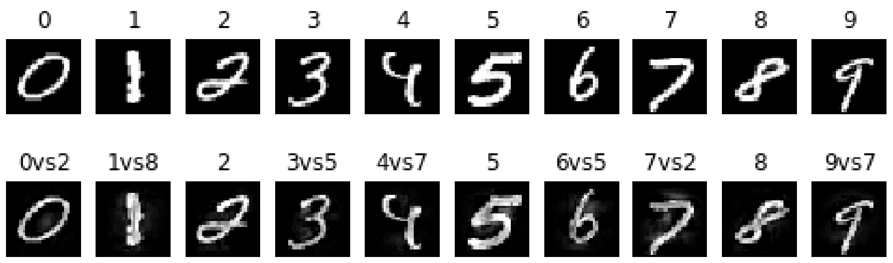


Figure 3.11: The top-side of the figure shows image of original MNIST and the bottom-side shows the perturbed images generated by ART Newton(ℓ_2) attack with $\eta = 0.1$. As a result, the predicted class labels by SVM model.

Virtual Adversarial Method (VAT)

The Virtual Adversarial Method (VAT) (Nicolae et al., 2018) is a gradient-based attack aims at finding ℓ_2 norm bounded adversarial perturbation by maximizing the Kullback-Leibler (KL) divergence between output distribution in the untargeted setting. Note that, VAT uses a local distributional smoothness (LDS) technique for the output distribution of the classifier in the adversarial training methods (Miyato et al., 2015).

The Virtual Adversarial Method with finite differences implementation in (Nicolae et al., 2018) is based on iterative approach to construct a perturbation d under minimum ℓ_2 norm and computes KL divergence.

3.1.2 Score-based attacks

In this section, we review Zeroth-order optimization (ZOO) attack only, i.e., a score-based attack algorithm to produce adversarial examples compatible with traditional ML algorithms.

Zeroth-order optimization (ZOO) Attack

The Zeroth-order optimization (ZOO) Attack (Nicolae et al., 2018) is a score-based (black-box) version of C&W(ℓ_2) attack in the targeted and untargeted settings (see 4.1-4.4, in Annex 5). Chen et al., (2017) proposed a black-box (query-based) attack, aims to estimate the gradients of the objective function with respect to the input $\mathbf{x}$ using stochastic coordinate descent, i.e., in each iteration computes the gradient of a batch of input coordinate or dimension instead of the whole input features. In the ART ZOO attack implementation, the optimization algorithm is based on ADAM coordinate descent method. The illustration in Figure 3.12 shows that an adversarial images produced by ART ZOO attack on MNIST.

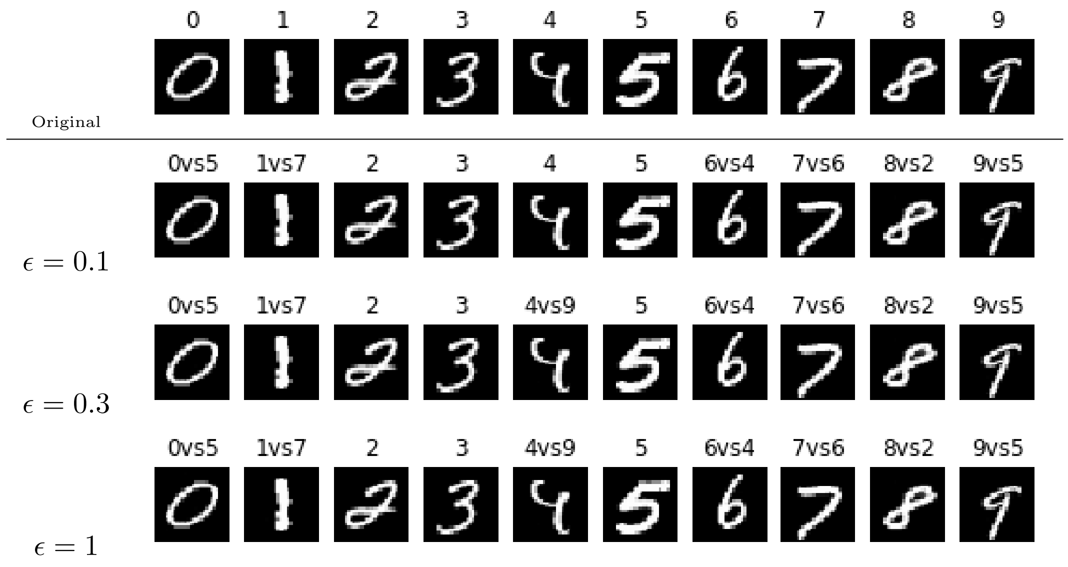


Figure 3.12: Image of original and perturbed images generated by ART ZOO(ℓ_2) attack on MNIST. As a result, the predicted class labels by DT model results 60%, 70%, and 70% fooling rate for step size 0.1, 0.3 and 1, respectively.

3.1.3 Decision-based attacks

In this section, we reviews two decision-based attack algorithms to produce adversarial examples compatible with traditional ML algorithms.

Boundary Attack (BA)

The Boundary Attack (BA) (Nicolae et al., 2018) is the first successful decision-based attack, and works in the targeted setting. Figure 3.13 shows the images generated using ART Boundary attack on DT, RF, GBDT, and SVM models on MNIST.

Brendel et al. (2018) proposed a heuristic algorithm, in which the algorithm first initialize a sample that is already adversarial, and then perturbs this sample along the decision boundary, until the perturbed input minimizes the ℓ_2 norm difference with respect to the original input is as follow:

- **Proposal distribution:** sample from normal distribution $\mathcal{N}(0, 1)$ and then rescale and clip the sample.
- **Projection:** project onto a sphere around the original input.

- **Decision boundary:** finally make a small movement towards the original input.

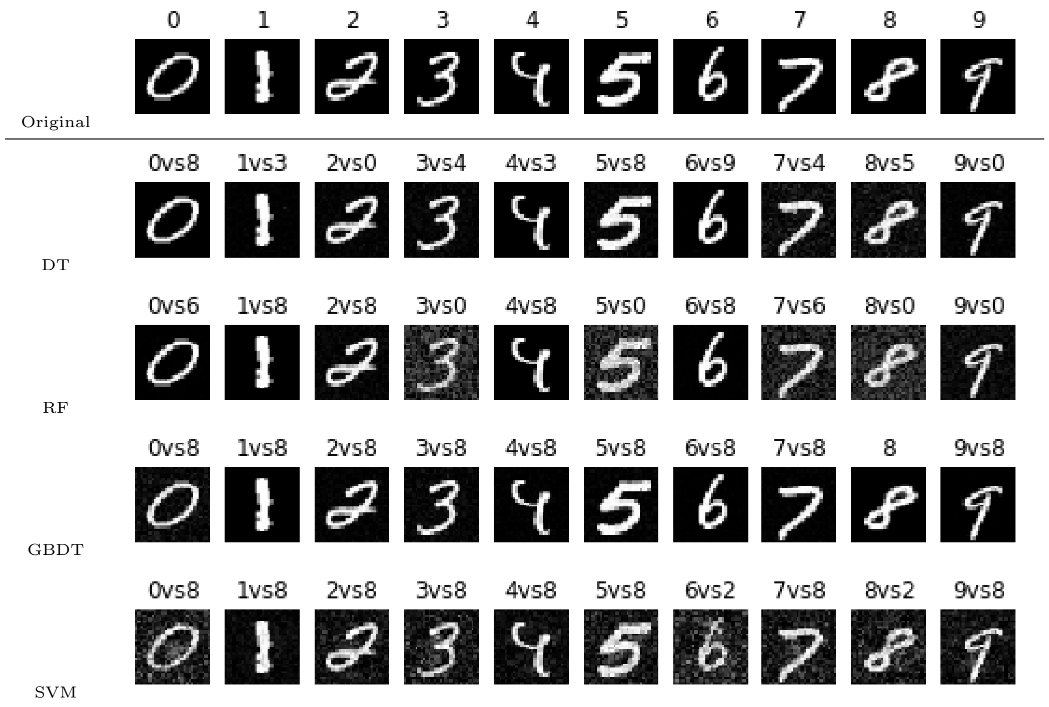


Figure 3.13: Image of original and perturbed images generated by BA(ℓ_2) against DT, RF, GBDT, and SVM models on MNIST with 100 iterations and $\epsilon = 0.01$. As a result, the predicted class labels for adversarial images by DT, RF, GBDT, and SVM models.

HopSkipJump Attack (HSJA)

The HopSkipJump Attack (HSJA) (Nicolae et al., 2018) is an algorithm that uses decision boundary to create an adversarial example in the targeted and untargeted settings. Figure 3.14 shows the images generated using ART HopSkipJump attack on DT, RF, GBDT, and SVM models on MNIST.

Chen and Jordan (2019) proposed HopSkipJump attack, which is an improved version of the Boundary attack, i.e., fewer model queries used to craft adversarial examples using ℓ_2 and ℓ_∞ norms. The algorithm introduced by the author's is an iterative algorithm and in each iteration the following functions are executed:

1. **Boundary search:** performs binary search from the last iteration if exists to approach the decision boundary. First, set upper and lower bounds as well as the threshold for the binary search, then start the binary search by updating the upper bound and lower bound iteratively, see Bin-Search algorithm in Chen and Jordan (2019).

2. **Gradient-direction estimation:** the gradient direction is estimated, since the gradient of the model with respect to the input is not available, see Eq.(16) in Chen and Jordan (2019).

3. **Step size search:** the updating step size along the gradient direction is initialized, see Eq.(13) in Chen and Jordan (2019).

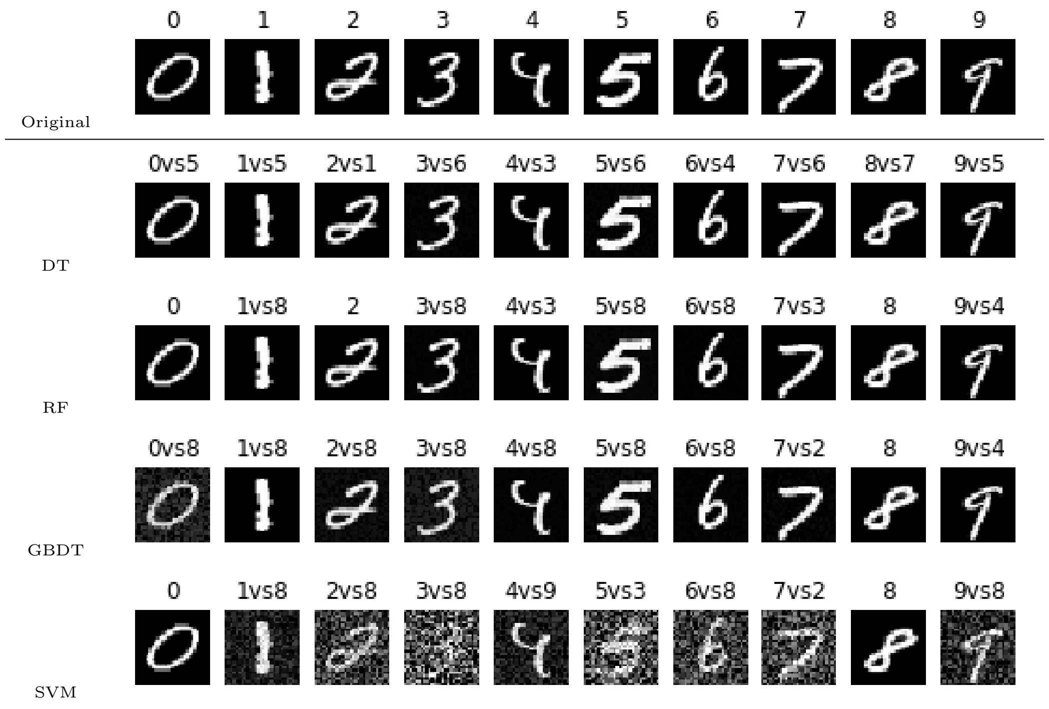


Figure 3.14: Image of original and perturbed images generated by HSJA(ℓ_∞) attack against DT, RF, GBDT, and SVM models on MNIST with 10 iterations and $\epsilon = 0.01$. As a result, the predicted class labels for adversarial images by DT, RF, GBDT, and SVM models.

3.1.4 Decision tree-based attacks

In this section, we see the decision tree based attack algorithm to produce adversarial examples.

Decision Tree attack

The Decision Tree attack (Nicolae et al., 2018) is an algorithm that uses decision tree structure to create an adversarial example in the untargeted setting (see Figure 3.16).

Decision Tree attack is a pioneer work on adversarial attack against DT model. In ART, this algorithm is implemented based on the crafting decision tree adversarial samples algorithm in Papernot et al., (2016).

Given a decision tree, original examples and its corresponding class labels as shown in Figure 3.15, the algorithm first finds the leaf in decision tree corresponding to example given and then search leaves with different classes in the neighborhood of the original leaf using simple tree search procedure, then the searched leaf used as an adversarial leaf. In the second step, the algorithm modifies the example accordingly to traverse to adversarial leaf. As a result the modified example (adversarial example) created, in which the targeted classifier outputs a wrong result.

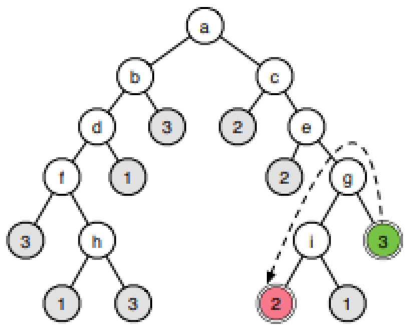


Figure 3.15: Generating adversarial examples with DecisionTree Attack (From Papernot et al., (2016))

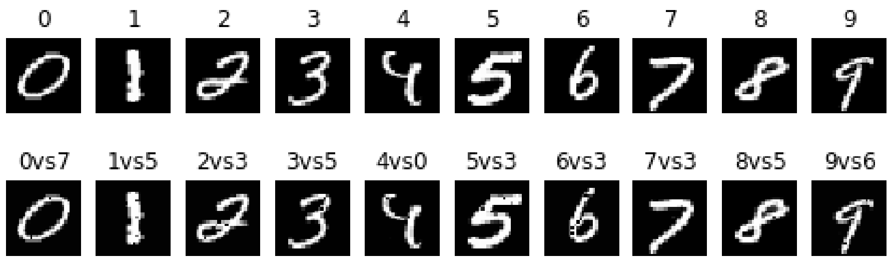


Figure 3.16: The top-side of the figure shows image of original MNIST and the bottom-side shows the perturbed images generated by ART Decision Tree attack and predicted class labels by DT model, i.e., 100% assigns the adversarial images to the wrong class but it looks very similar to the original image.

3.2 ART Poisoning Attacks

In case of adversarial poisoning attacks, adversaries feed malicious input that will be used to fit a model, then due to malicious training data are used to fit a model, this classifier test error will be high.

3.2.1 Adversarial Poisoning Attack on SVM

The Adversarial Poisoning attack on Support Vector Machines (SVM) is a white-box attack, in which adversaries introduce malicious input to fit the SVM model in the targeted and untargeted settings. The following pseudo-code describes the implementation of this attack in ART, which will accepts SVM model and malicious sample input, then it finds attack points (adversarial examples) iteratively starting from given malicious sample input, which are initialized as attack points (at least one malicious input required), then compute per-class derivatives with respect to the attack point, which the attack class will be the opposite of the model's classification for the attack point, results a poisonous model and adversarial examples returned. The malicious training data (adversarial and original examples) are used to fit the model, in which the decision boundaries of the model might wrongly classify new unseen test data or the model performance decreases. The implementation of this attack algorithm in ART is based on Biggio et al., (2012) and supports only LinearSVC and SVC model types provided by scikit-learn ML framework[2] .

Based on the ART implementation, we can review two specific functions from the Poisoning attack on SVM module functions. First create a wrapper class, which is used to train scikit-learn implementation of the SVM algorithm. Then initialize the data poison attack module "PoisoningAttackSVM", which has trained model, training data and attack parameter settings. The training data must be cleaned to remove duplicate examples and dividing it into training/test sets, and target labels for the attack be specified. On the first function poison, which works by accepting input as an initial attack points, then the algorithm iteratively finds the optimal attack points (i.e. poisoned examples and poisoned labels), then list of poisoned examples and labels are returned. Note that, providing parameters such as maximum number of iterations will increase data poisoning attack on original input data. The function attack_gradient, which implements the algorithm based on Eq. (8) in Biggio et al., (2012) calculates and returns list of attack gradients using the specified tolerance level.

Chapter 4

Experiments and Analysis

4.1 Experimental setup

Our experimental setup is based on the current version of Adversarial Robustness Toolbox (version 1.5.1), Scikit-learn (version 0.23.1) and LightGBM (version 3.1.1) with Jupyter Lab and Jupyter Notebooks, to perform the threat of evasion and poison attacks against supervised machine learning models: support vector machines (SVM), decision trees (DT), random forest (RF), gradient-boosting decision trees (GBDT); applied to census income tabular data and MNIST handwritten digit image classification dataset.

We compare ART Attacks (i.e., data poisoning and evasion attacks) against the baseline models, in which the models doesn't implement any adversarial defense techniques at training and test time, respectively.

We identified 15 evasion attacks and 1 poisoning attack implemented in ART; and in the following experiment, we will analyze and evaluate the performance of our chosen classifiers under ART attacks in targeted and untargeted adversarial settings, and evaluate their performance for tabular and image data. All the experiments are done on a system with Intel core i3 2.53GHz CPUs (4-core) and 4GB RAM.

4.2 Dataset

ART supports all kinds of data types such as tabular, audio, video, images and others. Among a number of datasets available, we used the following two datasets for our experimental analysis on tabular and image data types are as follow.

First, for a tabular data type we used the Census Income dataset from the UCI Machine Learning Repository[3], which refers from now as census. The census used in our work contains 48,842 entries, by merging of train data[4] and test data[5]. It has the following 15 columns entries: income group, age, work class, final weight, education level, education number, marital status, occupation, relationship, race,

[3]https://archive.ics.uci.edu/ml/datasets/Census+Income

[4]https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data

[5]https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.test

sex, capital gain, capital loss, working hours per week, and native-country. The class distribution in census: 76.1% entries labeled with ≤50k (or negative class) and 23.9% entries labeled with >50k (or positive class), which means there is skewness in the distribution. Hence, census class distribution are slightly skewed, with around 3/4 of the entries labeled with negative class.

Second, for an image data type we used the handwritten digits from the MNIST database[6], which will be referred to as MNIST, which contains grayscale images with size of 28 × 28 pixels of 10 digit images (from digit 0 to 9). This dataset contains 70,000 digit images and labels values from 0 to 9, by merging the training set and test set examples[7] (see Figure 4.1). Each digit image has the size of 28 X 28 pixels, of each pixel value is in the range of 0 and 255 of pixel intensity where 0 represents white color (or background) and 255 represents black color (or foreground).

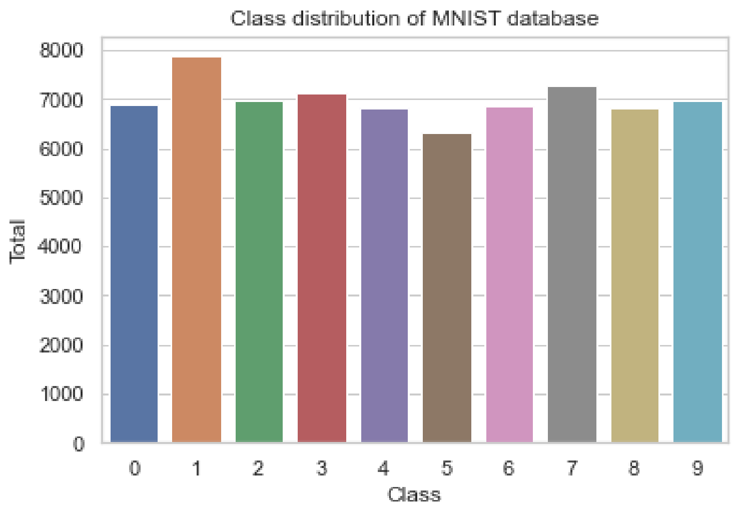


Figure 4.1: Class label (or digits) distribution of mnist data.

4.3 Training and Evaluation

Following that, we train and evaluate our chosen baseline models, i.e. SVM, DT, RF, and GBDT on the training data. With assumption to that, data preparation is recommended before training our baseline models; this process is mandatory to improve the performance of the model on test data, if the data is not processed properly, then an errors may be raised during model training.

4.3.1 Data Preparation

We used data preprocessing techniques on both datasets. The data preprocessing will be conducted with the below steps:

[6]http://yann.lecun.com/exdb/mnist/
[7]https://s3.amazonaws.com/img-datasets/mnist.npz

- First we need to handle missing values. In fact, missing values are issues in real world datasets. For instance, the census has missing values in some categorical features. If missing values were not handled either by removing or imputing, it will reduce the performance of the model or error will be raised by many machine learning algorithms. To resolve missing data issues in census, we used imputation methods for handling missing values that will be replaced by the median and mode value for each numerical and categorical columns, respectively (AMI, 2016).

- Then encoding categorical features, as machine learning algorithms like SVM assumes numeric inputs then all categorical features in census data needs to transform to numeric values. In census data, the following feature such as work class, education level, marital status, occupation, relationship, race, sex, and native-country are categorical data. To transform categorical data to numerical data, we used get_dummies from pandas library[8] for one-hot encoding, which binarize the categorical (discrete) features. To compute gradient loss in ART, class labels must be transformed to one-hot encoded labels, i.e., y is encoded as the y-th standard basis vector e_y (Nicolae et al., 2018).

- After handling categorical data, the next task is data scaling, we scaled numerical features in census and mnist data using normalization scaling method, which scales only the numeric features to [0,1] but categorical features are already in [0,1]. It was recommended to build machine learning model on scaled data. For example, SVM algorithm can efficiently compute the gradient in scaled data. In ART, the attack algorithms generates an adversarial example $\mathbf{x}'$ in the valid range of $[\mathbf{x}_{min}, \mathbf{x}_{max}]$, which mean in [0,1] when the given data is normalize (Nicolae et al., 2018).

- Last, we don't use all features in census to train our baseline models. For example, education number and level are basically the same, then we drop education number feature. The resulting set of features used in model training are: age, work class, final weight, education level, marital status, occupation, relationship, race, sex, capital gain, capital loss, working hours per week, and native country.

Considering to MNIST, This dataset contains 1 channel gray color handwritten digit images, each image is a size of 28×28 pixels, where each pixel value is in the range of 0 and 255. In our work we used the preprocessed data[9]. Then we applied two data preparation before model training and testing stages (1) data scaling, which normalizes values (or pixel intensity values) between 0 and 1. In this regard, ART can generates adversarial example x' in the valid data range of $[x_{min}, x_{max}]$, which is in the range of [0,1]; (2) each digit image with 28X28 (as shown in Figure 4.2 should be flattened to 784 features, where each feature value is in the range of 0 and 255. mnist is an example of high dimensional features dataset and in this work, we didn't apply any feature reduction methods.

[8] https://pandas.pydata.org/

[9] https://s3.amazonaws.com/img-datasets/mnist.npz

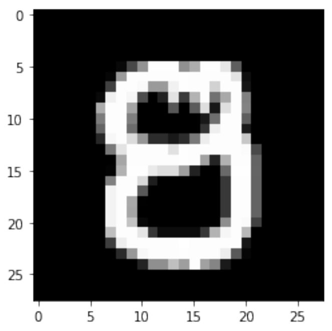


Figure 4.2: Plotting digit 'eight' from mnist

4.3.2 Model Training and Evaluation

On Tabular data type

In case of census, the problem is a binary classification task having two class labels: either ≤50k Income (class=0 or negative class) or >50k Income (class=1 or positive class). We split the census data into three parts while maintaining class distribution into 60%÷20%÷20% parts: In which ≈ 29k entries will be for training sets, and ≈ 9k entries will be for each validation and test sets. Validation sets are used for model evaluation by tuning hyperparameters while training a model. on other hand, training set and test set are used for model training and testing respectively.

Before data split, we shuffled the dataset using stratified k-fold cross validation[2] instead of random k-fold cross validation. This technique is useful to preserve the percentage of samples for each class under splitting the data in train/test sets.

We also use the same random seed to guarantee reproducibility on training and test sets, which helps to compare performance between different models having different hyperparameter settings.

Table 4.1: census training data

TargetLabel	*Entries*	*InPercentage*
≤50k	29,724	76.073%
>50k	9,349	23.927%

Table 4.2: census test data

TargetLabel	*Entries*	*InPercentage*
≤50k	7,431	76.067%
>50k	2,338	23.933%

Table 4.1 and 4.2 shows statistics information for census training and test data,

respectively. Both training and test data contains a class distribution of approximately 76% entries labeled with ≤50k (or negative class) and approximately 24% entries labeled with >50k (or positive class). This shows skewed distribution of 3/4 portions belongs to negative class on both training and test data.

For each baseline model, we used cross validation techniques with hyperparameter settings to optimize model performance which generalize the performance on test data. One of the useful advantages using hyperparameter tuning is to minimize prediction errors concerning bias-variance trade-off. For example, a regularization parameter such as the C-term in L_2 penalty in scikit-learn support vector machine are used as hyperparameter setting. Our machine learning model evaluation and selection is based on K-Fold cross validation techniques. In this method, training data is randomly separated into k folds, k-1 folds are used for training then one for testing, Then summarized into mean and standard deviation of all k different trained model performance scores.

The class distribution in census training data is skewed as shown in Table 4.1. The performance of the trained model will be poor on minor class as a result of the skewed class distribution. Hence, we use an oversampling technique called Synthetic Minority Oversampling Technique (SMOTE)[10] which is an over sampling method to overcome class imbalance problem. Then the new training set resampled to balanced class distribution (see Table 4.3). However, the test set remains unchanged.

Table 4.3: census training data: over-sampling using SMOTE

TargetLabel	*Entries*	*InPercentage*
≤50k	29,724	50%
>50k	29,724	50%

RandomizedSearchCV[2] is a scikit-learn API, which is a hyperparameter tuning method based on randomized search with cross validation. On model training, this method finds the best model by evaluating multiple models produced with randomly selected hyperparameter values on validation sets. This process is known as model selection. We fit a model using best hyperparameter values, then evaluate the model on unseen data to generalize the model performance. We use RandomizedSearchCV for hyperparameter tuning on training of SVM, DT and RF models. Whereas, this API is not compatible with GBDT trained with LightGBM. To use RandomizedSearchCV, we need to use LGBMClassifier[11], which is an extension of LightGBM that interface with scikit-learn API's. Whereas, the model trained using LGBMClassifier is not compatible with ART library. For this reason, we use an iterative procedure to search hyperparameter values. With this assumption, we fit our model using the best hyperparameter values and used as a baseline model in our ART attack experiments.

[10]https://imbalanced-learn.org
[11]https://github.com/microsoft/LightGBM

The following performance metrics are used to evaluate our classification models: precision, recall, micro F1 score, receiver-operating characteristic and area under the curve (ROC-AUC) and matthews correlation coefficient (MCC) on census test data, since the distribution of classes is not symmetrical; macro accuracy used for model performance on mnist test data.

On mnist data, the problem is multiclass classification task and having 10 class labels with relatively balanced dataset as shown in Figure 4.1, there is a balanced distribution of digits in the dataset, except 'digit 1' occurs more than other digits. Hence, macro accuracy (balanced accuracy) performance metric is useful evaluation metric.

Before considering a comparison between ART attacks for prediction performance of chosen classifiers on original and adversarial examples. Let's start with the summary and visualization of the performance of the baseline models on test data (original examples) and which are attack unaware models. Figure 4.3 shows confusion matrix results for each baseline classifiers: SVM, DT, RF and GBDT. We evaluate each baseline classifier on 9769 census test data.

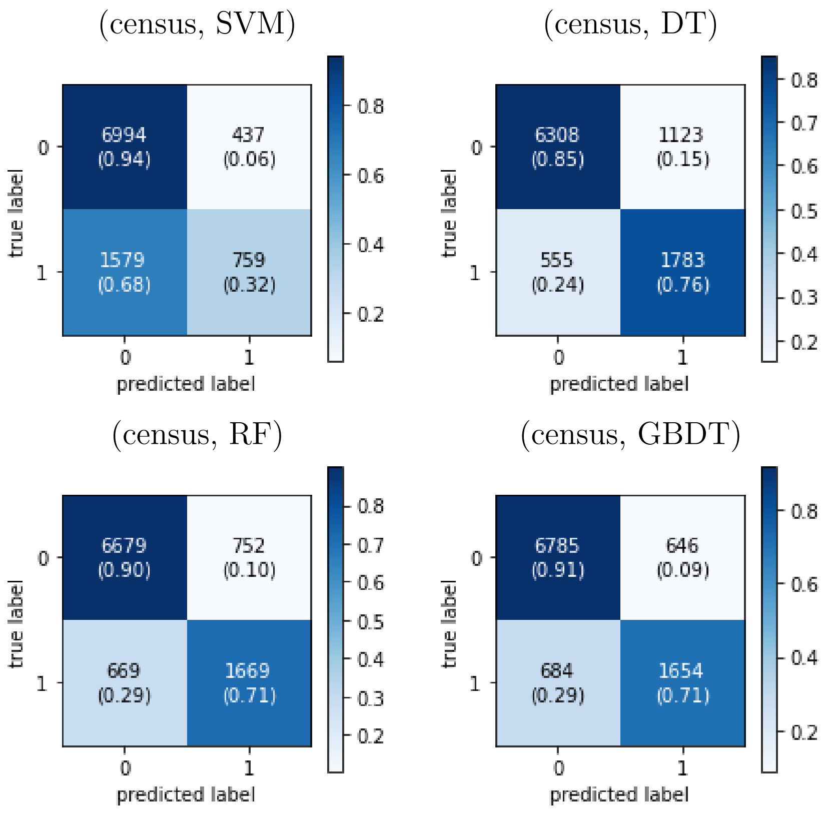


Figure 4.3: Visualization of confusion matrix result for census test sets

All evaluation metrics mentioned in this thesis are based on the confusion matrix (Fernández et al., 2018). In our model selection, ROC AUC metric is used to compare models, Figure 4.4 shows the ROC AUC performance of the best model on census test data. The hyper-parameter settings used to train best model configuration for the census and MNIST data are specified in Tables 4.5 and 4.6, respectively.

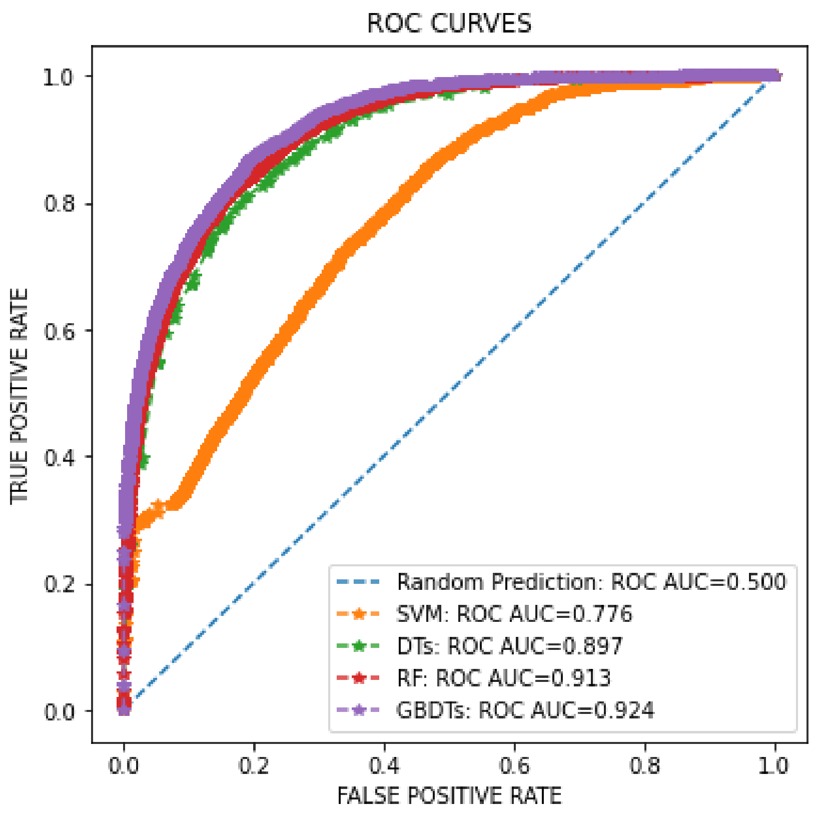


Figure 4.4: Visualization of ROC Curve on census

Table 4.4 shows baseline model performance results evaluated with $\approx 9k$ census test data: precision, recall, F1 score, ROC AUC and MCC (best results in boldface). GBDT has best performance compared to other models in precision, F1 score, ROC-AUC and MCC. whereas, DT has better result on recall compared to RF and GBDT.

Table 4.4: Experimental result on $\approx 9k$ census test data. (best results in boldface)

Model	Precision	Recall	F_1 score	ROC-AUC	MCC
SVM	0.635	0.325	0.794	0.776	0.346
DT	0.614	**0.763**	0.828	0.897	0.571
RF	0.689	0.714	0.855	0.913	0.605
GBDT	**0.719**	0.707	**0.864**	**0.924**	**0.624**

On Image data type

Our aim here is to correctly guess the digit value of a handwritten digit image on MNIST data, which indicates a multiclass classification problem. Now similar with model training on census data, we split the MNIST data into three parts while maintaining class distribution into 60%÷20%÷20% parts: In which 42k entries for

Table 4.5: Hyperparameter configuration used for model training on census

Classifier	Grid search intervals	Model hyperparameter

Scikit-learn SVC	C=stats.uniform(0.5, 10)
	gamma=stats.uniform(0.1, 1)
	kernel=['linear', 'rbf', 'poly']
		C=5.134701386869259
		gamma=0.8514309967653098
		kernel=rbf
		probability=True
		random_state=42

Scikit-learn DecisionTree	max_depth=randint(3, 50)
	max_leaf_nodes=randint(5, 1000),
	min_samples_split=randint(2, 100)
		max_depth=46
		max_leaf_nodes=97
		min_samples_split=5
		random_state=42

Scikit-learn RandomForest	max_depth=randint(4, 50),
	min_samples_split=randint(3, 25)
	n_estimators=randint(10, 1000),
		max_depth=45
		min_samples_split=11
		n_estimators=748
		random_state=42

LightGBM Gradient-Boosted Decision Trees	learning_rate=random.uniform(0, 1)
	boosting_type=['gbdt']
	sub_feature=random.uniform(0, 1)
	num_leaves=randint(20, 300)
	min_data=randint(10, 100)
	max_depth=randint(5, 97)
		learning_rate=0.19422055180753994
		boosting_type=gbdt
		objective=multiclass'
		num_class=(2,)
		metric=multi_logloss
		sub_feature=0.4582447205067339
		num_leaves=150
		min_data=12
		max_depth=88

training sets, 14k entries for validation and 14k entries for test sets.

In addition to the size of the split and before applying split, shuffle the dataset using stratified k-fold cross validation used instead of random k-fold cross validation. We use stratified k-fold cross validation to preserve the percentage of examples for each class under splitting the data in train/test sets.

Table 4.7 shows model performance results evaluated with 14k MNIST test data using macro accuracy metric. The GBDT model has achieved best accuracy performance compared to other models, and Figure (4.6) shows the per-class accuracy comparison of each models.

Figure 4.5 shows confusion matrix results for classifiers: SVM, DT, RF and GBDT. We evaluate each classifier on 14k MNIST test data (original example). As we can see, we had fewer errors on SVM, RF and GBDT models than in DT, where one digit was misclassified as another digit.

Table 4.6: Hyperparameter configuration used for model training on MNIST

Classifier	Grid search intervals	Model hyperparameter

Scikit-learn SVC	kernel=['poly', 'rbf']
	C:[0.001]
	gamma=[10]
		C=0.001
		gamma=10
		kernel=poly
		probability=True
		random_state=42

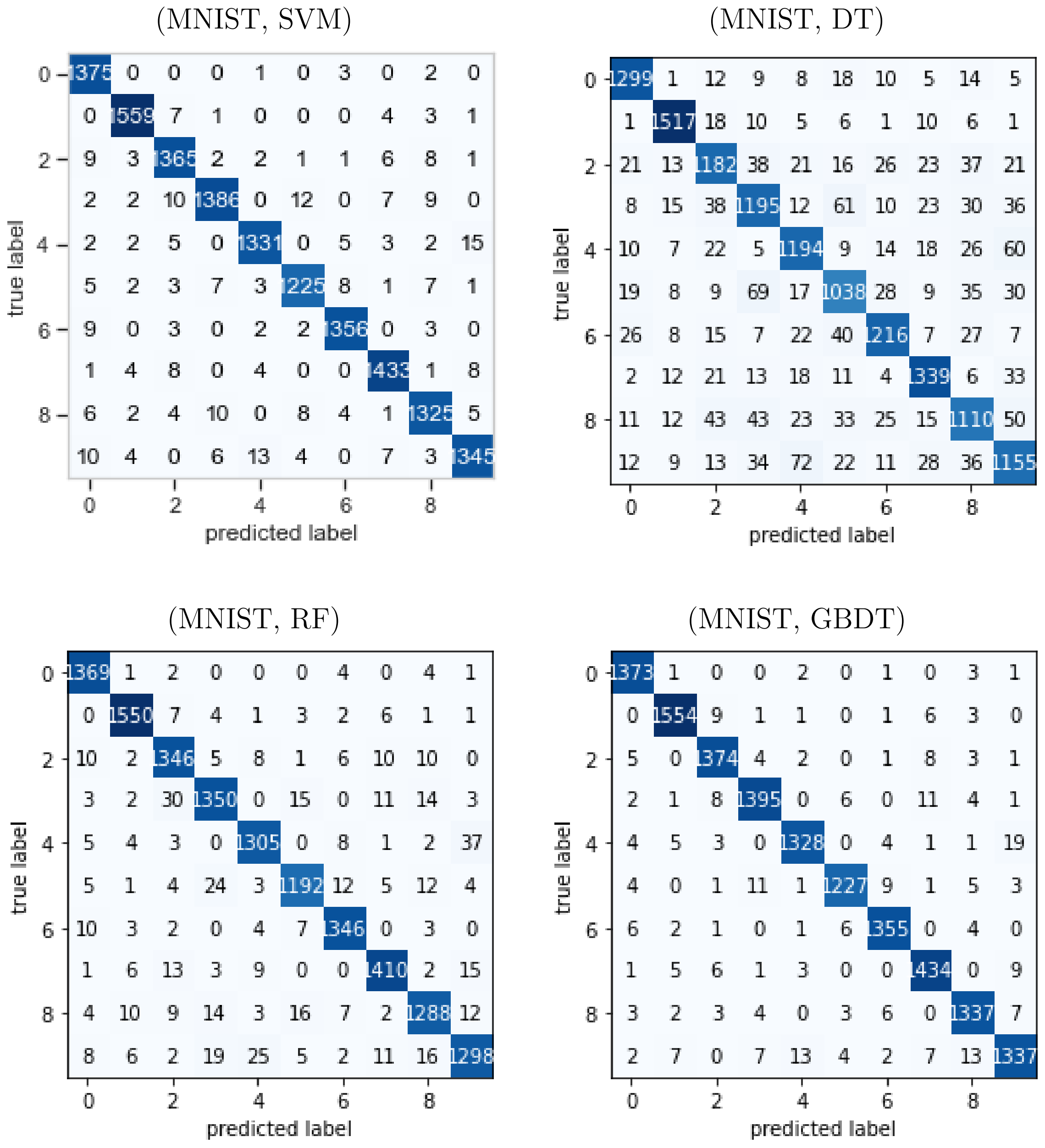
Scikit-learn DecisionTree	max_depth=(2, 4, 10, 784, None)	max_depth=784
		random_state=42

Scikit-learn RandomForest	n_estimators=(10, 25, 50, 100)	max_depth=784
		n_estimators=25
		n_jobs=-1
		random_state=42

LightGBM Gradient-Boosted Decision Trees	learning_rate=random.uniform(0, 1)
	boosting_type=['gbdt']
	sub_feature=random.uniform(0, 1)
	num_leaves=random.randint(20, 300)
	min_data=random.randint(10, 100)
	max_depth=random.randint(5, 784)
		learning_rate=0.242
		boosting_type=gbdt
		objective=multiclass
		num_class=(10,)
		metric=multi_logloss
		sub_feature=0.986
		num_leaves=179
		min_data=88
		max_depth=596
		num_iterations=100
		early_stopping_round=None

Table 4.7: Experimental result on 14k MNIST test data. (best results in boldface)

Model	Accuracy
SVM	0.9957
DT	0.9749
RF	0.9922
GBDT	**0.9959**



(MNIST, SVM)

1375	0	0	0	1	0	3	0	2	0
0	1559	7	1	0	0	0	4	3	1
9	3	1365	2	2	1	1	6	8	1
2	2	10	1386	0	12	0	7	9	0
2	2	5	0	1331	0	5	3	2	15
5	2	3	7	3	1225	8	1	7	1
9	0	3	0	2	2	1356	0	3	0
1	4	8	0	4	0	0	1433	1	8
6	2	4	10	0	8	4	1	1325	5
10	4	0	6	13	4	0	7	3	1345

(MNIST, DT)

1299	1	12	9	8	18	10	5	14	5
1	1517	18	10	5	6	1	10	6	1
21	13	1182	38	21	16	26	23	37	21
8	15	38	1195	12	61	10	23	30	36
10	7	22	5	1194	9	14	18	26	60
19	8	9	69	17	1038	28	9	35	30
26	8	15	7	22	40	1216	7	27	7
2	12	21	13	18	11	4	1339	6	33
11	12	43	43	23	33	25	15	1110	50
12	9	13	34	72	22	11	28	36	1155

(MNIST, RF)

1369	1	2	0	0	0	4	0	4	1
0	1550	7	4	1	3	2	6	1	1
10	2	1346	5	8	1	6	10	10	0
3	2	30	1350	0	15	0	11	14	3
5	4	3	0	1305	0	8	1	2	37
5	1	4	24	3	1192	12	5	12	4
10	3	2	0	4	7	1346	0	3	0
1	6	13	3	9	0	0	1410	2	15
4	10	9	14	3	16	7	2	1288	12
8	6	2	19	25	5	2	11	16	1298

(MNIST, GBDT)

1373	1	0	0	2	0	1	0	3	1
0	1554	9	1	1	0	1	6	3	0
5	0	1374	4	2	0	1	8	3	1
2	1	8	1395	0	6	0	11	4	1
4	5	3	0	1328	0	4	1	1	19
4	0	1	11	1	1227	9	1	5	3
6	2	1	0	1	6	1355	0	4	0
1	5	6	1	3	0	0	1434	0	9
3	2	3	4	0	3	6	0	1337	7
2	7	0	7	13	4	2	7	13	1337

Figure 4.5: Visualization of confusion matrix result on MNIST test data

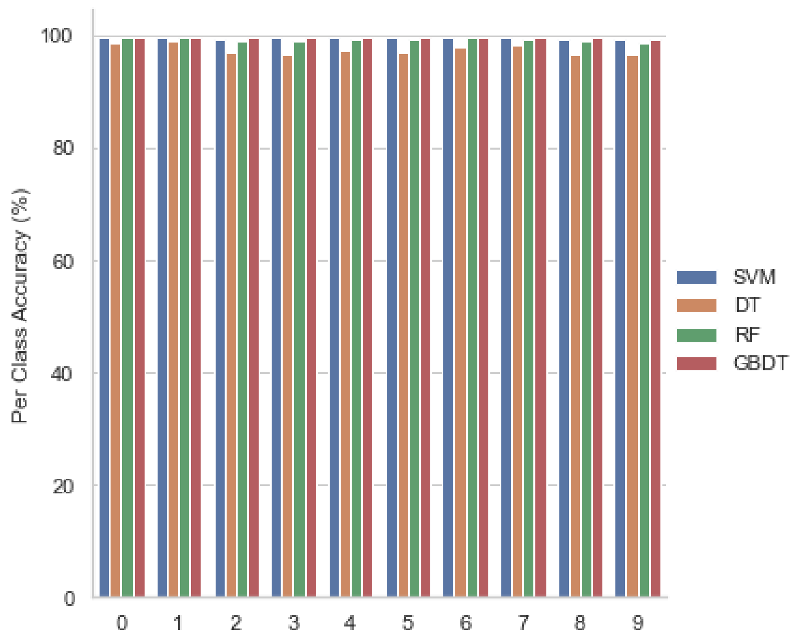


Figure 4.6: Classifier's per-class accuracy results on MNIST test data

4.4 Experimental Evaluation

Following reviews on adversarial attacks implemented in ART and trained models, we evaluate ART attacks against trained models on census and MNIST data at test and training time. All the adversarial examples were generated on the test data.

We evaluates the fooling (success) rate[12] , i.e., the percentage of successful adversarial examples of an attack given by:

$$Success_A = \sum_{i=1}^{n_{examples}} \begin{cases} 1(y_{adv_i} = y_i) & if\, A\, is\, a\, targeted\, attack \\ 1(y_{adv_i} \neq y_{o_i}) & otherwise \end{cases}$$

$$Success\, rate_A = \frac{1}{n_{examples}}(Success_A)$$

Where the i-th entry of y_o and y_{adv} specifies the predicted label of original example and adversarial example, respectively. whereas, the i-th entry of y is a target label assigned by the attack A. Note that, the success rate in targeted attack setting calculate based on the attack to the desired (or assigned) class label only.

Our experiments organized in two sections: (1) evasion attacks in ART, (2) data poisoning in ART.

4.4.1 Evasion attacks in ART

In this section, we report on the evasion attacks implemented in ART on chosen trained models. Tables 4.8 and 4.9 show the performance of those models before attack (or model performance on original examples). We consider two adversarial

[12]art/utils.py

settings: (1) Targeted attack, in which the attack uses random_targets[12] function to generate random class labels but different from the corresponding input true labels. for example, in binary classification task, the class label assigned to its corresponding opposite class label; (2) Untargeted attack, we didn't provide true labels with the classifier input because this approach prone to label leaking effect, instead for target label, we use the prediction of the classification of the input to prevent the label leaking problem (Nicolae et al., 2018). Figure 4.7 shows the confusion matrix that gives information to know how many digits were misclassified for 100 MNIST original examples by each baseline models.

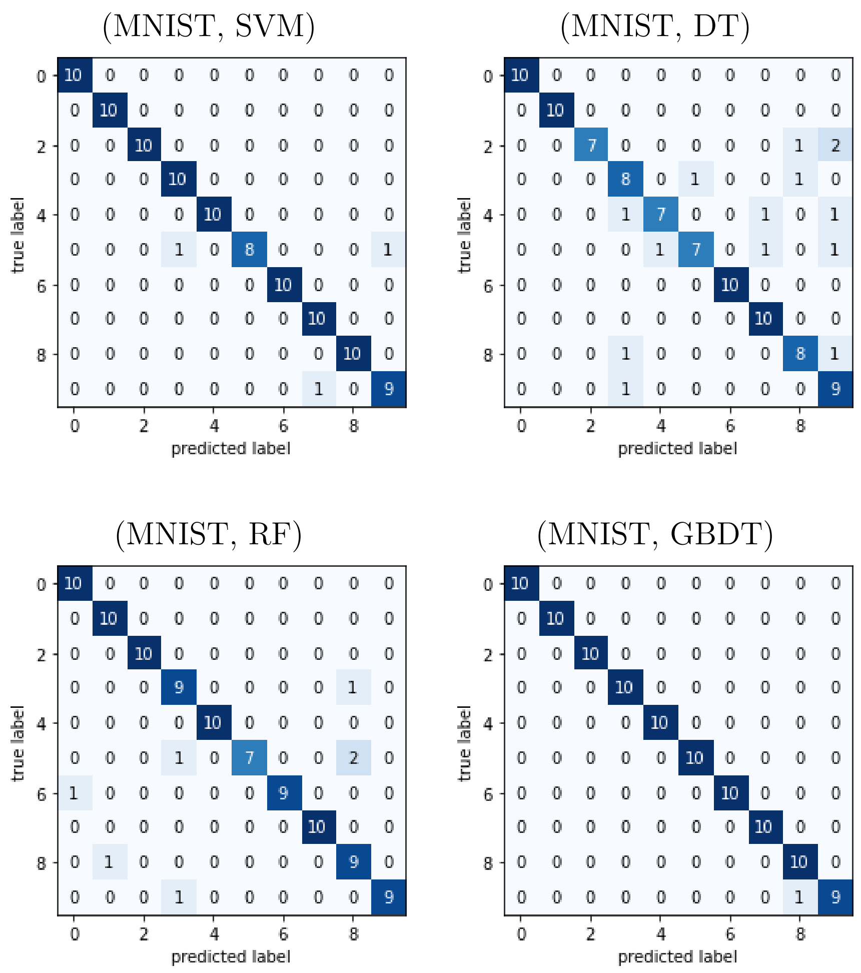


Figure 4.7: Visualization of confusion matrix for 100 MNIST original examples

Table 4.8: Experimental result on census 1500 original examples. (best results in boldface)

Baseline models	Precision	Recall	F_1 score	MCC
SVM	0.628	0.35	0.804	0.363
DT	0.592	**0.781**	0.827	0.568
RF	0.667	0.723	0.854	0.599
GBDT	**0.694**	0.714	**0.863**	**0.615**

Table 4.9: Experimental result on MNIST 100 original examples. (best results in boldface)

Baseline models	Accuracy
SVM	0.994
DT	0.972
RF	0.986
GBDT	**0.998**

Decision Tree-based attack

In this section, we report on the performance of ART DecisionTree attack against decision trees with different values of offsets/threshold, such as 0.0001, 0.001, 0.01, 0.1, and 1. Note that, this attack method gives similar result to the offset specified here and supports scikit-learn DecisionTree classifier only.

On Tabular data type: To evaluate this attack on tabular data type, we uses all census test data, which is approximately 9k original examples. Since, this attack is fast, we generates approximately 9k adversarial examples against decision trees in untargeted setting. As a result, the model wrongly predict by 92.38% on the adversarial examples. Table 4.10 shows the experimental result of ART DecisionTree attack with offset=0.01. We can see that the model performance on adversarial examples are decreases (best results in boldface).

Table 4.10: Experimental results using ART DecisionTree attack against decision trees on census.

Test data	Precision	Recall	Micro F1 score	MCC	Fooling rate(%)
Original	**0.614**	**0.763**	**0.828**	**0.571**	-
Adversarial	0.082	0.214	0.237	-0.478	**92.38**

On Image data type: We evaluate adversarial attacked examples generated by ART DecisionTree attack with different values of offsets on 100 MNIST original examples in the untargeted setting. The model test accuracy on adversarial examples and original examples are 2% and 97.2%, respectively. In addition, the fooling rate of this attack has 100%. As an example, consider Annex 5 shows the prediction on adversarial examples generated with 0.1 and 1 offsets. The adversarial example produced by this attack are very similar to the original image while being misclassified by the model, which is 100 percent wrong prediction (Annex 5 shows the comparison of this attack with Boundary attack, HSJA(ℓ_2), and ZOO attack), DecisionTree attack outperformed all other models against decision trees in terms of the adversarial images produced are similar with the original image and fooling rate, but its limitation is, it works only in the untargeted setting only. On the other hand, accuracy depend on the model performance, in which the performance of the model is 97.2% on original examples (see Table 4.9). For example, the targeted model classify digit 3 by 5 on original examples and then the same digit classified as 3 in adversarial examples. The result is shown in Figure 4.8.

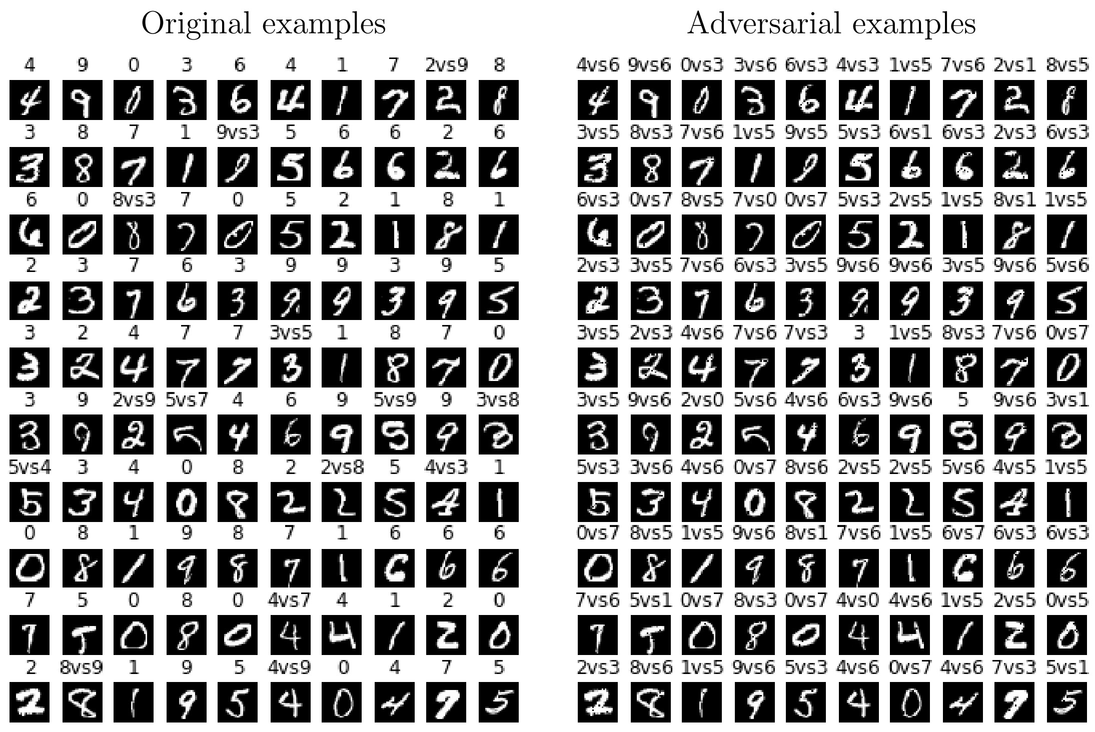

Original examples
Adversarial examples
4 9 0 3 6 4 1 7 2vs9 8
3 8 7 1 9vs3 5 6 6 2 6
6 0 8vs3 7 0 5 2 1 8 1
2 3 7 6 3 9 9 3 9 5
3 2 4 7 7 3vs5 1 8 7 0
3 9 2vs9 5vs7 4 6 9 5vs9 9 3vs8
5vs4 3 4 0 8 2 2vs8 5 4vs3 1
0 8 1 9 8 7 1 6 6 6
7 5 0 8 0 4vs7 4 1 2 0
2 8vs9 1 9 5 4vs9 0 4 7 5
4vs6 9vs6 0vs3 3vs6 6vs3 4vs3 1vs5 7vs6 2vs1 8vs5
3vs5 8vs3 7vs6 1vs5 9vs5 5vs3 6vs1 6vs3 2vs3 6vs3
6vs3 0vs7 8vs5 7vs0 0vs7 5vs3 2vs5 1vs5 8vs1 1vs5
2vs3 3vs5 7vs6 6vs3 3vs5 9vs6 9vs6 3vs5 9vs6 5vs6
3vs5 2vs3 4vs6 7vs6 7vs3 3 1vs5 8vs3 7vs6 0vs7
3vs5 9vs6 2vs0 5vs6 4vs6 6vs3 9vs6 5 9vs6 3vs1
5vs3 3vs6 4vs6 0vs7 8vs6 2vs5 2vs5 5vs6 4vs5 1vs5
0vs7 8vs5 1vs5 9vs6 8vs1 7vs6 1vs5 6vs7 6vs3 6vs3
7vs6 5vs1 0vs7 8vs3 0vs7 4vs0 4vs6 1vs5 2vs5 0vs5
2vs3 8vs6 1vs5 9vs6 5vs3 4vs6 0vs7 4vs6 7vs3 5vs1

Figure 4.8: MNIST: ART DecisionTree attack against Decision Trees with offset=0.01

Gradient-based attacks

In this section, we run two experiments using the parameters in Table 4.11 in the targeted and untargeted settings. First, we evaluates ART FGM, FGSM, BIM, and PGD attacks against SVM model on 1500 census and 100 MNIST test data. Second, we evaluates ART FGM, FGSM, UP and UAP against SVM model on 1500 census and 100 MNIST test data.

On Tabular data type: ART FGM, FGSM, BIM, and PGD attacks

- **Targeted setting:** In Table 4.12, FGM(ℓ_1) and FGM(ℓ_2) attack achieves 95.06% and 95.33% success rate with $\varepsilon = 0.1$, respectively. However, FGSM(ℓ_∞) attack achieve 97.8% success rate with $\varepsilon = 0.1$, whereas FGM(ℓ_1) and FGM(ℓ_2) at ε=0.5 and ε=0.3, respectively. F_1 score drops from 0.804 on original examples (Table 4.8) to 0.049, 0.047, and 0.022 on adversarial examples created under the FGM(ℓ_1), FGM(ℓ_2), and FGSM(ℓ_∞) attacks, respectively. Figure 4.9 shows that FGSM(ℓ_∞) performance metrics when computing with and without minimal perturbation budget. BIM(ℓ_∞) attack with ε=0.1 achieves 97.8% fooling rate, and MCC drops from 0.363 on original examples (Table 4.8) to -0.937 on adversarial examples. In PGD attack, with ε=0.1, we achieves 95.6%, 97.39%, and 97.8% fooling rate under the ℓ_1, ℓ_2, and ℓ_∞ distance metrics, respectively. MCC drops from 0.363 on original examples to -0.874, -0.926, and -0.937 on adversarial examples created by PGD(ℓ_1), PGD(ℓ_2), PGD(ℓ_∞), respectively. Since ART implements an extension on FGM/FGSM attack, the attack success rate achieved is comparable with the iterative versions, i.e., BIM and PGD attacks.

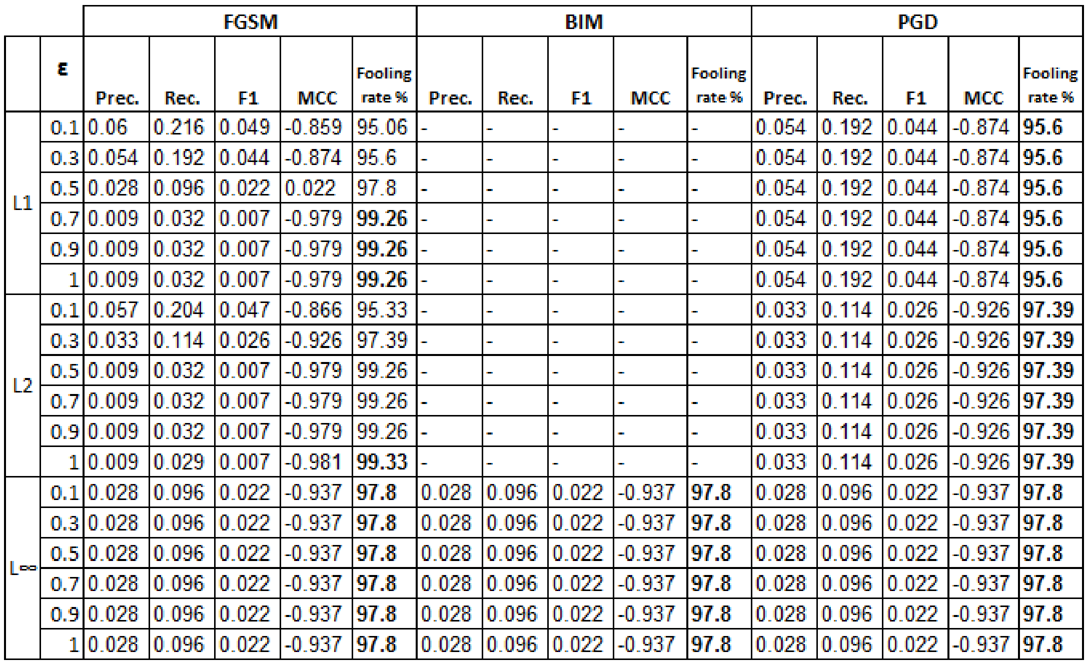
- **Untargeted setting:** In Table 4.13, those attacks fooling rate is similar with in the targeted setting, But the performance of the model is changed,

Table 4.11: Parameters for the gradient-based attack algorithms

Attack Algorithm	Parameters
FGM(ℓ_1), FGM(ℓ_2), FGSM(ℓ_∞)	ε=0.1, 0.3, 0.5, 0.7, 0.9, 1.0; ε_{step}=0.1; minimal perturbation=True
BIM(ℓ_∞)	ε=0.1, 0.3, 0.5, 0.7, 0.9, 1.0; ε_{step}=0.1; maximum iteration=2
PGD(ℓ_1), PGD(ℓ_2), PGD(ℓ_∞)	ε=0.1, 0.3, 0.5, 0.7, 0.9, 1.0; ε_{step}=0.1; maximum iteration=2
UP(ℓ_1), UP(ℓ_2), UP(ℓ_∞)	ε=0.1, 0.3, 0.5, 0.7, 0.9, 1.0; ε_{step}=0.1; maximum iteration=1
UAP(ℓ_1), UAP(ℓ_2), UAP(ℓ_∞)	ε=0.1, 0.3, 0.5, 0.7, 0.9, 1.0; ε_{step}=0.1; maximum iteration=1
C&W(ℓ_2), C&W(ℓ_∞)	ε=0.1, 0.3, 0.5, 0.7, 0.9, 1.0; ε_{step}=0.1; maximum iteration=2
JSMA(ℓ_0)	θ=0.1, 0.3, 0.5, 0.7, 0.9, 1.0; γ=0.1; maximum iteration=2
NewtonFool	η=0.1, 0.3, 0.5, 0.7, 0.9, 1.0; maximum iteration=2
DeepFool	ε=0.1, 0.3, 0.5, 0.7, 0.9, 1.0; nb_grads=10; maximum iteration=2
EAD(ℓ_1), EAD(ℓ_2), EAD(EN)	ε=0.1, 0.3, 0.5, 0.7, 0.9, 1.0; maximum iteration=2
VAT(ℓ_2)	ε=0.1, 0.3, 0.5, 0.7, 0.9, 1.0; finite_diff=1e-6; maximum iteration=2

For example, MCC result increases from -0.937 on targeted attack (Table 4.12) to -0.248.

Table 4.12: census: Experimental results on 1500 adversarial examples generated by ART FGSM, BIM, and PGD attacks under the ℓ_1, ℓ_2, and ℓ_∞ distance metrics bounded with different values of ε in **targeted** setting (best attack success rate in boldface).



		FGSM					BIM					PGD				
	ε	Prec.	Rec.	F1	MCC	Fooling rate %	Prec.	Rec.	F1	MCC	Fooling rate %	Prec.	Rec.	F1	MCC	Fooling rate %
L1	0.1	0.06	0.216	0.049	-0.859	95.06	-	-	-	-	-	0.054	0.192	0.044	-0.874	**95.6**
	0.3	0.054	0.192	0.044	-0.874	95.6	-	-	-	-	-	0.054	0.192	0.044	-0.874	**95.6**
	0.5	0.028	0.096	0.022	0.022	97.8	-	-	-	-	-	0.054	0.192	0.044	-0.874	**95.6**
	0.7	0.009	0.032	0.007	-0.979	**99.26**	-	-	-	-	-	0.054	0.192	0.044	-0.874	**95.6**
	0.9	0.009	0.032	0.007	-0.979	**99.26**	-	-	-	-	-	0.054	0.192	0.044	-0.874	**95.6**
	1	0.009	0.032	0.007	-0.979	**99.26**	-	-	-	-	-	0.054	0.192	0.044	-0.874	**95.6**
L2	0.1	0.057	0.204	0.047	-0.866	95.33	-	-	-	-	-	0.033	0.114	0.026	-0.926	**97.39**
	0.3	0.033	0.114	0.026	-0.926	97.39	-	-	-	-	-	0.033	0.114	0.026	-0.926	**97.39**
	0.5	0.009	0.032	0.007	-0.979	99.26	-	-	-	-	-	0.033	0.114	0.026	-0.926	**97.39**
	0.7	0.009	0.032	0.007	-0.979	99.26	-	-	-	-	-	0.033	0.114	0.026	-0.926	**97.39**
	0.9	0.009	0.032	0.007	-0.979	99.26	-	-	-	-	-	0.033	0.114	0.026	-0.926	**97.39**
	1	0.009	0.029	0.007	-0.981	**99.33**	-	-	-	-	-	0.033	0.114	0.026	-0.926	**97.39**
L∞	0.1	0.028	0.096	0.022	-0.937	**97.8**	0.028	0.096	0.022	-0.937	**97.8**	0.028	0.096	0.022	-0.937	**97.8**
	0.3	0.028	0.096	0.022	-0.937	**97.8**	0.028	0.096	0.022	-0.937	**97.8**	0.028	0.096	0.022	-0.937	**97.8**
	0.5	0.028	0.096	0.022	-0.937	**97.8**	0.028	0.096	0.022	-0.937	**97.8**	0.028	0.096	0.022	-0.937	**97.8**
	0.7	0.028	0.096	0.022	-0.937	**97.8**	0.028	0.096	0.022	-0.937	**97.8**	0.028	0.096	0.022	-0.937	**97.8**
	0.9	0.028	0.096	0.022	-0.937	**97.8**	0.028	0.096	0.022	-0.937	**97.8**	0.028	0.096	0.022	-0.937	**97.8**
	1	0.028	0.096	0.022	-0.937	**97.8**	0.028	0.096	0.022	-0.937	**97.8**	0.028	0.096	0.022	-0.937	**97.8**

In Table 4.17, PGD(ℓ_1), PGD(ℓ_2), and PGD(ℓ_∞) attacks achieves 97.6%, 97.8%, and 97.8% fooling rate with perturbation budget $\varepsilon = 0.1$ and iterations= 1, 2, and 3 in the targeted and untargeted settings, respectively. The result shown that the fooling rate will increases with given iteration values.

On Image data type: ART FGSM, BIM, and PGD attacks

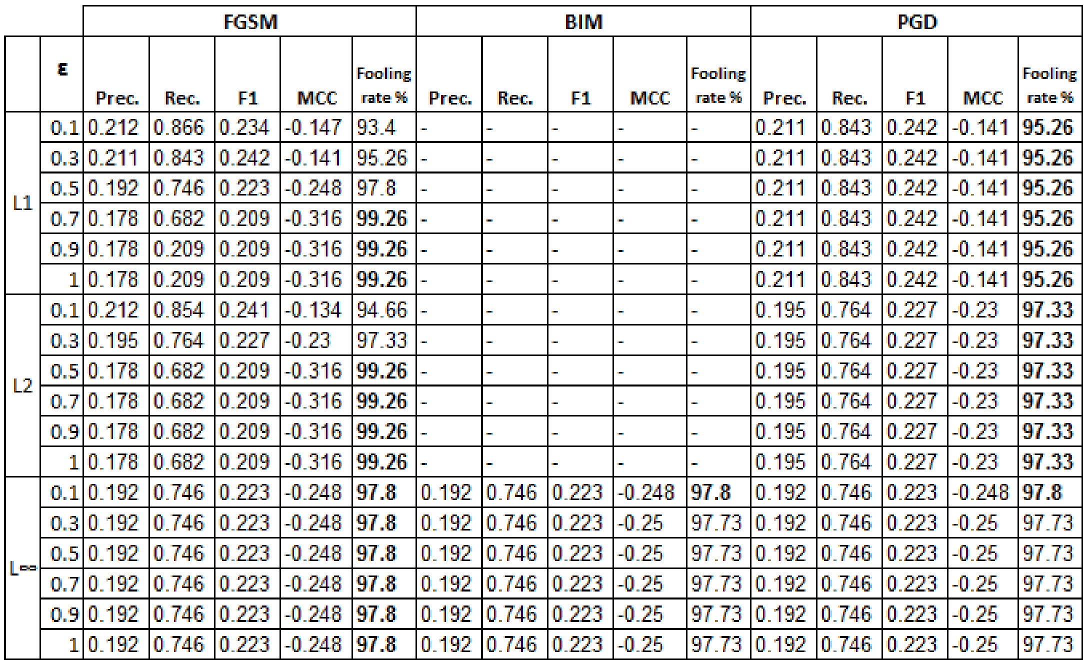
- **Targeted and untargeted setting:** Since these attack methods using ℓ_1 or ℓ_2 distance has 0% attack success rate. The following result in Table 4.14 is an adversarial examples generated with ℓ_∞ distance on MNIST data. In this setting, FGSM(ℓ_∞) has the highest fooling rate among BIM(ℓ_∞) and PGD(ℓ_∞) as shown in 2.1, 2.2 and 2.3 Annex 5.

On Tabular data type: ART FGM, FGSM, UP and UAP attacks

Universal perturbation, discussed in Section 3.1.1, executes another attacks iteratively, in this experiment, we evaluate the FGSM(ℓ_∞) algorithm runs inside UP/Targeted UAP algorithms versus FGSM(ℓ_∞). As a result, accumulation perturbations over all inputs has less attack success rate (see Tables 4.15, 4.16).

- **Targeted setting:** In Table 4.15, FGM(ℓ_1) and FGM(ℓ_2), and FGSM(ℓ_∞) attacks achieves 95.06%, 95.33%, and 97.8% fooling rate with perturbation budget $\varepsilon = 0.1$. UAP(ℓ_1), UAP(ℓ_2), and UAP(ℓ_∞) attacks with ε=0.1

Table 4.13: census: Experimental results on 1500 adversarial examples generated by ART FGSM, BIM, and PGD attacks under the ℓ_1, ℓ_2, and ℓ_∞ distance metrics bounded with different values of ε in **untargeted** setting (best attack success rate in boldface).



		FGSM					BIM					PGD				
	ε	Prec.	Rec.	F1	MCC	Fooling rate %	Prec.	Rec.	F1	MCC	Fooling rate %	Prec.	Rec.	F1	MCC	Fooling rate %
L1	0.1	0.212	0.866	0.234	-0.147	93.4	-	-	-	-	-	0.211	0.843	0.242	-0.141	**95.26**
	0.3	0.211	0.843	0.242	-0.141	95.26	-	-	-	-	-	0.211	0.843	0.242	-0.141	**95.26**
	0.5	0.192	0.746	0.223	-0.248	97.8	-	-	-	-	-	0.211	0.843	0.242	-0.141	**95.26**
	0.7	0.178	0.682	0.209	-0.316	**99.26**	-	-	-	-	-	0.211	0.843	0.242	-0.141	**95.26**
	0.9	0.178	0.209	0.209	-0.316	**99.26**	-	-	-	-	-	0.211	0.843	0.242	-0.141	**95.26**
	1	0.178	0.209	0.209	-0.316	**99.26**	-	-	-	-	-	0.211	0.843	0.242	-0.141	**95.26**
L2	0.1	0.212	0.854	0.241	-0.134	94.66	-	-	-	-	-	0.195	0.764	0.227	-0.23	**97.33**
	0.3	0.195	0.764	0.227	-0.23	97.33	-	-	-	-	-	0.195	0.764	0.227	-0.23	**97.33**
	0.5	0.178	0.682	0.209	-0.316	**99.26**	-	-	-	-	-	0.195	0.764	0.227	-0.23	**97.33**
	0.7	0.178	0.682	0.209	-0.316	**99.26**	-	-	-	-	-	0.195	0.764	0.227	-0.23	**97.33**
	0.9	0.178	0.682	0.209	-0.316	**99.26**	-	-	-	-	-	0.195	0.764	0.227	-0.23	**97.33**
	1	0.178	0.682	0.209	-0.316	**99.26**	-	-	-	-	-	0.195	0.764	0.227	-0.23	**97.33**
L∞	0.1	0.192	0.746	0.223	-0.248	**97.8**	0.192	0.746	0.223	-0.248	**97.8**	0.192	0.746	0.223	-0.248	**97.8**
	0.3	0.192	0.746	0.223	-0.248	**97.8**	0.192	0.746	0.223	-0.25	97.73	0.192	0.746	0.223	-0.25	97.73
	0.5	0.192	0.746	0.223	-0.248	**97.8**	0.192	0.746	0.223	-0.25	97.73	0.192	0.746	0.223	-0.25	97.73
	0.7	0.192	0.746	0.223	-0.248	**97.8**	0.192	0.746	0.223	-0.25	97.73	0.192	0.746	0.223	-0.25	97.73
	0.9	0.192	0.746	0.223	-0.248	**97.8**	0.192	0.746	0.223	-0.25	97.73	0.192	0.746	0.223	-0.25	97.73
	1	0.192	0.746	0.223	-0.248	**97.8**	0.192	0.746	0.223	-0.25	97.73	0.192	0.746	0.223	-0.25	97.73

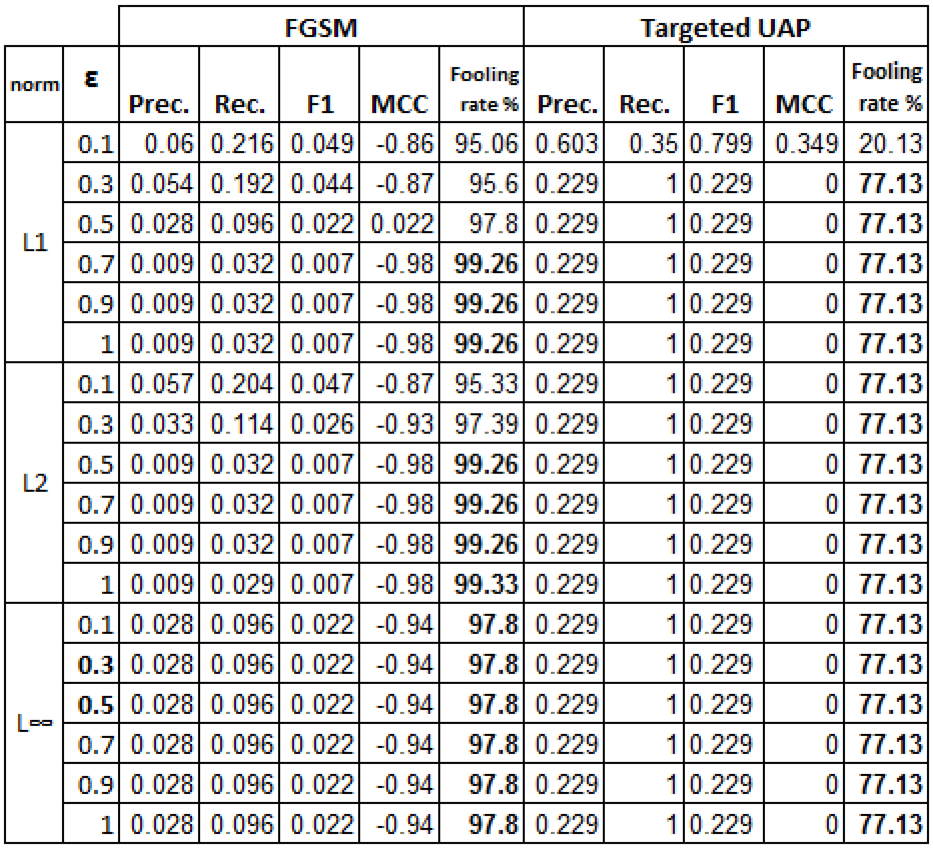
Table 4.14: MNIST: Experimental results on 100 adversarial examples FGSM(ℓ_∞), BIM(ℓ_∞), and PGD(ℓ_∞) attacks bounded with different values of ε in the targeted and untargeted settings (best attack success rate in boldface).

Objective		FGSM(ℓ_∞)		BIM(ℓ_∞)		PGD(ℓ_∞)	
	Epsilon(ε)	Acc.	Fooling rate(%)	Acc.	Fooling rate(%)	Acc.	Fooling rate(%)
Targeted	0.1	0.97	0	0.97	0	0.97	0
	0.2	0.85	5	0.83	**4**	0.83	3
	0.3	0.85	5	0.83	4	0.72	14
	0.4	0.18	**75**	0.83	4	0.61	**21**
Untargeted	0.1	0.89	12	0.88	13	0.88	13
	0.2	0.59	42	0.56	**45**	0.55	46
	0.3	0.59	42	0.56	45	0.38	63
	0.4	0.1	**91**	0.56	45	0.28	**73**

achieves 20.13%, 77.13%, and 77.13% fooling rate, respectively. MCC result drops from 0.363 on original examples to 0.349, 0, and 0 on adversarial examples created by UAP(ℓ_1), UAP(ℓ_2), and UAP(ℓ_∞) attacks, respectively.

- **Untargeted setting:** In Table 4.16 the FGM(ℓ_1),FGM(ℓ_2), and FGSM(ℓ_∞) attacks achieves 93.4%, 94.66%, and 97.8% fooling rate with $\varepsilon = 0.1$, respectively. All UP(ℓ_1), UP(ℓ_2), and UP(ℓ_∞) attacks with ε=0.1 achieves 86.73% fooling rate. MCC drops from 0.363 on original examples (Table 4.8) to 0 on adversarial examples created UP(ℓ_1), UP(ℓ_2), and UP(ℓ_∞) attacks.

Table 4.15: census: Experimental results on 1500 adversarial examples generated by ART Targeted FGSM and UAP attacks under the ℓ_1, ℓ_2, and ℓ_∞ distance metrics bounded with different values of ε (best attack success rate in boldface).



		FGSM					Targeted UAP				
norm	ε	Prec.	Rec.	F1	MCC	Fooling rate %	Prec.	Rec.	F1	MCC	Fooling rate %
L1	0.1	0.06	0.216	0.049	-0.86	95.06	0.603	0.35	0.799	0.349	20.13
	0.3	0.054	0.192	0.044	-0.87	95.6	0.229	1	0.229	0	**77.13**
	0.5	0.028	0.096	0.022	0.022	97.8	0.229	1	0.229	0	**77.13**
	0.7	0.009	0.032	0.007	-0.98	**99.26**	0.229	1	0.229	0	**77.13**
	0.9	0.009	0.032	0.007	-0.98	**99.26**	0.229	1	0.229	0	**77.13**
	1	0.009	0.032	0.007	-0.98	**99.26**	0.229	1	0.229	0	**77.13**
L2	0.1	0.057	0.204	0.047	-0.87	95.33	0.229	1	0.229	0	**77.13**
	0.3	0.033	0.114	0.026	-0.93	97.39	0.229	1	0.229	0	**77.13**
	0.5	0.009	0.032	0.007	-0.98	**99.26**	0.229	1	0.229	0	**77.13**
	0.7	0.009	0.032	0.007	-0.98	**99.26**	0.229	1	0.229	0	**77.13**
	0.9	0.009	0.032	0.007	-0.98	**99.26**	0.229	1	0.229	0	**77.13**
	1	0.009	0.029	0.007	-0.98	**99.33**	0.229	1	0.229	0	**77.13**
L∞	0.1	0.028	0.096	0.022	-0.94	**97.8**	0.229	1	0.229	0	**77.13**
	0.3	0.028	0.096	0.022	-0.94	**97.8**	0.229	1	0.229	0	**77.13**
	0.5	0.028	0.096	0.022	-0.94	**97.8**	0.229	1	0.229	0	**77.13**
	0.7	0.028	0.096	0.022	-0.94	**97.8**	0.229	1	0.229	0	**77.13**
	0.9	0.028	0.096	0.022	-0.94	**97.8**	0.229	1	0.229	0	**77.13**
	1	0.028	0.096	0.022	-0.94	**97.8**	0.229	1	0.229	0	**77.13**

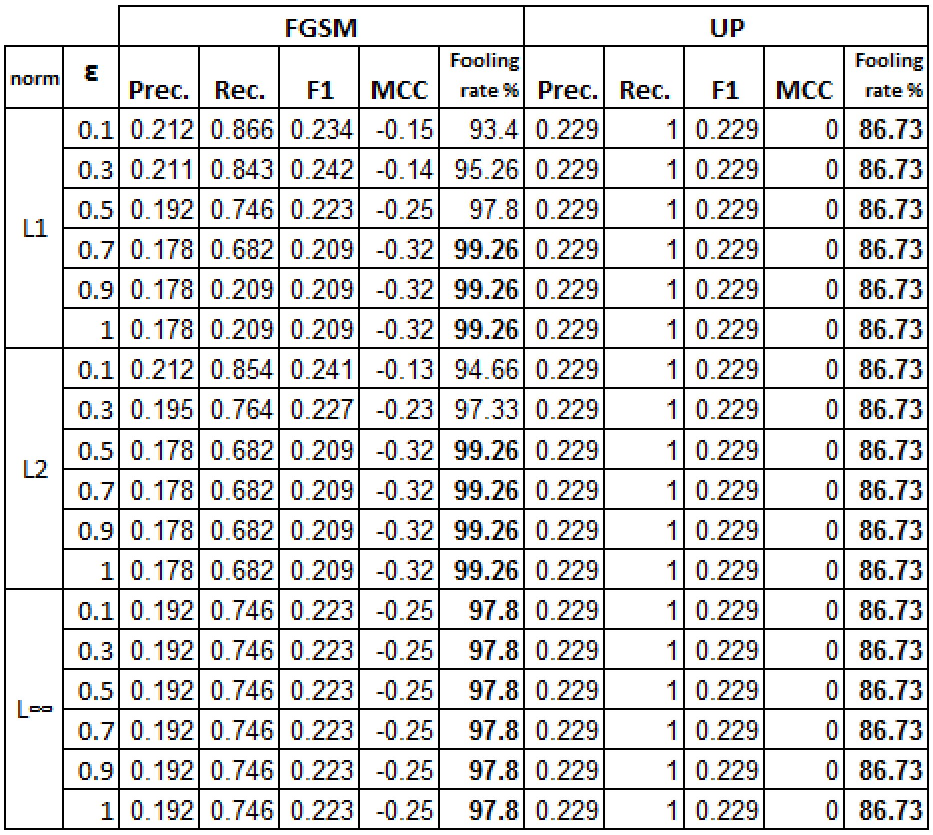
On Image data type: ART FGSM, UP and UAP attacks

- **Targeted and untargeted setting:** Since the FGSM(ℓ_1) and FGSM(ℓ_2) attacks fooling rate is 0% on MNIST data, Table 4.18 shows ART FGSM(ℓ_∞), UP(ℓ_∞), and Targeted UAP(ℓ_∞) attacks only. In both attack settings, FGSM(ℓ_∞) has the highest fooling rate than UP(ℓ_∞) and UAP(ℓ_∞) attacks.

Decision-based attacks

In both targeted and untargeted setting, we evaluate decision-based attacks: ART Boundary attack and HopSkipJump attack on SVM, DT, RF, or GBDT on census

Table 4.16: census: Experimental results on 1500 adversarial examples generated by ART Untargeted FGSM and UP attacks under the ℓ_1, ℓ_2, and ℓ_∞ distance metrics bounded with different values of ε.



		FGSM					**UP**				
norm	**ε**	**Prec.**	**Rec.**	**F1**	**MCC**	**Fooling rate %**	**Prec.**	**Rec.**	**F1**	**MCC**	**Fooling rate %**
L1	0.1	0.212	0.866	0.234	-0.15	93.4	0.229	1	0.229	0	**86.73**
	0.3	0.211	0.843	0.242	-0.14	95.26	0.229	1	0.229	0	**86.73**
	0.5	0.192	0.746	0.223	-0.25	97.8	0.229	1	0.229	0	**86.73**
	0.7	0.178	0.682	0.209	-0.32	**99.26**	0.229	1	0.229	0	**86.73**
	0.9	0.178	0.209	0.209	-0.32	**99.26**	0.229	1	0.229	0	**86.73**
	1	0.178	0.209	0.209	-0.32	**99.26**	0.229	1	0.229	0	**86.73**
L2	0.1	0.212	0.854	0.241	-0.13	94.66	0.229	1	0.229	0	**86.73**
	0.3	0.195	0.764	0.227	-0.23	97.33	0.229	1	0.229	0	**86.73**
	0.5	0.178	0.682	0.209	-0.32	**99.26**	0.229	1	0.229	0	**86.73**
	0.7	0.178	0.682	0.209	-0.32	**99.26**	0.229	1	0.229	0	**86.73**
	0.9	0.178	0.682	0.209	-0.32	**99.26**	0.229	1	0.229	0	**86.73**
	1	0.178	0.682	0.209	-0.32	**99.26**	0.229	1	0.229	0	**86.73**
L∞	0.1	0.192	0.746	0.223	-0.25	**97.8**	0.229	1	0.229	0	**86.73**
	0.3	0.192	0.746	0.223	-0.25	**97.8**	0.229	1	0.229	0	**86.73**
	0.5	0.192	0.746	0.223	-0.25	**97.8**	0.229	1	0.229	0	**86.73**
	0.7	0.192	0.746	0.223	-0.25	**97.8**	0.229	1	0.229	0	**86.73**
	0.9	0.192	0.746	0.223	-0.25	**97.8**	0.229	1	0.229	0	**86.73**
	1	0.192	0.746	0.223	-0.25	**97.8**	0.229	1	0.229	0	**86.73**

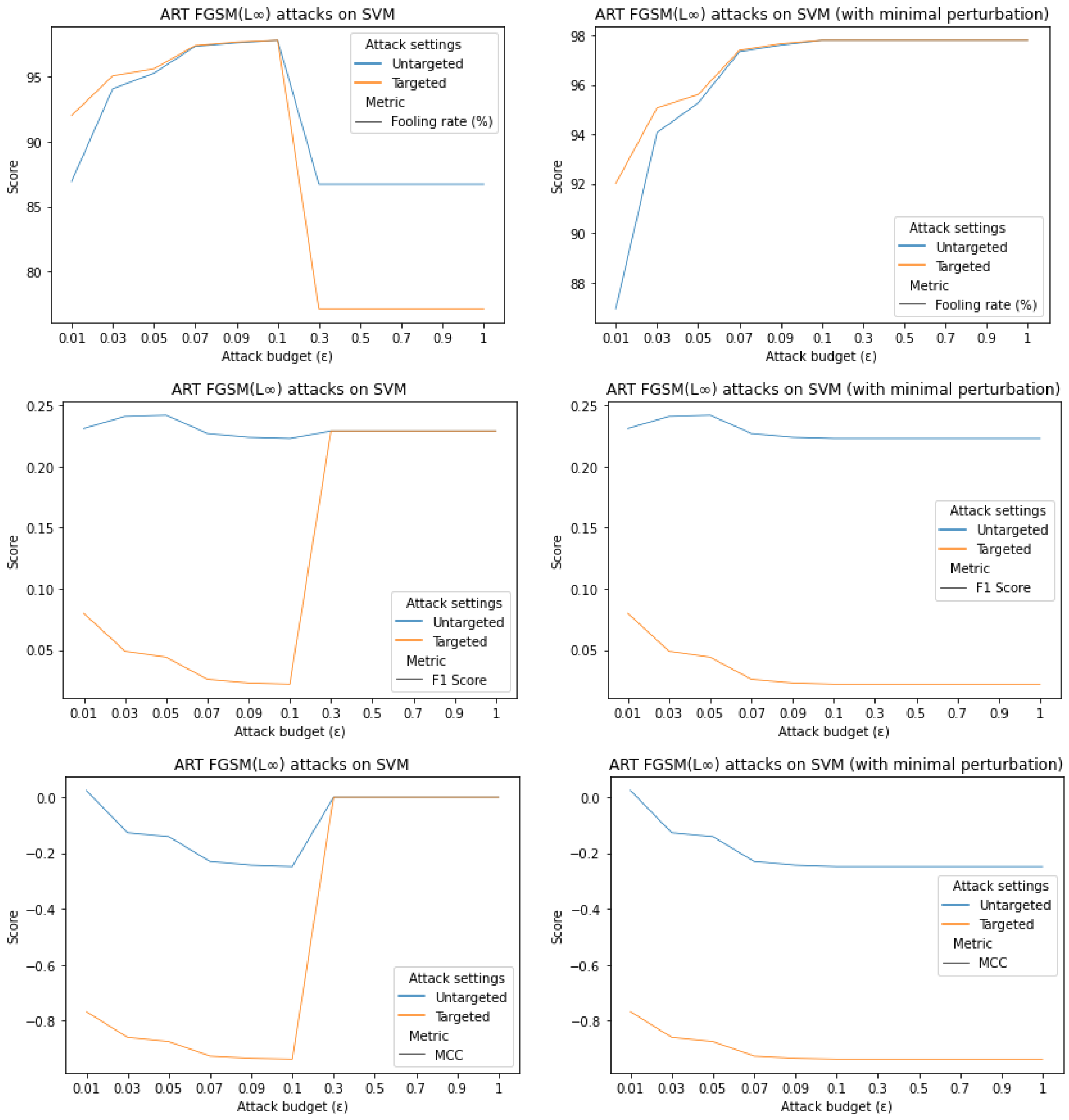


Figure 4.9: census: shows comparisons on SVM model performance and attack success rate on adversarial examples created by ART FGSM(ℓ_∞) with computing minimal perturbation or not. (Note that, we restrict the number of steps used to 0.01 for attack budgets from 0.01 to 0.1, and step size=0.1 for attack budgets from 0.1 to 1, in which to keep the computational cost of experiments manageable and correct.)

Table 4.17: census: Experimental results on 1500 adversarial examples generated by ART PGD attacks under the ℓ_1, ℓ_2, and ℓ_∞ distance metrics bounded by ε=0.1 with in different iterations in targeted and untargeted settings (best attack success rate in boldface).

Norm	Objective	Max. Iter.	Avg. Time (per attack)	Prec.	Rec.	F1	MCC	Fooling rate
ℓ_1	Targeted	1	2.58 sec	0.06	0.216	0.049	-0.859	95.06
		2	4.31 sec	0.054	0.192	0.044	-0.874	95.6
		3	10.5 sec	0.03	0.105	0.024	-0.932	**97.60**
	Untargeted	1	2.42 sec	0.212	0.866	0.234	-0.147	93.4
		2	5.68 sec	0.211	0.843	0.242	-0.141	95.26
		3	9.90 sec	0.194	0.755	0.225	-0.24	**97.53**
ℓ_2	Targeted	1	3.31 sec	0.057	0.204	0.047	-0.866	95.33
		2	4.79 sec	0.033	0.114	0.026	-0.926	97.39
		3	9.66 sec	0.028	0.096	0.022	-0.937	**97.80**
	Untargeted	1	2.47 sec	0.212	0.854	0.241	-0.134	94.66
		2	6.29 sec	0.195	0.764	0.227	-0.23	97.33
		3	6.65 sec	0.192	0.746	0.223	-0.248	**97.80**
ℓ_∞	Targeted	1	3.69 sec	0.028	0.096	0.022	-0.937	97.8
		2	4.64 sec	0.028	0.096	0.022	-0.937	97.8
		3	7.99 sec	0.028	0.096	0.022	-0.937	**97.80**
	Untargeted	1	3.1 sec	0.192	0.746	0.223	-0.248	97.8
		2	4.47 sec	0.192	0.746	0.223	-0.248	97.8
		3	10.09 sec	0.192	0.746	0.223	-0.248	**97.80**

Table 4.18: MNIST: Targeted and untargeted FGSM, UP, and UAP. Experimental results on 100 adversarial samples (best fooling rate in boldface).

Objective		FGSM(ℓ_∞)		Targeted UAP(ℓ_∞)		UP(ℓ_∞)	
	Epsilon (ϵ)	Accuracy	Fooling rate(%)	Accuracy	Fooling rate(%)	Accuracy	Fooling rate(%)
Targeted	0.1	0.97	0	0.95	3	-	-
	0.3	0.85	5	0.59	41	-	-
	0.5	0.03	96	0.35	64	-	-
	0.7	0.02	**98**	0.28	71	-	-
	0.9	0.02	**98**	0.13	**87**	-	-
	1	0.02	**98**	0.2	80	-	-
Untargeted	0.1	0.89	12	-	-	0.96	4
	0.3	0.59	42	-	-	0.89	12
	0.5	0.01	**100**	-	-	0.5	49
	0.7	0.01	**100**	-	-	0.15	**84**
	0.9	0.01	**100**	-	-	0.3	70
	1	0.01	**100**	-	-	0.54	45

and MNIST data with different parameter values. Table 4.19 shows the parameters used for decision-based attack algorithms.

Table 4.19: Parameters for the decision-based attack algorithms

Attack Algorithms	Parameters
HJSA(ℓ_∞), HJSA(ℓ_2)	Maximum number of iterations (max_iter=2), Maximum number of evaluations for estimating gradient(max_eval=4); Initial number of evaluations for estimating gradient(init_eval=2), Maximum number of trials for initial generation of adversarial examples (init_size=1)
BA	length of the total perturbation (orthogonal step $\delta = 0.01$), step towards the target ($\varepsilon = 0.01$), step_adapt=0.01, Maximum number of iterations=2 (but for MNIST 100 iterations used), Number of trials per iteration=5, Number of samples per trial=10, Number of trials for initial generation of adversarial examples=10

On Tabular data type: The following Tables 4.20 and 4.21 shows experimental results for adversarial examples created by ART Boundary and HopSkipJump attacks on 1500 census test data, respectively (Best attack result in bold). HopSkipJump attacks on random forest is computationally expensive. In both attack algorithms, the targeted attack setting achieves the highest fooling rate.

On Image data type: The following Table 4.22 and Figure 4.23 shows experimental results for adversarial examples created by ART Boundary and HopSkipJump attacks on 100 MNIST test data, respectively (Best attack result in bold). We show adversarial examples created by the ART Boundary attack for MNIST in 3.2, Annex 5. and, by HopSkipJump attack in 3.1, Annex 5.

For example, adversarial examples is produced by Boundary attack with minimal perturbations computed under ℓ_2 norm bound with ϵ=0.01, $\delta = 0.01$, and iterations of 10, 100,and 1000 are shown in the Figure 4.10.

Table 4.20: Boundary attack for census 1500 examples with $\varepsilon = 0.01$ *and* $\delta = 0.01$ in the targeted and untargeted settings (best attack success rate in boldface)

Model	Objective	Avg. time (per attack)	Prec.	Rec.	F1	MCC	Fooling rate
SVM	Targeted	0.26 sec	0	0	0.719	-0.128	**28.13**
	Untargeted	0.33 sec	0	0	0.771	0	13.26
DT	Targeted	0.01 sec	0.123	0.472	0.108	-0.68	**89.2**
	Untargeted	0.01 sec	0.18	0.697	0.203	-0.326	81.13
RF	Targeted	1.29 sec	0.1	0.376	0.086	-0.749	**91.4**
	Untargeted	1.6 sec	0.177	0.688	0.198	-0.345	86.4
GBDT	Targeted	0.01 sec	0.108	0.408	0.093	-0.727	**90.66**
	Untargeted	0.01 sec	0.184	0.729	0.199	-0.328	85.93

Table 4.21: ART HopSkipJump attack for census 1500 original examples with targeted and untargeted settings.

Model	Objective	Dist.	Avg. Time (per attack)	Prec.	Rec.	F1	MCC	Fooling rate
SVM	Targeted	ℓ_∞	0.4 sec	0.094	0.35	0.08	-0.767	**92**
		ℓ_2	0.33 sec	0.094	0.35	0.08	-0.767	**92**
	Untargeted	ℓ_∞	0.49 sec	0.229	1	0.229	0	86.73
		ℓ_2	0.4 sec	0.229	1	0.229	0	86.73
DT	Targeted	ℓ_∞	0.03 sec	0.184	0.738	0.193	-0.348	**80.66**
		ℓ_2	0.02 sec	0.186	0.741	0.201	-0.318	**79.86**
	Untargeted	ℓ_∞	0.03 sec	0.23	0.956	0.257	0.012	67.2
		ℓ_2	0.02 sec	0.228	0.95	0.254	-0.004	68.73
RF	Targeted	ℓ_∞	**7.82 sec**	0.176	0.682	0.197	-0.349	**80.26**
		ℓ_2	**5.98 sec**	0.175	0.679	0.193	-0.362	**80.66**
	Untargeted	ℓ_∞	**8.44 sec**	0.229	0.939	0.263	0.003	71.86
		ℓ_2	**6.73 sec**	0.229	0.927	0.271	0.006	71.86
GBDT	Targeted	ℓ_∞	0.11 sec	0.176	0.685	0.193	-0.361	**80.66**
		ℓ_2	0.09 sec	0.167	0.644	0.183	-0.405	**81.73**
	Untargeted	ℓ_∞	0.06 sec	0.233	0.959	0.267	0.037	72.66
		ℓ_2	0.05 sec	0.231	0.959	0.259	0.02	73.46

Table 4.22: ART HopSkipJump attack for MNIST 100 examples with targeted and untargeted settings.

Model	Objective	Distance	Avg. Time (per attack)	Accuracy	Fooling rate (%)
SVM	Targeted	ℓ_∞	0.17 sec	0.79	19
		ℓ_2	0.14 sec	0.82	17
	Untargeted	ℓ_∞	0.71 sec	0.1	**91**
		ℓ_2	0.59 sec	0.09	**92**
DT	Targeted	ℓ_∞	0.01 sec	0.8	11
		ℓ_2	0.01 sec	0.8	10
	Untargeted	ℓ_∞	0.04 sec	0.12	**87**
		ℓ_2	0.03 sec	0.11	**92**
RF	Targeted	ℓ_∞	0.08 sec	0.86	11
		ℓ_2	0.08 sec	0.81	15
	Untargeted	ℓ_∞	0.56 sec	0.12	**87**
		ℓ_2	0.42 sec	0.09	**92**
GBDT	Targeted	ℓ_∞	0.02 sec	0.79	20
		ℓ_2	0.01 sec	0.79	20
	Untargeted	ℓ_∞	0.06 sec	0.11	**89**
		ℓ_2	0.04 sec	0.11	**89**

Table 4.23: Boundary attack for MNIST 100 examples with $\varepsilon = 0.01$ and $\delta = 0.01$ in the targeted and untargeted settings (best attack success rate in boldface)

Model	Objective	Avg. Time (per attack)	Accuracy	Fooling rate (%)
SVM	Targeted	16.73 sec	0.66	33
	Untargeted	34.83 sec	0.02	**98**
DT	Targeted	0.49 sec	0.39	57.99
	Untargeted	0.83 sec	0.01	**100**
RF	Targeted	0.27 sec	0.79	20
	Untargeted	1.24 sec	0.1	**89**
GBDT	Targeted	3.97 sec	0.66	31
	Untargeted	11.46 sec	0.02	**99**

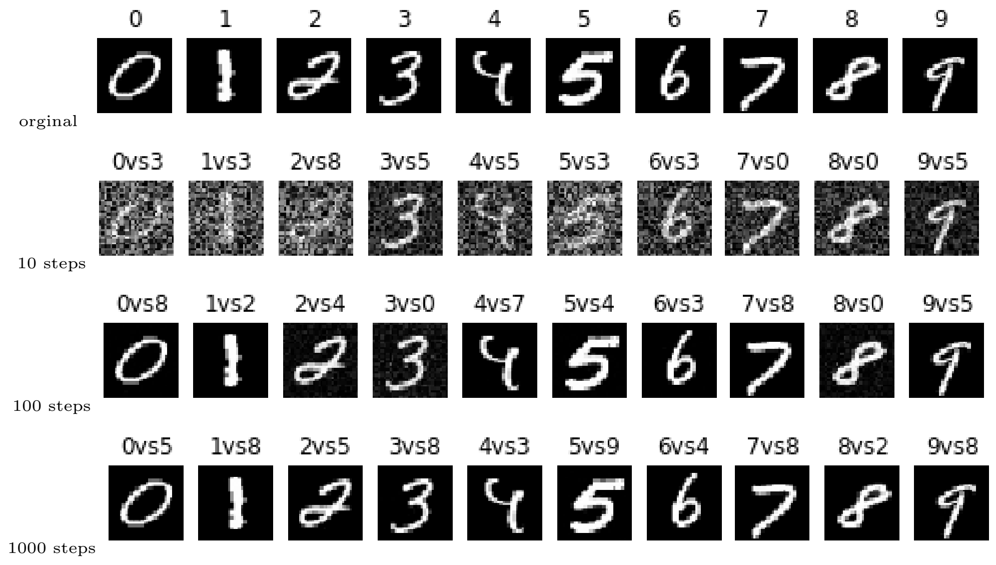


Figure 4.10: Image of original and adversarial examples generated by ART Boundary attack on MNIST with $\epsilon = 0.01$ and $\delta = 0.01$.

Score-based attacks

In this section, we report on the score-based attack in the targeted and untargeted setting. We compare ART ZOO attack on SVM, DTs, RF, or GBDTs for census and MNIST data with different parameter values used in Table .

Table 4.24: Parameters for the score-based attack algorithms

Attack Algorithm	Parameters
ZOO	Step size for numerical estimation of derivatives ($\varepsilon = [0.1, 0.3, 0.5, 0.7, 0.9]$), Maximum number of iterations (max_iter=2), confidence=0, learning_rate=0.01, binary_search_steps=2, initial_const=0.001, abort_early=True, nb_parallel=784(for MNIST), nb_parallel=97(for census), batch_size=1

On Image data type: We evaluate the ART ZOO attack against SVM, DT, RF, and GBDT on MNIST test set in the targeted and untargeted settings in Table 4.25. In this setting, we observe that SVM is robust to ZOO attack, but this attack is computationally expensive on SVM.

Table 4.25: MNIST: Experimental results on 100 adversarial examples generated by ART ZOO attacks with different values of ε in the targeted and untargeted settings (best attack success rate in boldface).

Model		Targeted			Untargeted		
	eps. (ε)	Avg. Time (per attack)	Acc.	Fooling rate(%)	Avg. Time (per attack)	Acc.	Fooling rate(%)
SVM	0.1	**91.29 sec**	0.98	0	**91.41 sec**	0.97	3
	0.3	**72.95 sec**	0.98	0	**72.98 sec**	0.97	3
	0.5	**74.07 sec**	0.98	0	**74.06 sec**	0.97	3
	0.7	**76.59 sec**	0.98	0	**76.59 sec**	0.97	3
	0.9	**79.72 sec**	0.98	0	**79.72 sec**	0.97	3
DT	0.1	2.69 sec	0.65	26	3.83 sec	0.23	**83**
	0.3	3.69 sec	0.66	25	3.76 sec	0.27	76
	0.5	3.72 sec	0.68	23	3.75 sec	0.27	74
	0.7	3.61 sec	0.68	23	3.87 sec	0.27	74
	0.9	3.56 sec	0.75	16	3.42 sec	0.3	71
RF	0.1	2.87 sec	0.67	28.9	2.86 sec	0.26	**76**
	0.3	2.88 sec	0.73	23	3 sec	0.3	72
	0.5	2.85 sec	0.75	21	3 sec	0.31	71
	0.7	2.85 sec	0.74	21	2.88 sec	0.3	72
	0.9	2.88 sec	0.72	23	2.87 sec	0.34	68
GBDT	0.1	7.46 sec	0.95	3	7.19 sec	0.82	**19**
	0.3	7.37 sec	0.95	3	6.96 sec	0.86	15
	0.5	7.35 sec	0.94	4	6.48 sec	0.9	11
	0.7	7.42 sec	0.95	3	5 sec	0.84	16
	0.9	7.29 sec	0.95	3	4.95 sec	0.8	21

On Tabular data type: Table 4.26 shows the experimental results of an adversarial examples generated by ART ZOO attacks for each model using census 1500 original examples. For each models, the performance we obtained is shown in the table. For example, SVM model fooled by 92% and 86.73% in targeted and untargeted attack settings, respectively. Note that, this attack method succeed more on SVM than DT, RF or GBDT models.

Table 4.26: census: Experimental results on 1500 adversarial examples generated by ART ZOO attacks with different values of ε in the targeted and untargeted settings (best attack success rate in boldface).

Model		Targeted					Untargeted				
	eps. (ε)	Pre.	Rec.	F1	MCC	Fooling rate(%)	Pre.	Rec.	F1	MCC	Fooling rate(%)
SVM	0.1	0.094	0.35	0.08	-0.767	**92**	0.229	1	0.229	0	**86.73**
	0.3	0.094	0.35	0.08	-0.767	92	0.229	1	0.229	0	86.73
	0.5	0.094	0.35	0.08	-0.767	92	0.229	1	0.229	0	86.73
	0.7	0.094	0.35	0.08	-0.767	92	0.229	1	0.229	0	86.73
	0.9	0.094	0.35	0.08	-0.767	92	0.229	1	0.229	0	86.73
DT	0.1	0.089	0.21	0.327	-0.361	**67.26**	0.194	0.397	0.485	-0.078	**65.66**
	0.3	0.089	0.21	0.33	-0.358	67	0.195	0.397	0.487	-0.075	65.4
	0.5	0.09	0.21	0.334	-0.353	66.6	0.195	0.394	0.491	-0.073	64.93
	0.7	0.091	0.21	0.343	-0.343	65.73	0.198	0.391	0.499	-0.066	64
	0.9	0.107	0.21	0.421	-0.259	57.93	0.234	0.379	0.574	0.009	55.93
RF	0.1	0.024	0.044	0.382	-0.405	**61.8**	0.181	0.306	0.525	-0.09	**61.46**
	0.3	0.028	0.044	0.435	-0.355	56.46	0.204	0.294	0.575	-0.042	55.86
	0.5	0.028	0.044	0.435	-0.355	56.46	0.202	0.292	0.575	-0.045	55.8
	0.7	0.028	0.044	0.438	-0.353	56.2	0.204	0.292	0.577	-0.042	55.53
	0.9	0.028	0.044	0.439	-0.351	56.06	0.204	0.292	0.579	-0.04	55.4
GBDT	0.1	0.162	0.114	0.663	-0.07	33.73	0.212	0.099	0.71	-0.013	**30.33**
	0.3	0.162	0.114	0.663	-0.07	33.73	0.316	0.146	0.733	0.072	27.93
	0.5	0.154	0.111	0.658	-0.078	34.2	0.254	0.105	0.725	0.019	28.59
	0.7	0.178	0.134	0.661	-0.055	33.93	0.287	0.125	0.729	0.046	28.46
	0.9	0.152	0.111	0.655	-0.082	**34.46**	0.31	0.143	0.731	0.067	28.06

4.4.2 Poisoning attacks in ART

In this section, we report on ART poisoning attack against SVM on census and MNIST data in the targeted settings.

Table 4.27: Parameters for the Poisoning attack algorithms

Attack Algorithms	Parameters
Poisoning Attacks on SVM	ε= 0.3 or ε= 1, ε_{step}=0.1, maximum iteration=10 15 examples (Attack data points on census data) 315 training sets÷180 test sets (For census data) 10 examples (Attack data points on MNIST data) 1169 training sets÷565 test sets (For MNIST data)

ART Poisoning Attack on SVM

We evaluates the ART Poisoning attack against SVM with polynomial kernel on MNIST data and Radial Basis Function (RBF) kernel on census data. Table 4.27 shows the parameter we used. The performance results for the trained clean and poison SVM models are shown in Table 4.28 and Table 4.29.

On Tabular data type: In this experiment, we poisoned the trained SVM model using census data based on procedures in Section 3.2.1. For example, for a model poisoned by $\varepsilon = 0.3$, $\varepsilon_{step} = 0.1$, 15 poison examples, and 10 iterations achieves 20% attack success rate at training time; whereas, with $\varepsilon = 1$ rises to 66.66%. while the model $F1$ score on test data decreases from 0.8 (on clean model) to 0.767 and 0.772 (on poisoned model) as shown in Table 4.28.

Table 4.28: Experimental results on clean and poison SVM model on 180 census original examples (Best result in bold).

Trained SVM model	Precision	Recall	F1 score	MCC
Clean	**0.667**	**0.4**	**0.8**	**0.404**
Poison($\varepsilon = 0.3$)	0.6	0.2	0.767	0.244
Poison($\varepsilon = 1$)	0.667	0.178	0.772	0.257

On Image data type: Now that the data poisoning attack have been discussed in Section 3.2.1. The ART Poisoning attack against MNIST data is evaluated as follow: A trained clean SVM model is poisoned by ART Poisoning on SVM attack method with the $\varepsilon = 0.3$, $\varepsilon_{step} = 0.1$, 10 poison examples, and 10 iterations achieves 70% attack success rate at training time; whereas, with $\varepsilon = 1$ rises to 80%. while the test accuracy of the model decreases from 99.47% (on clean model) to 97.33% and 97.60% (on poisoned model) as shown in Table 4.29.

Now, our assessment of the library summarized by report on the pros and cons of various ART Attacks against our chosen models on tabular and image data

Table 4.29: Experimental results on clean and poison SVM model on 565 MNIST original examples (Best result in bold).

SVM model	Accuracy
Clean	**0.9947**
Poison($\varepsilon = 0.3$)	0.9733
Poison($\varepsilon = 0.1$)	0.9760

in the targeted and untargeted settings experimented are provided in Table 4.30 (100% fooling rate in bold). The meaning of the symbols are as follow: (++) for very effective attack, (+) for effective attack, (−−) for not effective or the attack algorithm is not compatible against the model/ML framework, finally (−) for attacks its success rate is below 15%. The results are based on the parameters used on experiment in Table 4.19 for decision-based attacks, Table 4.11 for gradient-based attacks, Table 4.24 for score-based attack, and Table 4.27 for data poisoning attack.

Table 4.30: Summary: ART Evasion and Poisoning Attacks .

ART Attack Algorithms	Targeted				Untargeted			
	SVM	DT	RF	GBDT	SVM	DT	RF	GBDT
TABULAR DATA TYPE								
FGM(ℓ_1)	++	−−	−−	−−	++	−−	−−	−−
FGM(ℓ_2)	++	−−	−−	−−	++	−−	−−	−−
FGSM(ℓ_∞)	++	−−	−−	−−	++	−−	−−	−−
BIM(ℓ_∞)	++	−−	−−	−−	++	−−	−−	−−
PGD(ℓ_1)	++	−−	−−	−−	++	−−	−−	−−
PGD(ℓ_2)	++	−−	−−	−−	++	−−	−−	−−
PGD(ℓ_∞)	++	−−	−−	−−	++	−−	−−	−−
UP(ℓ_∞)	−−	−−	−−	−−	+	−−	−−	−−
UAP(ℓ_∞)	+	−−	−−	−−	−−	−−	−−	−−
JSMA(ℓ_0)	−−	−−	−−	−−	++	−−	−−	−−
C&W(ℓ_2)	++	−−	−−	−−	++	−−	−−	−−
C&W(ℓ_∞)	++	−−	−−	−−	+	−−	−−	−−
EAD(EN)	−−	−−	−−	−−	++	−−	−−	−−
NewtonFool(ℓ_2)	−−	−−	−−	−−	−−	−−	−−	−−
DeepFool(ℓ_2)	−−	−−	−−	−−	−−	−−	−−	−−
VAT(ℓ_2)	−−	−−	−−	−−	−	−−	−−	−−
HSJA(ℓ_2)	++	++	++	++	++	+	+	+
HSJA(ℓ_∞)	++	++	++	++	++	+	+	+
BA(ℓ_2)	+	++	++	++	−	++	++	++
ZOO(ℓ_2)	++	+	+	−	++	+	+	−
DecisionTree	−−	−−	−−	−−	−−	++	−−	−−
Poisoning Attack on SVM	++	−−	−−	−−	++	−−	−−	−−
IMAGE DATA TYPE								
FGM(ℓ_1)	−−	−−	−−	−−	−−	−−	−−	−−
FGM(ℓ_2)	−−	−−	−−	−−	−−	−−	−−	−−
FGSM(ℓ_∞)	++	−−	−−	−−	++	−−	−−	−−
BIM(ℓ_∞)	−	−−	−−	−−	+	−−	−−	−−
PGD(ℓ_1)	−−	−−	−−	−−	−−	−−	−−	−−
PGD(ℓ_2)	−−	−−	−−	−−	−−	−−	−−	−−
PGD(ℓ_∞)	−	−−	−−	−−	+	−−	−−	−−
UP(ℓ_∞)	−−	−−	−−	−−	+	−−	−−	−−
UAP(ℓ_∞)	+	−−	−−	−−	−−	−−	−−	−−
JSMA(ℓ_0)	−−	−−	−−	−−	++	−−	−−	−−
C&W(ℓ_2)	+	−−	−−	−−	++	−−	−−	−−
C&W(ℓ_∞)	++	−−	−−	−−	++	−−	−−	−−
EAD(EN)	−−	−−	−−	−−	++	−−	−−	−−
NewtonFool(ℓ_2)	−−	−−	−−	−−	−	−−	−−	−−
DeepFool(ℓ_2)	−−	−−	−−	−−	−	−−	−−	−−
VAT(ℓ_2)	−−	−−	−−	−−	−	−−	−−	−−
HSJA(ℓ_2)	+	+	+	+	++	++	++	++
HSJA(ℓ_∞)	+	+	+	+	++	++	++	++
BA(ℓ_2)	+	+	+	+	++	++	++	++
ZOO(ℓ_2)	−−	−	−	−−	−−	++	++	−
DecisionTree	−−	−−	−−	−−	−−	++	−−	−−
Poisoning Attack on SVM	++	−−	−−	−−	++	−−	−−	−−

Chapter 5

Conclusion

Evasion and Poisoning attacks can be a serious threat to traditional ML based systems and applications such as fraud detection. Adversarial Robustness Toolbox (ART) is an open source Python ML security library by IBM, which consists of various framework-agnostic modules of the state-of-the-art attack and defense algorithms. However, considering the attack modules, except Poisoning attack on SVM and Decision Tree attacks implementations, all other attacks implemented in ART are based on DL algorithms and applications such as computer vision and natural language understanding. Hence, our research focused on understanding and evaluating ART library on evasion and poisoning attacks against traditional ML models via adversarial examples of tabular and image data types. Table 3.1 is an experimented list of evasion attacks implemented in ART, but Table 5.1 doesn't.

Concluding from the results and analysis in Section 4.4, We would like to point out that in our experiments, adversarial examples generated by ART are very effective in comparing models test accuracy in adversarial settings. ART Decision Tree attack is very effective against decision trees in untargeted setting only. Most ART gradient-based attacks are very effective on SVM models in tabular data than in image data in both targeted and untargeted settings. ART black-box attack algorithms are very successful to generate adversarial examples on all chosen models and data types, for example, ART HSJA attack successfully attacks all chosen models on tabular and image data in the targeted and untargeted settings. Decision tree is the weakest classifier on image, for instance, BA and Decision tree attack algorithms fool 100% a decision tree classifier on image data, and EAD(EN) attack against SVM model does. Table 4.30 invites us to pursue understanding of these attacks pros and cons against chosen ML models on tabular and image data in the targeted and untargeted settings.

Finally, in this thesis, we showed various ART evasion and poisoning attacks against traditional ML models, which are attack unaware models. As future work, we extend this work to incorporate ART defense mechanisms such as adversarial training methods and evaluate adversarial examples generated by ART evasion and poisoning attacks on resilient ML models.

Bibliography

[1] Missing data imputation using fuzzy-rough methods. *Neurocomputing*, 205:152–164, 2016. ISSN 0925-2312. URL https://www.sciencedirect.com/science/article/pii/S0925231216302582.

[2] M. Andriushchenko, F. Croce, N. Flammarion, and M. Hein. Square attack: a query-efficient black-box adversarial attack via random search, 2020. URL https://arxiv.org/abs/1912.00049.

[3] B. Biggio, B. Nelson, and P. Laskov. Poisoning attacks against support vector machines. abs/1206.6389v1, 2012. URL https://arxiv.org/abs/1206.6389v1.

[4] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli. Evasion attacks against machine learning at test time. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 2429–2432. Springer, 2013.

[5] L. Breiman. Random forests. *Machine Learning*, 45, 5–32, 2001. doi: https://doi.org/10.1023/A:1010933404324.

[6] L. Breiman, J. Friedman, A. O. Richard, and J. S. Charles. *Classification and regression trees*. Wadsworth, 1984. ISBN 0-534-98053-8.

[7] W. Brendel, J. Rauber, and M. Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. *ICLR*, abs/1712.04248, 2018. URL https://arxiv.org/abs/1712.04248.

[8] W. Brendel, J. Rauber, M. Kümmerer, I. Ustyuzhaninov, and M. Bethge. Accurate, reliable and fast robustness evaluation, 2019. URL https://arxiv.org/abs/1907.01003.

[9] T. B. Brown, D. Mané, A. Roy, M. Abadi, and J. Gilmer. Adversarial patch, 2018. URL https://arxiv.org/abs/1712.09665.

[10] N. Carlini and D. Wagner. Towards evaluating the robustness of neural networks. *In IEEE Symposium on Security and Privacy*, abs/1608.04644, 2017. URL http://arxiv.org/abs/1608.04644.

[11] J. Chen and M. I. Jordan. Boundary attack++: Query-efficient decision-based adversarial attack. *CoRR*, abs/1904.02144, 2019. URL https://arxiv.org/abs/1904.02144.

[12] P.-Y. Chen, Y. Sharma, H. Zhang, J. Yi, and C.-J. Hsieh. Ead: Elastic-net attacks to deep neural networks via adversarial examples. *CoRR*, abs/1709.04114, 2017. URL https://arxiv.org/abs/1709.04114.

[13] P.-Y. Chen, H. Zhang, Y. Sharma, J. Yi, and C.-J. Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. *CoRR*, abs/1708.03999, 2017. URL https://arxiv.org/abs/1708.03999.

[14] S.-T. Chen, C. Cornelius, J. Martin, and D. H. Chau. Shapeshifter: Robust physical adversarial attack on faster r-cnn object detector. *Lecture Notes in Computer Science*, abs/1804.05810:52–68, 2019. ISSN 1611-3349. URL https://arxiv.org/abs/1804.05810.

[15] F. Croce and M. Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks, 2020. URL https://arxiv.org/abs/2003.01690.

[16] L. Engstrom, B. Tran, D. Tsipras, L. Schmidt, and A. Madry. Exploring the landscape of spatial robustness, 2019. URL https://arxiv.org/abs/1712.02779.

[17] A. Fernández, S. García, M. Galar, R. C. Prati, B. Krawczyk, and F. Herrera. *Learning from Imbalanced Data Sets*, pages 47–61. Springer International Publishing, 2018.

[18] A. Ghiasi, A. Shafahi, and T. Goldstein. Breaking certified defenses: Semantic adversarial examples with spoofed robustness certificates, 2020. URL https://arxiv.org/abs/2003.08937.

[19] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *CoRR*, abs/1412.6572, 2014. URL https://arxiv.org/abs/1412.6572.

[20] K. Grosse, D. Pfaff, M. T. Smith, and M. Backes. The limitations of model uncertainty in adversarial settings, 2019. URL https://arxiv.org/abs/1812.02606.

[21] T. Gu, B. Dolan-Gavitt, and S. Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain, 2019. URL https://arxiv.org/abs/1708.06733.

[22] C. Guo, J. R. Gardner, Y. You, A. G. Wilson, and K. Q. Weinberger. Simple black-box adversarial attacks, 2019. URL https://arxiv.org/abs/1905.07121.

[23] H. Hirano and K. Takemoto. Simple iterative method for generating targeted universal adversarial perturbations. abs/1911.06502, 2019. URL https://arxiv.org/abs/1911.06502.

[24] N. Inkawhich, M. Inkawhich, Y. Chen, and H. Li. Adversarial attacks for optical flow-based action recognition classifiers, 2018. URL https://arxiv.org/abs/1811.11875.

[25] U. Jang, X. Wu, and S. Jha. Objective metrics and gradient descent algorithms for adversarial examples in machine learning. In *Proceedings of the 33rd Annual Computer Security Applications Conference*, page 262–277. Association for Computing Machinery, 2017. ISBN 9781450353458. URL https://dl.acm.org/doi/10.1145/3134600.3134635.

[26] S. Kotyan and D. V. Vargas. Adversarial robustness assessment: Why both l_0 and l_∞ attacks are necessary, 2020. URL https://arxiv.org/abs/1906.06026.

[27] A. Kurakin, I. J. Goodfellow, and S. Bengio. Adversarial examples in the physical world. *CoRR*, abs/1607.02533, 2016. URL https://arxiv.org/abs/1607.02533.

[28] M. Lee and Z. Kolter. On physical adversarial patches for object detection, 2019. URL https://arxiv.org/abs/1906.11897.

[29] L. Lin, D. Sida, C. Zhiwei, H. Jinghui, J. Shu, and Y. Kunmeng. Using improved gradient-boosted decision tree algorithm based on kalman filter (gbdt-kf) in time series prediction, 2020. URL https://doi.org/10.1007/s11227-019-03130-y.

[30] X. Liu, H. Yang, Z. Liu, L. Song, H. Li, and Y. Chen. Dpatch: An adversarial patch attack on object detectors, 2019. URL https://arxiv.org/abs/1806.02299v4.

[31] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks. *CoRR*, abs/1706.06083, 2017. URL https://arxiv.org/abs/1706.06083.

[32] T. Mitchell. McGraw Hill, 1997. ISBN 0070428077.

[33] T. Miyato, S. ichi Maeda, M. Koyama, K. Nakae, and S. Ishii. Distributional smoothing with virtual adversarial training. abs/1507.00677, 2015. URL https://arxiv.org/abs/1507.00677.

[34] S. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. Deepfool: a simple and accurate method to fool deep neural networks. *CoRR*, abs/1511.04599, 2015. URL `https://arxiv.org/abs/1511.04599`.

[35] S. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard. Universal adversarial perturbations. *CoRR*, abs/1610.08401, 2016. URL `https://arxiv.org/abs/1610.08401`.

[36] S. Murthy. Automatic construction of decision trees from data: A multi-disciplinary survey. *Data Mining and Knowledge Discovery*, 2, 345–389, 1998. doi: https://doi.org/10.1023/A:1009744630224.

[37] M.-I. Nicolae, M. Sinn, M. N. Tran, B. Buesser, A. Rawat, M. Wistuba, V. Zantedeschi, N. Baracaldo, B. Chen, H. Ludwig, I. Molloy, and B. Edwards. Adversarial robustness toolbox v1.2.0. *CoRR*, abs/1807.01069, 2018. URL `https://arxiv.org/abs/1807.01069`.

[38] N. Papernot, P. D. McDaniel, and I. J. Goodfellow. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *CoRR*, abs/1605.07277, 2016. URL `https://arxiv.org/abs/1605.07277`.

[39] N. Papernot, P. D. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami. The limitations of deep learning in adversarial settings. *IEEE*, abs/1511.07528, 2016. URL `https://arxiv.org/abs/1511.07528`.

[40] S. Sabour, Y. Cao, F. Faghri, and D. J. Fleet. Adversarial manipulation of deep representations, 2016. URL `https://arxiv.org/abs/1511.05122`.

[41] A. Shafahi, W. R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein. Poison frogs! targeted clean-label poisoning attacks on neural networks. *CoRR*, abs/1804.00792, 2018. URL `https://arxiv.org/abs/1804.00792`.

[42] J. Su, D. V. Vargas, and K. Sakurai. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, abs/1710.08864, 2019. URL `https://arxiv.org/abs/1710.08864`.

[43] T. J. L. Tan and R. Shokri. Bypassing backdoor detection algorithms in deep learning, 2020. URL `https://arxiv.org/abs/1905.13409`.

[44] V. Vapnik. Principles of risk minimization for learning theory. In J. Moody, S. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems*, volume 4. Morgan-Kaufmann, 1992. URL `https://proceedings.neurips.cc/paper/1991/file/ff4d5fbbafdf976cfdc032e3bde78de5-Paper.pdf`.

[45] E. Wong, F. R. Schmidt, and J. Z. Kolter. Wasserstein adversarial examples via projected sinkhorn iterations, 2020. URL `https://arxiv.org/abs/1902.07906`.

Appendix I: Analysis Results

ART evasion and poison attacks that is not supported to generate adversarial examples on Scikit-learn or lightGBM ML frameworks were excluded from the study see Table 5.1.

Table 5.1: List of ART evasion and poison attacks not evaluated.

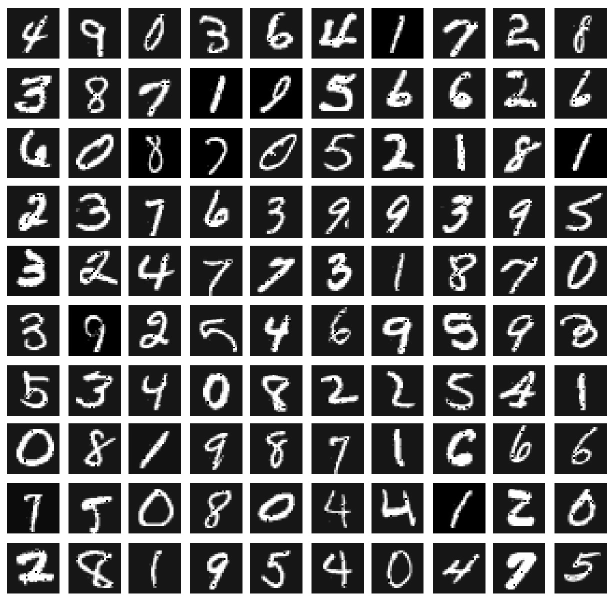
ART Evasion and Poison Attacks		
No.	**Attack Name**	**Error message**
1	AutoAttack(Croce and Hein, 2020)	The loss type cross_entropy is not supported for the provided estimator.
2	Auto Projected Gradient Descent Attack (Croce and Hein, 2020)	AutoProjectedGradientDescent is expecting logits as estimator output, the provided estimator seems to predict probabilities.
3	Square Attack (Andriushchenko et al., 2020)	Unrecognized input dimension.
4	Threshold Attack (Kotyan and Vargas, 2020)	ThresholdAttack requires an estimator derived from class 'art.estimators.estimator.NeuralNetworkMixin'
5	Adversarial Embedding attack (Tan and Shokri, 2020)	-
6	Shadow Attack (Ghiasi et al., 2020)	Unrecognized input dimension. Shadow Attack can only be applied to image data.
7	Brendel & Bethge adversarial attack (Brendel et al., 2019)	The provided classifier is an instance of class 'NoneType'
8	High Confidence Low Uncertainty (HCLU) Attack(Grosse et al., 2019)	HighConfidenceLowUncertainty requires an estimator derived from class 'art.estimators.classification.GPy. GPyGaussianProcessClassifier'
9	Pixel Attack(Su et al., 2019)	PixelAttack requires an estimator derived from class 'art.estimators.estimator.NeuralNetworkMixin'
10	Spatial transformation attack (Engstrom et al., 2019)	SpatialTransformation requires an estimator derived from class 'art.estimators.estimator. NeuralNetworkMixin'
11	Simple Black-box Adversarial (SimBA) (Guo et al., 2019)	'ScikitlearnSVC' object has no attribute 'channels_first'
12	Robust DPatch attack (Lee and Kolter, 2019)	RobustDPatch requires an estimator derived from class 'art.estimators.object_detection.object_detector. ObjectDetectorMixin'
13	ShapeShifter attack (Chen et al., 2019)	ShapeShifter requires an estimator derived from class 'art.estimators.object_detection.tensorflow_faster_rcnn. TensorFlowFasterRCNN'

14	Adversarial patch attack ‘DPatch‘ (Liu et al., 2019)	DPatch requires an estimator derived from class ’art.estimators.object_detection.object_detector. ObjectDetectorMixin’
15	Clean-Label Backdoor Attacks (Gu et al., 2019)	-
16	Backdoor Attacks (Gu et al., 2019)	-
17	Wasserstein attack (Wong et al., 2020)	ScikitlearnSVC’ object has no attribute ’channels_first’
18	Frame saliency attack (Inkawhich et al., 2018)	FrameSaliencyAttack requires an estimator derived from class ’art.estimators.estimator.NeuralNetworkMixin’
19	Feature Collision Poisoning Attack (Shafahi et al., 2018)	-
20	Adversarial Patch attack (Brown et al., 2018)	AdversarialPatch requires an estimator derived from class ’art.estimators.estimator.NeuralNetworkMixin’
21	Feature Adversaries attack (Sabour et al., 2016)	FeatureAdversaries requires an estimator derived from class ’art.estimators.estimator.NeuralNetworkMixin’

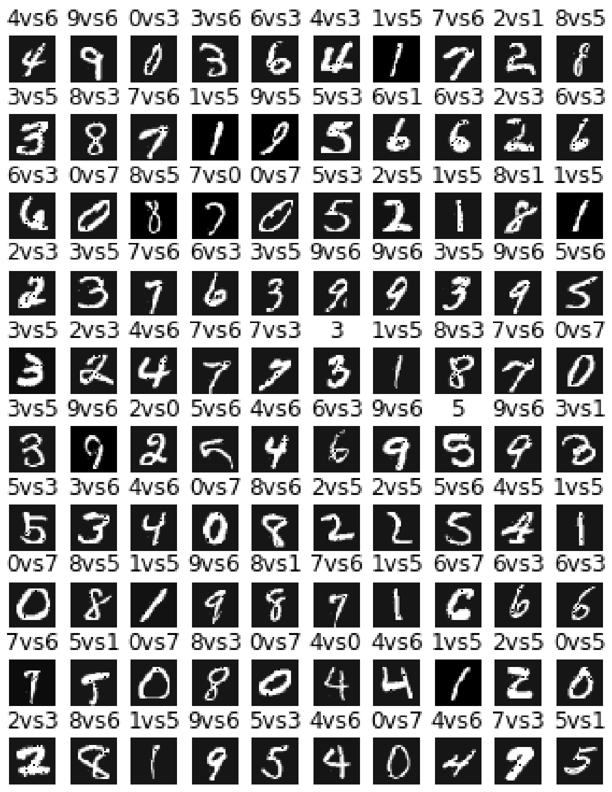
Appendix II: Experimental Results

1 MNIST: Adversarial examples generated by ART DecisionTree attack against Decision Trees in the untargeted setting.

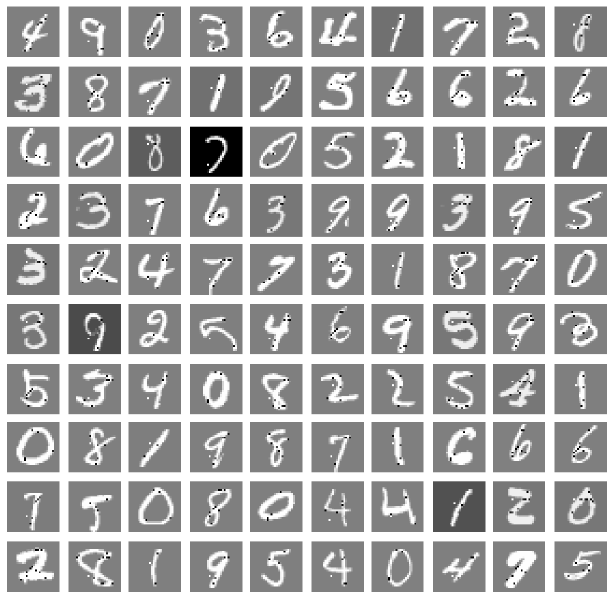
offset=0.1 (With out label)



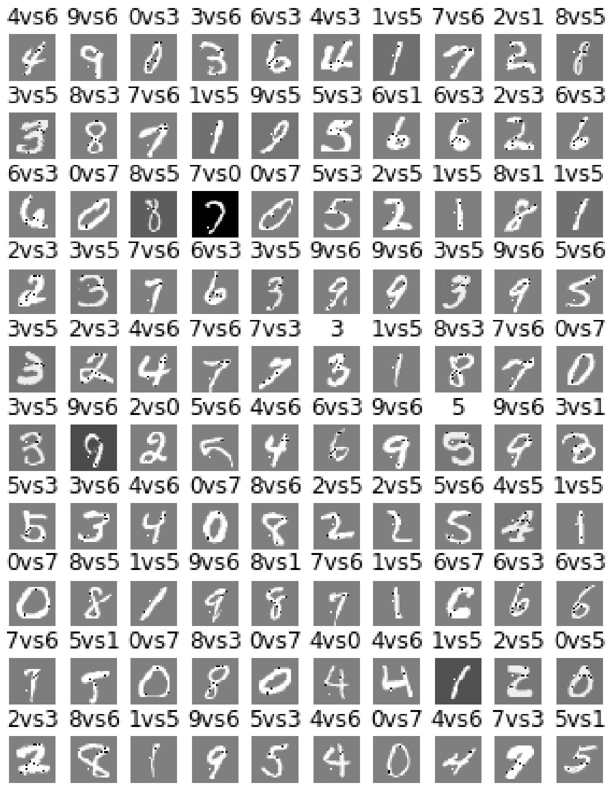
(With label)


4vs6 9vs6 0vs3 3vs6 6vs3 4vs3 1vs5 7vs6 2vs1 8vs5
3vs5 8vs3 7vs6 1vs5 9vs5 5vs3 6vs1 6vs3 2vs3 6vs3
6vs3 0vs7 8vs5 7vs0 0vs7 5vs3 2vs5 1vs5 8vs1 1vs5
2vs3 3vs5 7vs6 6vs3 3vs5 9vs6 9vs6 3vs5 9vs6 5vs6
3vs5 2vs3 4vs6 7vs6 7vs3 3 1vs5 8vs3 7vs6 0vs7
3vs5 9vs6 2vs0 5vs6 4vs6 6vs3 9vs6 5 9vs6 3vs1
5vs3 3vs6 4vs6 0vs7 8vs6 2vs5 2vs5 5vs6 4vs5 1vs5
0vs7 8vs5 1vs5 9vs6 8vs1 7vs6 1vs5 6vs7 6vs3 6vs3
7vs6 5vs1 0vs7 8vs3 0vs7 4vs0 4vs6 1vs5 2vs5 0vs5
2vs3 8vs6 1vs5 9vs6 5vs3 4vs6 0vs7 4vs6 7vs3 5vs1

offset=1 (With out label)

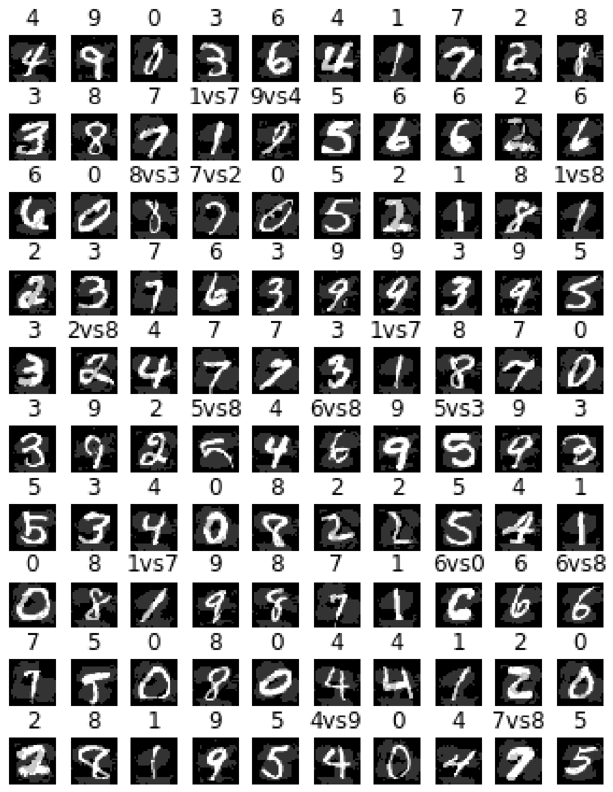


(With label)

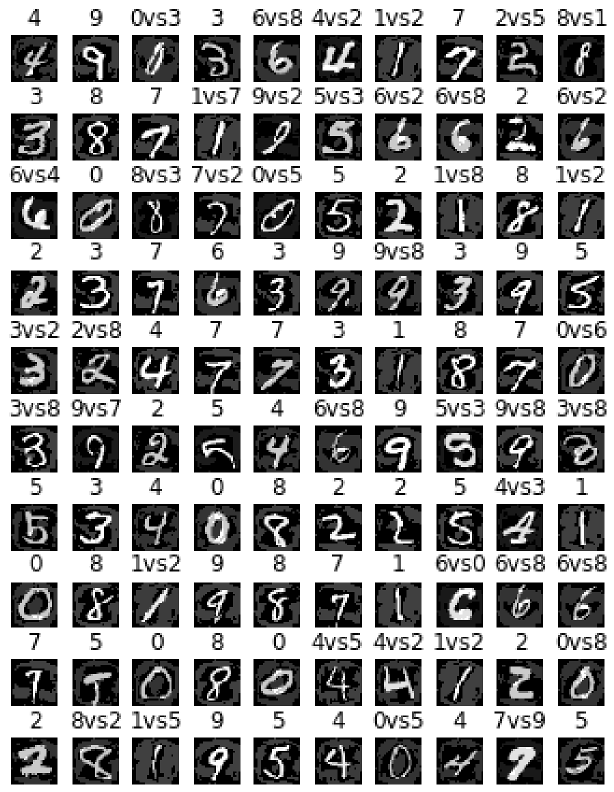

4vs6 9vs6 0vs3 3vs6 6vs3 4vs3 1vs5 7vs6 2vs1 8vs5
3vs5 8vs3 7vs6 1vs5 9vs5 5vs3 6vs1 6vs3 2vs3 6vs3
6vs3 0vs7 8vs5 7vs0 0vs7 5vs3 2vs5 1vs5 8vs1 1vs5
2vs3 3vs5 7vs6 6vs3 3vs5 9vs6 9vs6 3vs5 9vs6 5vs6
3vs5 2vs3 4vs6 7vs6 7vs3 3 1vs5 8vs3 7vs6 0vs7
3vs5 9vs6 2vs0 5vs6 4vs6 6vs3 9vs6 5 9vs6 3vs1
5vs3 3vs6 4vs6 0vs7 8vs6 2vs5 2vs5 5vs6 4vs5 1vs5
0vs7 8vs5 1vs5 9vs6 8vs1 7vs6 1vs5 6vs7 6vs3 6vs3
7vs6 5vs1 0vs7 8vs3 0vs7 4vs0 4vs6 1vs5 2vs5 0vs5
2vs3 8vs6 1vs5 9vs6 5vs3 4vs6 0vs7 4vs6 7vs3 5vs1

2.1 MNIST: Adversarial examples generated by ART FGSM(ℓ_∞) attack against SVM in the targeted and untargeted settings.

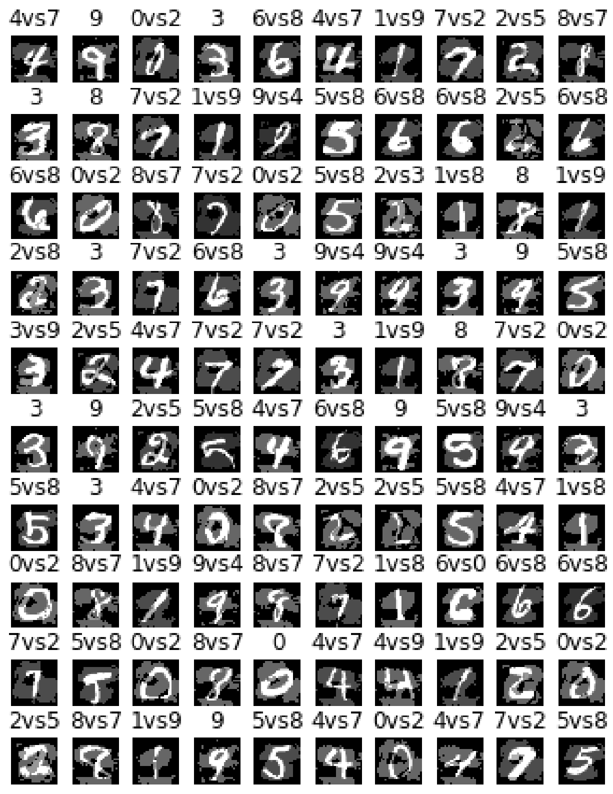
(Targeted, $\epsilon = 0.3$)


4 9 0 3 6 4 1 7 2 8
3 8 7 1vs7 9vs4 5 6 6 2 6
6 0 8vs3 7vs2 0 5 2 1 8 1vs8
2 3 7 6 3 9 9 3 9 5
3 2vs8 4 7 7 3 1vs7 8 7 0
3 9 2 5vs8 4 6vs8 9 5vs3 9 3
5 3 4 0 8 2 2 5 4 1
0 8 1vs7 9 8 7 1 6vs0 6 6vs8
7 5 0 8 0 4 4 1 2 0
2 8 1 9 5 4vs9 0 4 7vs8 5

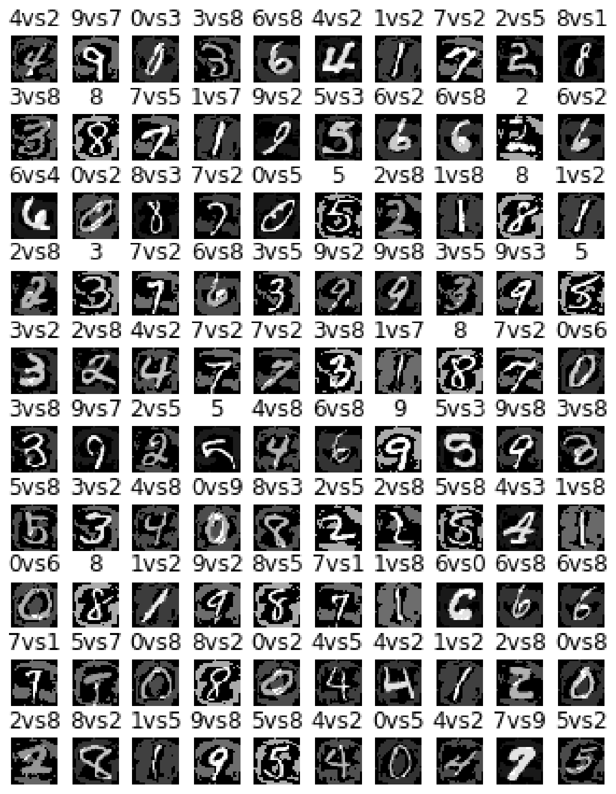
(Untargeted, $\epsilon = 0.3$)


4 9 0vs3 3 6vs8 4vs2 1vs2 7 2vs5 8vs1
3 8 7 1vs7 9vs2 5vs3 6vs2 6vs8 2 6vs2
6vs4 0 8vs3 7vs2 0vs5 5 2 1vs8 8 1vs2
2 3 7 6 3 9 9vs8 3 9 5
3vs2 2vs8 4 7 7 3 1 8 7 0vs6
3vs8 9vs7 2 5 4 6vs8 9 5vs3 9vs8 3vs8
5 3 4 0 8 2 2 5 4vs3 1
0 8 1vs2 9 8 7 1 6vs0 6vs8 6vs8
7 5 0 8 0 4vs5 4vs2 1vs2 2 0vs8
2 8vs2 1vs5 9 5 4 0vs5 4 7vs9 5

(Targeted, $\epsilon = 0.4$)

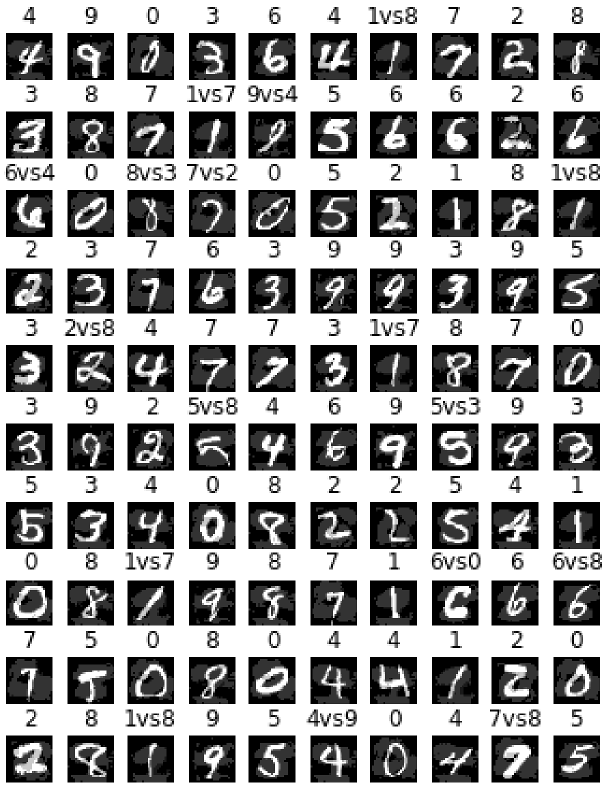

4vs7 9 0vs2 3 6vs8 4vs7 1vs9 7vs2 2vs5 8vs7
3 8 7vs2 1vs9 9vs4 5vs8 6vs8 6vs8 2vs5 6vs8
6vs8 0vs2 8vs7 7vs2 0vs2 5vs8 2vs3 1vs8 8 1vs9
2vs8 3 7vs2 6vs8 3 9vs4 9vs4 3 9 5vs8
3vs9 2vs5 4vs7 7vs2 7vs2 3 1vs9 8 7vs2 0vs2
3 9 2vs5 5vs8 4vs7 6vs8 9 5vs8 9vs4 3
5vs8 3 4vs7 0vs2 8vs7 2vs5 2vs5 5vs8 4vs7 1vs8
0vs2 8vs7 1vs9 9vs4 8vs7 7vs2 1vs8 6vs0 6vs8 6vs8
7vs2 5vs8 0vs2 8vs7 0 4vs7 4vs9 1vs9 2vs5 0vs2
2vs5 8vs7 1vs9 9 5vs8 4vs7 0vs2 4vs7 7vs2 5vs8

(Untargeted, $\epsilon = 0.4$)

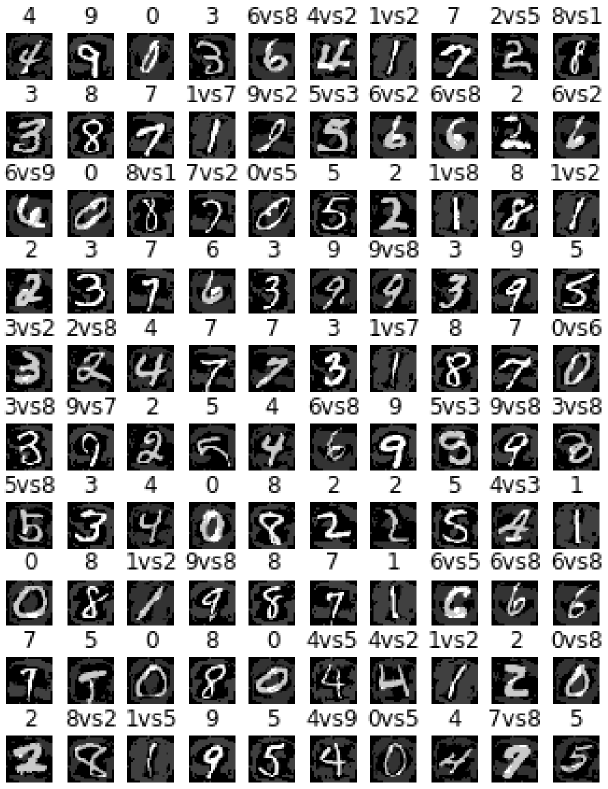

4vs2 9vs7 0vs3 3vs8 6vs8 4vs2 1vs2 7vs2 2vs5 8vs1
3vs8 8 7vs5 1vs7 9vs2 5vs3 6vs2 6vs8 2 6vs2
6vs4 0vs2 8vs3 7vs2 0vs5 5 2vs8 1vs8 8 1vs2
2vs8 3 7vs2 6vs8 3vs5 9vs2 9vs8 3vs5 9vs3 5
3vs2 2vs8 4vs2 7vs2 7vs2 3vs8 1vs7 8 7vs2 0vs6
3vs8 9vs7 2vs5 5 4vs8 6vs8 9 5vs3 9vs8 3vs8
5vs8 3vs2 4vs8 0vs9 8vs3 2vs5 2vs8 5vs8 4vs3 1vs8
0vs6 8 1vs2 9vs2 8vs5 7vs1 1vs8 6vs0 6vs8 6vs8
7vs1 5vs7 0vs8 8vs2 0vs2 4vs5 4vs2 1vs2 2vs8 0vs8
2vs8 8vs2 1vs5 9vs8 5vs8 4vs2 0vs5 4vs2 7vs9 5vs2

2.2 MNIST: Adversarial examples generated by ART BIM(ℓ_∞) attack against SVM in the targeted and untargeted settings.

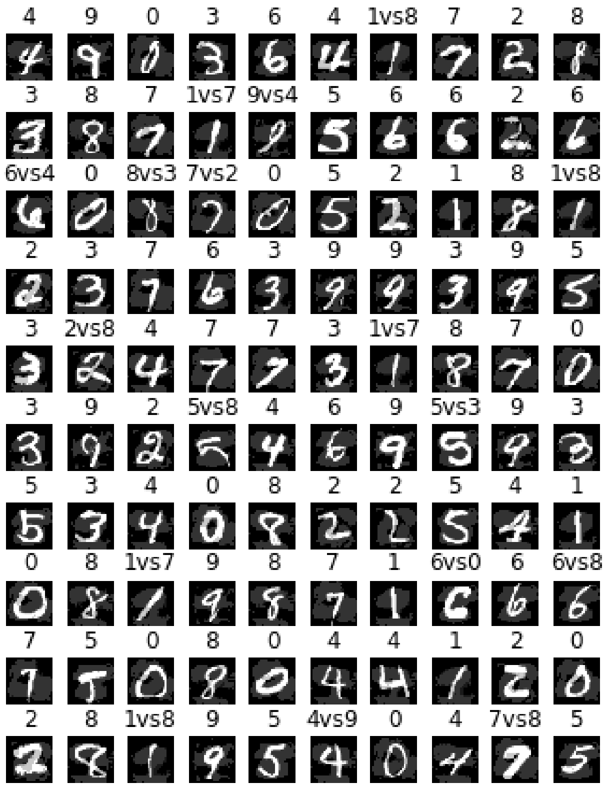
(Targeted, $\epsilon = 0.3$)


4 9 0 3 6 4 1vs8 7 2 8
3 8 7 1vs7 9vs4 5 6 6 2 6
6vs4 0 8vs3 7vs2 0 5 2 1 8 1vs8
2 3 7 6 3 9 9 3 9 5
3 2vs8 4 7 7 3 1vs7 8 7 0
3 9 2 5vs8 4 6 9 5vs3 9 3
5 3 4 0 8 2 2 5 4 1
0 8 1vs7 9 8 7 1 6vs0 6 6vs8
7 5 0 8 0 4 4 1 2 0
2 8 1vs8 9 5 4vs9 0 4 7vs8 5

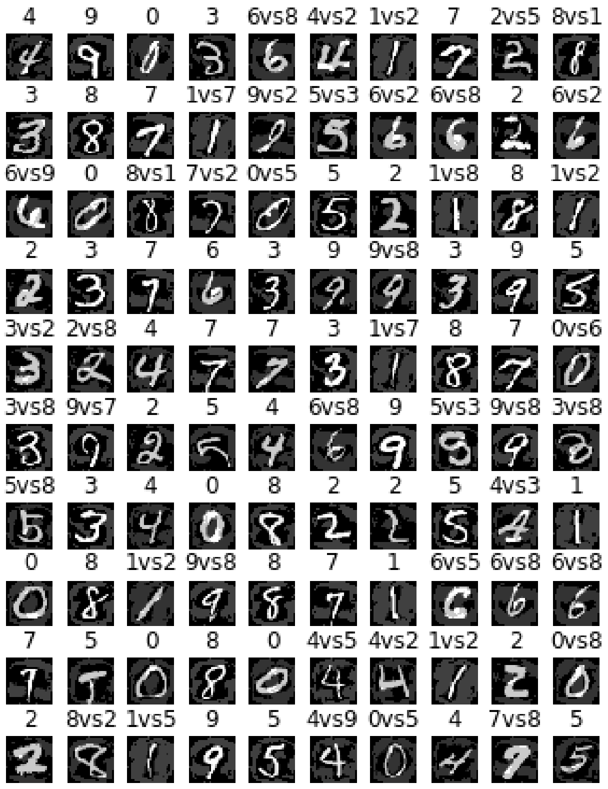
(Untargeted, $\epsilon = 0.3$)


4 9 0 3 6vs8 4vs2 1vs2 7 2vs5 8vs1
3 8 7 1vs7 9vs2 5vs3 6vs2 6vs8 2 6vs2
6vs9 0 8vs1 7vs2 0vs5 5 2 1vs8 8 1vs2
2 3 7 6 3 9 9vs8 3 9 5
3vs2 2vs8 4 7 7 3 1vs7 8 7 0vs6
3vs8 9vs7 2 5 4 6vs8 9 5vs3 9vs8 3vs8
5vs8 3 4 0 8 2 2 5 4vs3 1
0 8 1vs2 9vs8 8 7 1 6vs5 6vs8 6vs8
7 5 0 8 0 4vs5 4vs2 1vs2 2 0vs8
2 8vs2 1vs5 9 5 4vs9 0vs5 4 7vs8 5

(Targeted, $\epsilon = 0.4$)

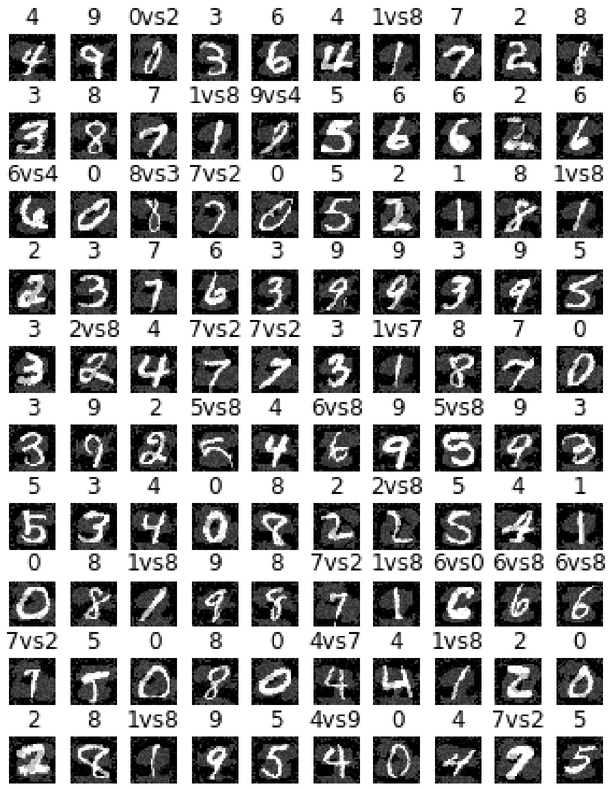

4 9 0 3 6 4 1vs8 7 2 8
3 8 7 1vs7 9vs4 5 6 6 2 6
6vs4 0 8vs3 7vs2 0 5 2 1 8 1vs8
2 3 7 6 3 9 9 3 9 5
3 2vs8 4 7 7 3 1vs7 8 7 0
3 9 2 5vs8 4 6 9 5vs3 9 3
5 3 4 0 8 2 2 5 4 1
0 8 1vs7 9 8 7 1 6vs0 6 6vs8
7 5 0 8 0 4 4 1 2 0
2 8 1vs8 9 5 4vs9 0 4 7vs8 5

(Untargeted, $\epsilon = 0.4$)

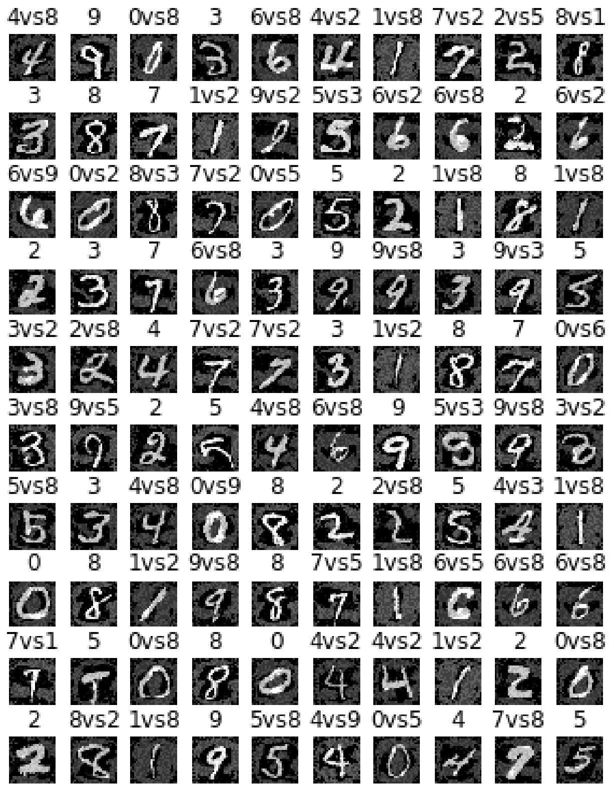

4 9 0 3 6vs8 4vs2 1vs2 7 2vs5 8vs1
3 8 7 1vs7 9vs2 5vs3 6vs2 6vs8 2 6vs2
6vs9 0 8vs1 7vs2 0vs5 5 2 1vs8 8 1vs2
2 3 7 6 3 9 9vs8 3 9 5
3vs2 2vs8 4 7 7 3 1vs7 8 7 0vs6
3vs8 9vs7 2 5 4 6vs8 9 5vs3 9vs8 3vs8
5vs8 3 4 0 8 2 2 5 4vs3 1
0 8 1vs2 9vs8 8 7 1 6vs5 6vs8 6vs8
7 5 0 8 0 4vs5 4vs2 1vs2 2 0vs8
2 8vs2 1vs5 9 5 4vs9 0vs5 4 7vs8 5

2.3 MNIST: Adversarial examples generated by ART PGD(ℓ_∞) attack against SVM in the targeted and untargeted settings.

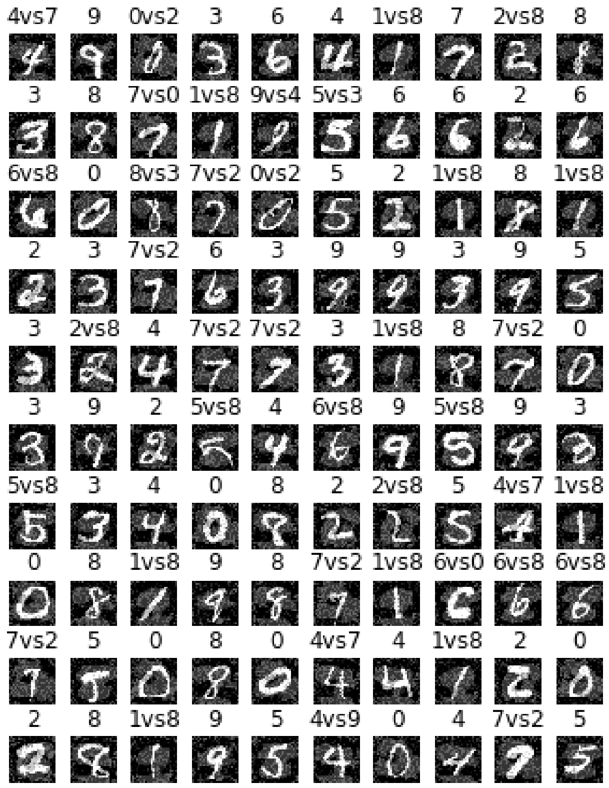
(Targeted, $\epsilon = 0.3$)


4 9 0vs2 3 6 4 1vs8 7 2 8
3 8 7 1vs8 9vs4 5 6 6 2 6
6vs4 0 8vs3 7vs2 0 5 2 1 8 1vs8
2 3 7 6 3 9 9 3 9 5
3 2vs8 4 7vs2 7vs2 3 1vs7 8 7 0
3 9 2 5vs8 4 6vs8 9 5vs8 9 3
5 3 4 0 8 2 2vs8 5 4 1
0 8 1vs8 9 8 7vs2 1vs8 6vs0 6vs8 6vs8
7vs2 5 0 8 0 4vs7 4 1vs8 2 0
2 8 1vs8 9 5 4vs9 0 4 7vs2 5

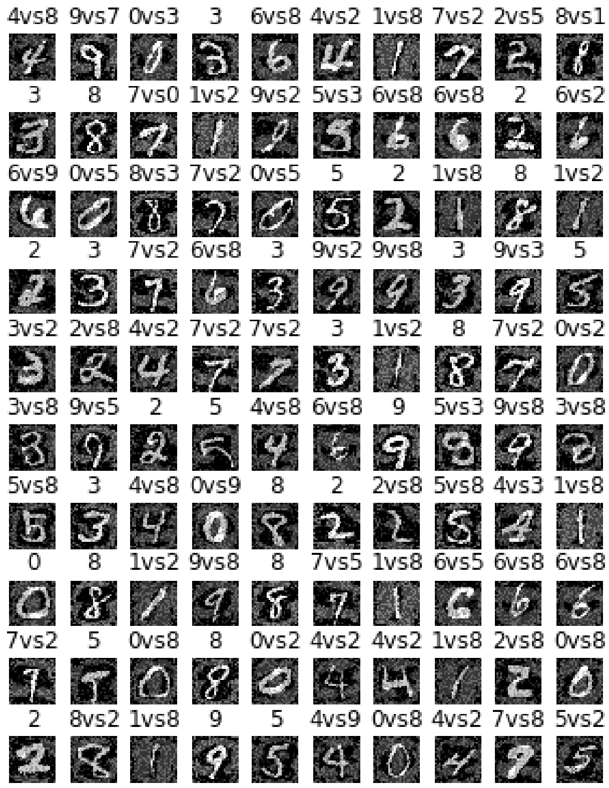
(Untargeted, $\epsilon = 0.3$)


4vs8 9 0vs8 3 6vs8 4vs2 1vs8 7vs2 2vs5 8vs1
3 8 7 1vs2 9vs2 5vs3 6vs2 6vs8 2 6vs2
6vs9 0vs2 8vs3 7vs2 0vs5 5 2 1vs8 8 1vs8
2 3 7 6vs8 3 9 9vs8 3 9vs3 5
3vs2 2vs8 4 7vs2 7vs2 3 1vs2 8 7 0vs6
3vs8 9vs5 2 5 4vs8 6vs8 9 5vs3 9vs8 3vs2
5vs8 3 4vs8 0vs9 8 2 2vs8 5 4vs3 1vs8
0 8 1vs2 9vs8 8 7vs5 1vs8 6vs5 6vs8 6vs8
7vs1 5 0vs8 8 0 4vs2 4vs2 1vs2 2 0vs8
2 8vs2 1vs8 9 5vs8 4vs9 0vs5 4 7vs8 5

(Targeted, $\epsilon = 0.4$)

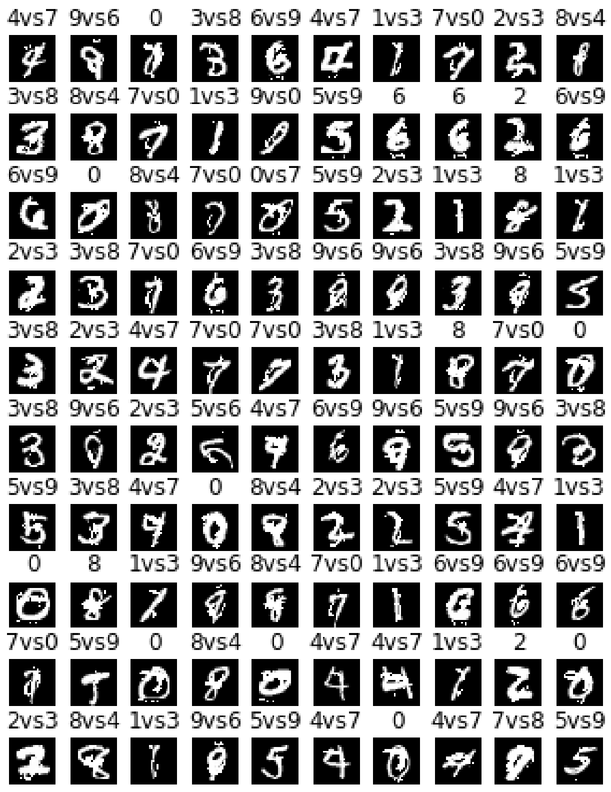

4vs7 9 0vs2 3 6 4 1vs8 7 2vs8 8
3 8 7vs0 1vs8 9vs4 5vs3 6 6 2 6
6vs8 0 8vs3 7vs2 0vs2 5 2 1vs8 8 1vs8
2 3 7vs2 6 3 9 9 3 9 5
3 2vs8 4 7vs2 7vs2 3 1vs8 8 7vs2 0
3 9 2 5vs8 4 6vs8 9 5vs8 9 3
5vs8 3 4 0 8 2 2vs8 5 4vs7 1vs8
0 8 1vs8 9 8 7vs2 1vs8 6vs0 6vs8 6vs8
7vs2 5 0 8 0 4vs7 4 1vs8 2 0
2 8 1vs8 9 5 4vs9 0 4 7vs2 5

(Untargeted, $\epsilon = 0.4$)

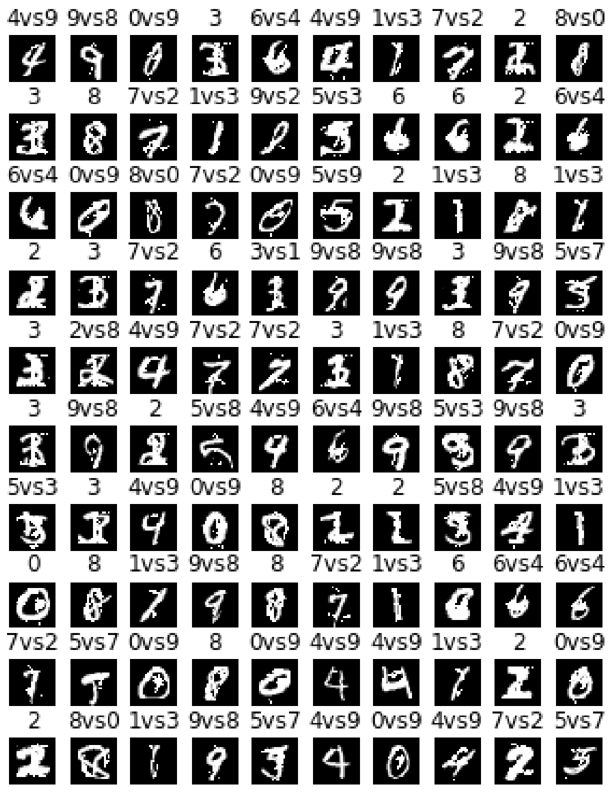

4vs8 9vs7 0vs3 3 6vs8 4vs2 1vs8 7vs2 2vs5 8vs1
3 8 7vs0 1vs2 9vs2 5vs3 6vs8 6vs8 2 6vs2
6vs9 0vs5 8vs3 7vs2 0vs5 5 2 1vs8 8 1vs2
2 3 7vs2 6vs8 3 9vs2 9vs8 3 9vs3 5
3vs2 2vs8 4vs2 7vs2 7vs2 3 1vs2 8 7vs2 0vs2
3vs8 9vs5 2 5 4vs8 6vs8 9 5vs3 9vs8 3vs8
5vs8 3 4vs8 0vs9 8 2 2vs8 5vs8 4vs3 1vs8
0 8 1vs2 9vs8 8 7vs5 1vs8 6vs5 6vs8 6vs8
7vs2 5 0vs8 8 0vs2 4vs2 4vs2 1vs8 2vs8 0vs8
2 8vs2 1vs8 9 5 4vs9 0vs8 4vs2 7vs8 5vs2

2.4 MNIST: Adversarial examples generated by ART JSMA(ℓ_0) attack against SVM in the untargeted settings.

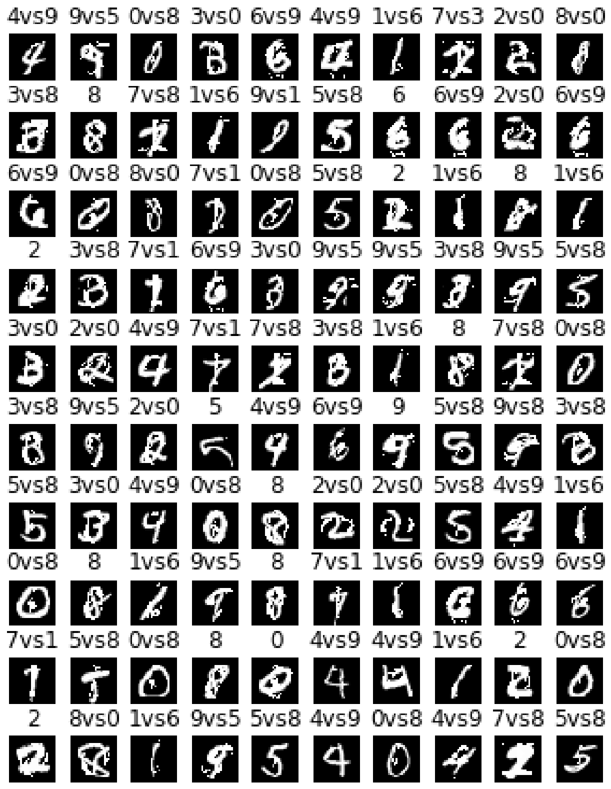
($\theta = 0.1, \gamma = 0.1$, Fooling rate(84%))


4vs7 9vs6 0 3vs8 6vs9 4vs7 1vs3 7vs0 2vs3 8vs4
3vs8 8vs4 7vs0 1vs3 9vs0 5vs9 6 6 2 6vs9
6vs9 0 8vs4 7vs0 0vs7 5vs9 2vs3 1vs3 8 1vs3
2vs3 3vs8 7vs0 6vs9 3vs8 9vs6 9vs6 3vs8 9vs6 5vs9
3vs8 2vs3 4vs7 7vs0 7vs0 3vs8 1vs3 8 7vs0 0
3vs8 9vs6 2vs3 5vs6 4vs7 6vs9 9vs6 5vs9 9vs6 3vs8
5vs9 3vs8 4vs7 0 8vs4 2vs3 2vs3 5vs9 4vs7 1vs3
0 8 1vs3 9vs6 8vs4 7vs0 1vs3 6vs9 6vs9 6vs9
7vs0 5vs9 0 8vs4 0 4vs7 4vs7 1vs3 2 0
2vs3 8vs4 1vs3 9vs6 5vs9 4vs7 0 4vs7 7vs8 5vs9

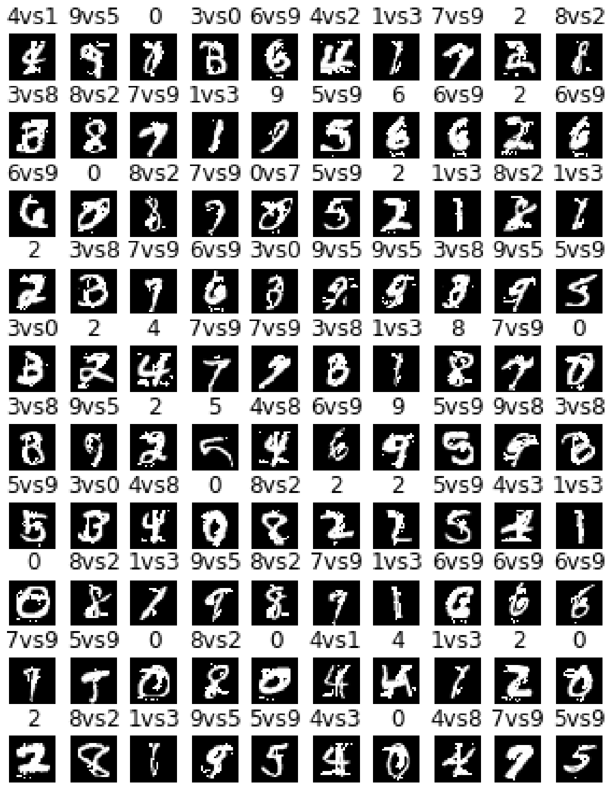
($\theta = 0.3, \gamma = 0.1$, Fooling rate(70%))


4vs9 9vs8 0vs9 3 6vs4 4vs9 1vs3 7vs2 2 8vs0
3 8 7vs2 1vs3 9vs2 5vs3 6 6 2 6vs4
6vs4 0vs9 8vs0 7vs2 0vs9 5vs9 2 1vs3 8 1vs3
2 3 7vs2 6 3vs1 9vs8 9vs8 3 9vs8 5vs7
3 2vs8 4vs9 7vs2 7vs2 3 1vs3 8 7vs2 0vs9
3 9vs8 2 5vs8 4vs9 6vs4 9vs8 5vs3 9vs8 3
5vs3 3 4vs9 0vs9 8 2 2 5vs8 4vs9 1vs3
0 8 1vs3 9vs8 8 7vs2 1vs3 6 6vs4 6vs4
7vs2 5vs7 0vs9 8 0vs9 4vs9 4vs9 1vs3 2 0vs9
2 8vs0 1vs3 9vs8 5vs7 4vs9 0vs9 4vs9 7vs2 5vs7

($\theta = 0.7, \gamma = 0.1$, Fooling rate(86%))

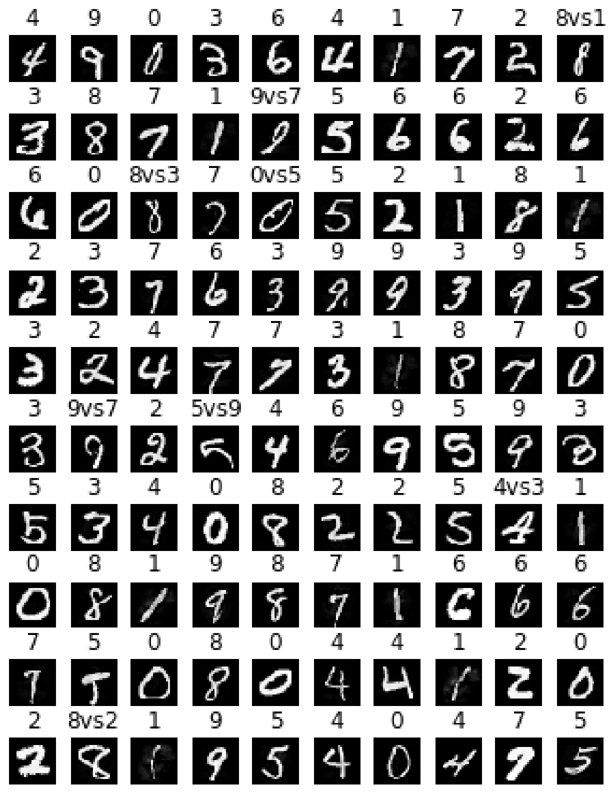

4vs9 9vs5 0vs8 3vs0 6vs9 4vs9 1vs6 7vs3 2vs0 8vs0
3vs8 8 7vs8 1vs6 9vs1 5vs8 6 6vs9 2vs0 6vs9
6vs9 0vs8 8vs0 7vs1 0vs8 5vs8 2 1vs6 8 1vs6
2 3vs8 7vs1 6vs9 3vs0 9vs5 9vs5 3vs8 9vs5 5vs8
3vs0 2vs0 4vs9 7vs1 7vs8 3vs8 1vs6 8 7vs8 0vs8
3vs8 9vs5 2vs0 5 4vs9 6vs9 9 5vs8 9vs8 3vs8
5vs8 3vs0 4vs9 0vs8 8 2vs0 2vs0 5vs8 4vs9 1vs6
0vs8 8 1vs6 9vs5 8 7vs1 1vs6 6vs9 6vs9 6vs9
7vs1 5vs8 0vs8 8 0 4vs9 4vs9 1vs6 2 0vs8
2 8vs0 1vs6 9vs5 5vs8 4vs9 0vs8 4vs9 7vs8 5vs8

($\theta = 0.9, \gamma = 0.1$, Fooling rate(76%))

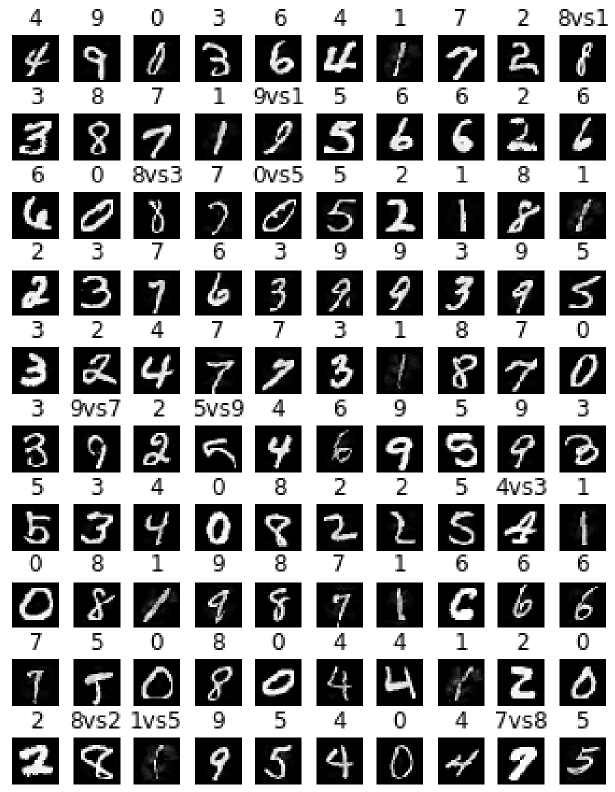

4vs1 9vs5 0 3vs0 6vs9 4vs2 1vs3 7vs9 2 8vs2
3vs8 8vs2 7vs9 1vs3 9 5vs9 6 6vs9 2 6vs9
6vs9 0 8vs2 7vs9 0vs7 5vs9 2 1vs3 8vs2 1vs3
2 3vs8 7vs9 6vs9 3vs0 9vs5 9vs5 3vs8 9vs5 5vs9
3vs0 2 4 7vs9 7vs9 3vs8 1vs3 8 7vs9 0
3vs8 9vs5 2 5 4vs8 6vs9 9 5vs9 9vs8 3vs8
5vs9 3vs0 4vs8 0 8vs2 2 2 5vs9 4vs3 1vs3
0 8vs2 1vs3 9vs5 8vs2 7vs9 1vs3 6vs9 6vs9 6vs9
7vs9 5vs9 0 8vs2 0 4vs1 4 1vs3 2 0
2 8vs2 1vs3 9vs5 5vs9 4vs3 0 4vs8 7vs9 5vs9

2.5 MNIST: Adversarial examples generated by ART DeepFool(ℓ_2) attack against SVM in the untargeted settings.

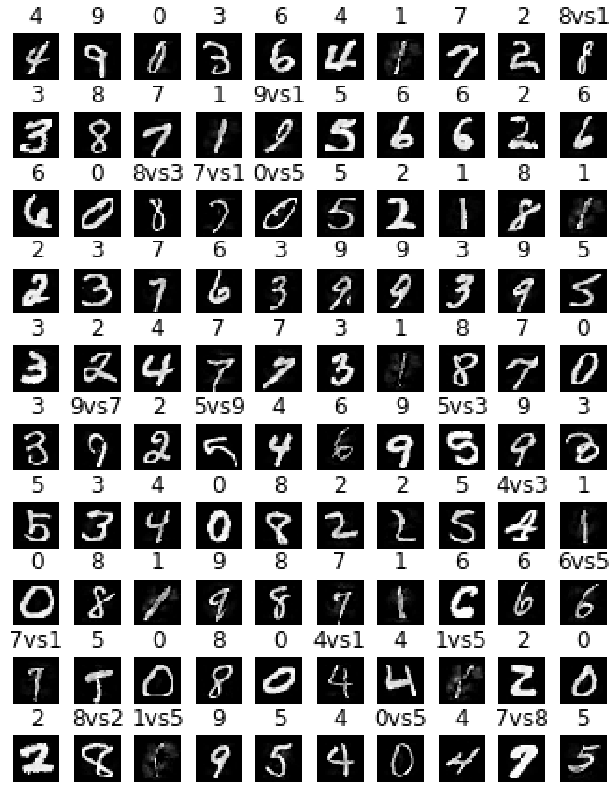
($\epsilon = 0.1$, Fooling rate(8%))



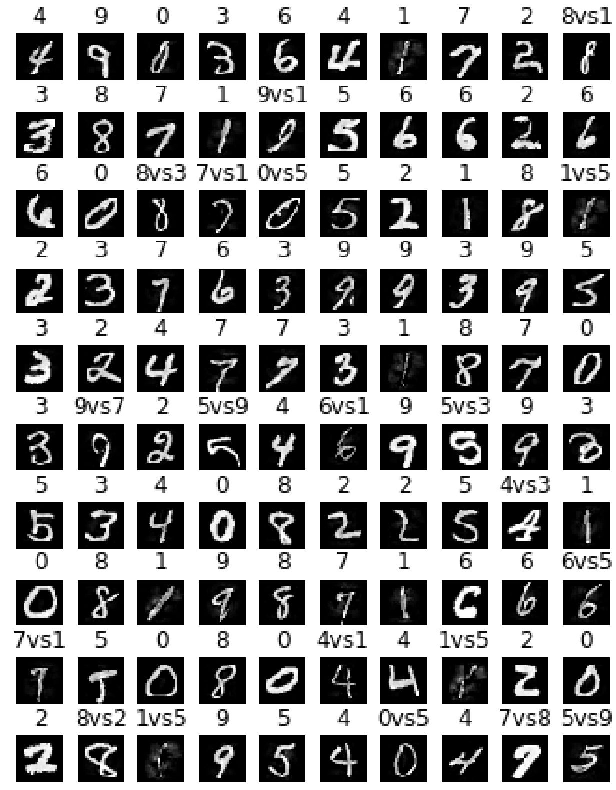
($\epsilon = 0.3$, Fooling rate(11%))



($\epsilon = 0.7$, Fooling rate(18%))

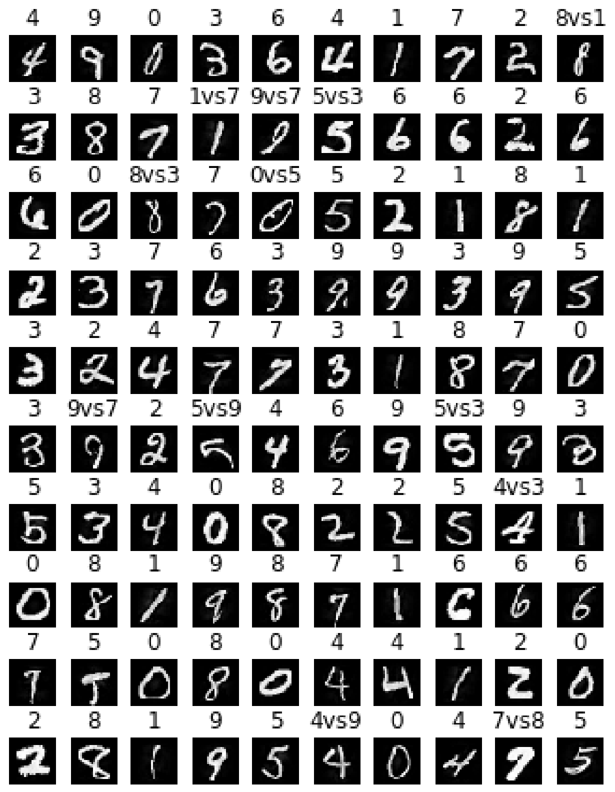


($\epsilon = 0.9$, Fooling rate(21%))

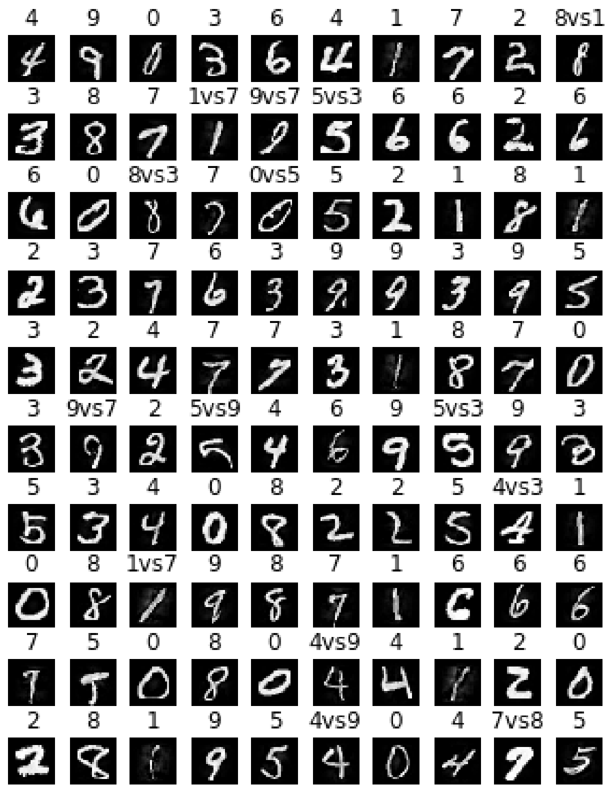


2.6 MNIST: Adversarial examples generated by ART NewtonFool(ℓ_2) attack against SVM in the untargeted settings.

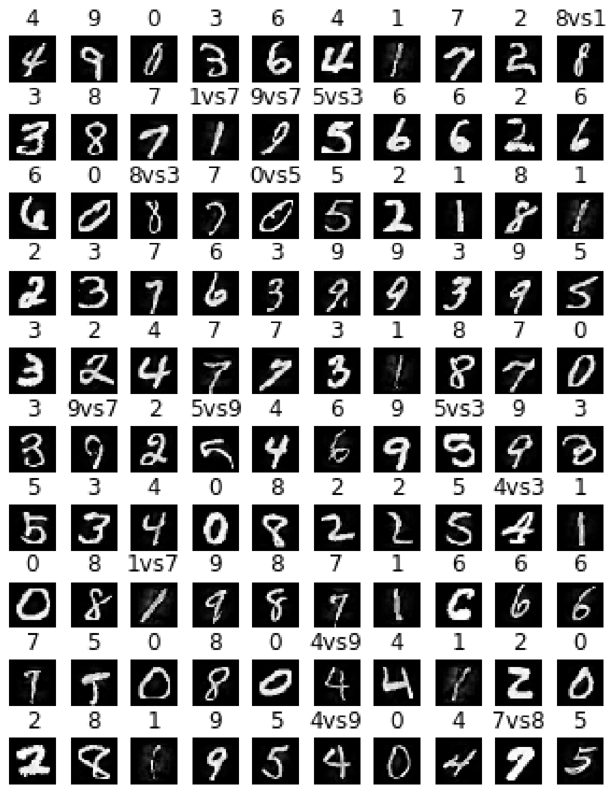
($\eta = 0.1$, Fooling rate(8%))



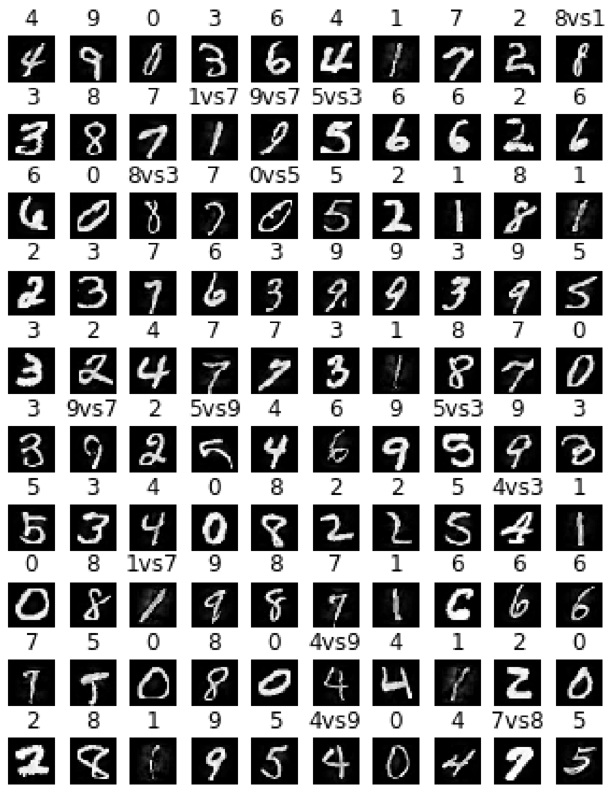
($\eta = 0.3$, Fooling rate(14%))



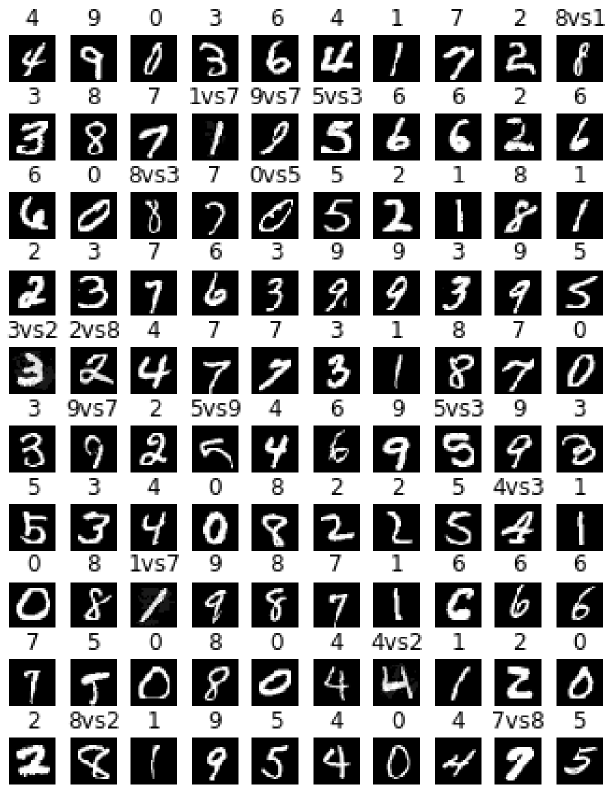
($\eta = 0.7$, Fooling rate(14%))

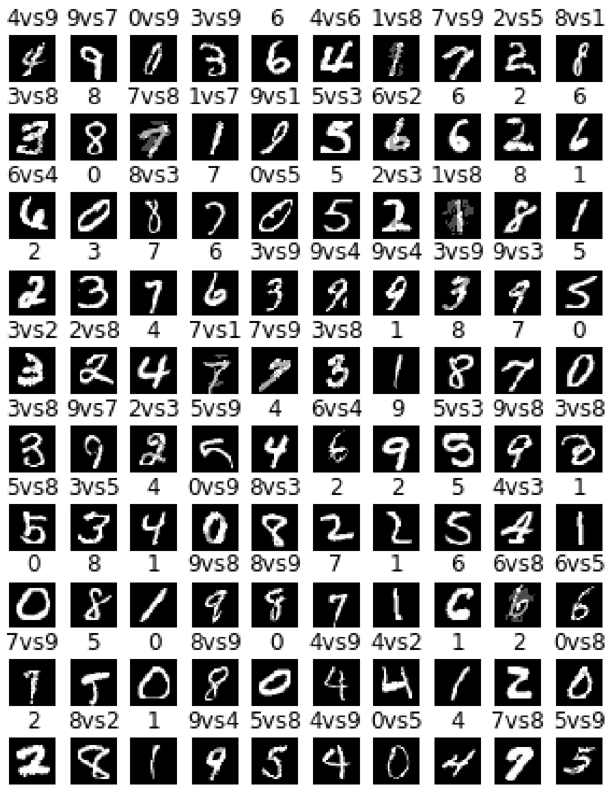


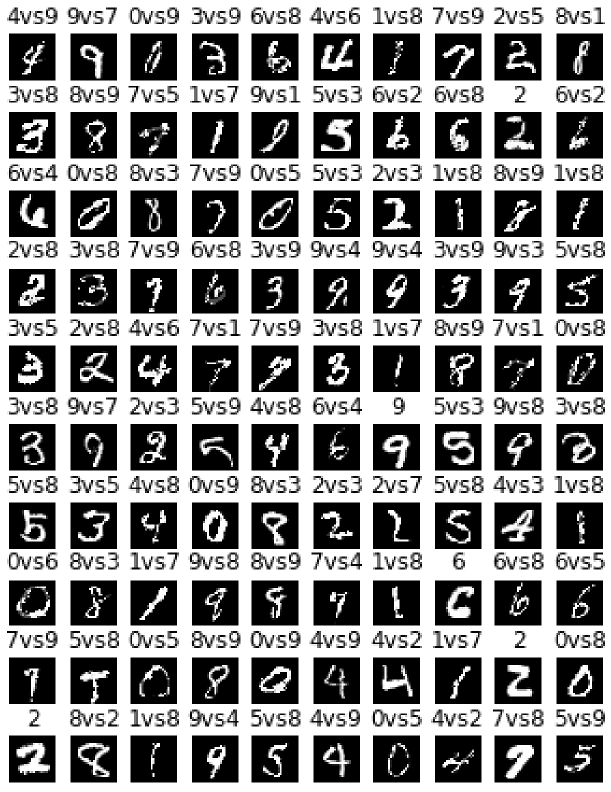
($\eta = 0.9$, Fooling rate(14%))

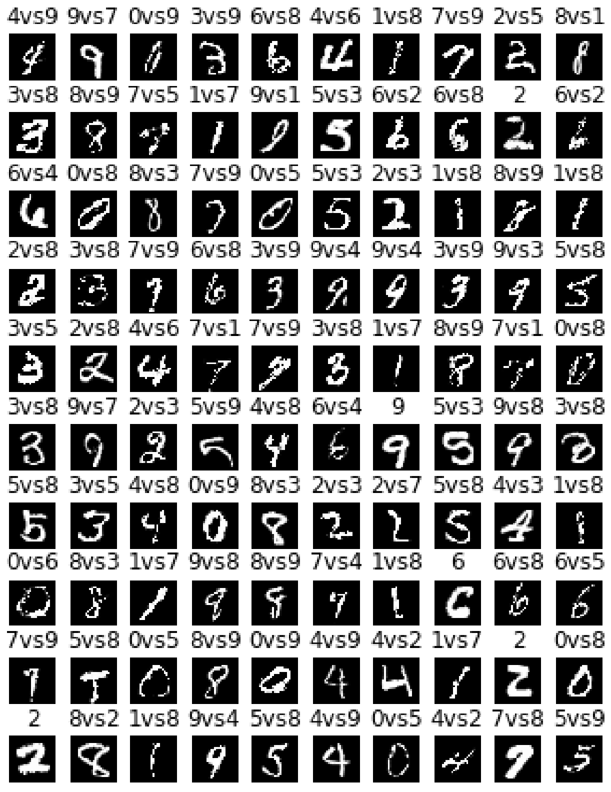


2.7 MNIST: Adversarial examples generated by ART C&W(ℓ_∞) attack against SVM in the untargeted settings with 10 iterations.


4 9 0 3 6 4 1 7 2 8vs1
3 8 7 1vs7 9vs7 5vs3 6 6 2 6
6 0 8vs3 7 0vs5 5 2 1 8 1
2 3 7 6 3 9 9 3 9 5
3vs2 2vs8 4 7 7 3 1 8 7 0
3 9vs7 2 5vs9 4 6 9 5vs3 9 3
5 3 4 0 8 2 2 5 4vs3 1
0 8 1vs7 9 8 7 1 6 6 6
7 5 0 8 0 4 4vs2 1 2 0
2 8vs2 1 9 5 4 0 4 7vs8 5
($\epsilon = 0.1$, Fooling rate(16%))

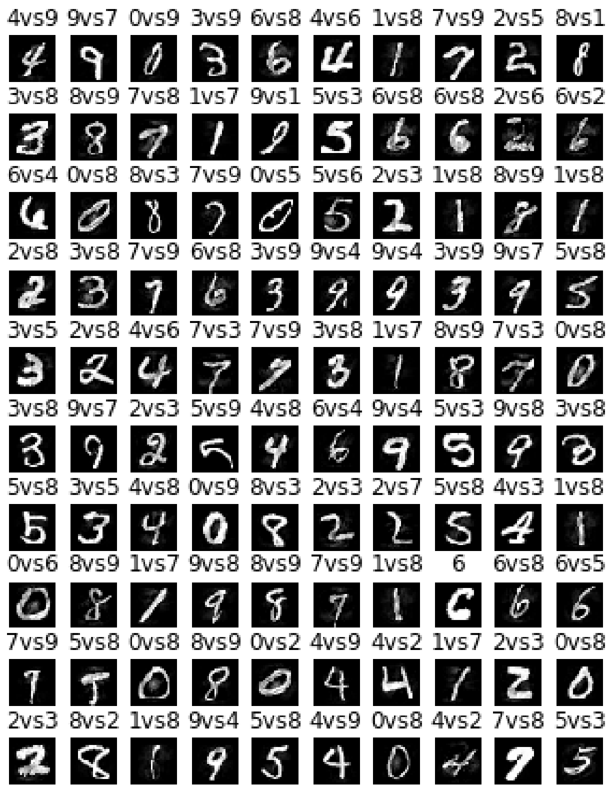

4vs9 9vs7 0vs9 3vs9 6 4vs6 1vs8 7vs9 2vs5 8vs1
3vs8 8 7vs8 1vs7 9vs1 5vs3 6vs2 6 2 6
6vs4 0 8vs3 7 0vs5 5 2vs3 1vs8 8 1
2 3 7 6 3vs9 9vs4 9vs4 3vs9 9vs3 5
3vs2 2vs8 4 7vs1 7vs9 3vs8 1 8 7 0
3vs8 9vs7 2vs3 5vs9 4 6vs4 9 5vs3 9vs8 3vs8
5vs8 3vs5 4 0vs9 8vs3 2 2 5 4vs3 1
0 8 1 9vs8 8vs9 7 1 6 6vs8 6vs5
7vs9 5 0 8vs9 0 4vs9 4vs2 1 2 0vs8
2 8vs2 1 9vs4 5vs8 4vs9 0vs5 4 7vs8 5vs9
($\epsilon = 0.3$, Fooling rate(60%))


4vs9 9vs7 0vs9 3vs9 6vs8 4vs6 1vs8 7vs9 2vs5 8vs1
3vs8 8vs9 7vs5 1vs7 9vs1 5vs3 6vs2 6vs8 2 6vs2
6vs4 0vs8 8vs3 7vs9 0vs5 5vs3 2vs3 1vs8 8vs9 1vs8
2vs8 3vs8 7vs9 6vs8 3vs9 9vs4 9vs4 3vs9 9vs3 5vs8
3vs5 2vs8 4vs6 7vs1 7vs9 3vs8 1vs7 8vs9 7vs1 0vs8
3vs8 9vs7 2vs3 5vs9 4vs8 6vs4 9 5vs3 9vs8 3vs8
5vs8 3vs5 4vs8 0vs9 8vs3 2vs3 2vs7 5vs8 4vs3 1vs8
0vs6 8vs3 1vs7 9vs8 8vs9 7vs4 1vs8 6 6vs8 6vs5
7vs9 5vs8 0vs5 8vs9 0vs9 4vs9 4vs2 1vs7 2 0vs8
2 8vs2 1vs8 9vs4 5vs8 4vs9 0vs5 4vs2 7vs8 5vs9
($\epsilon = 0.7$, Fooling rate(96%))

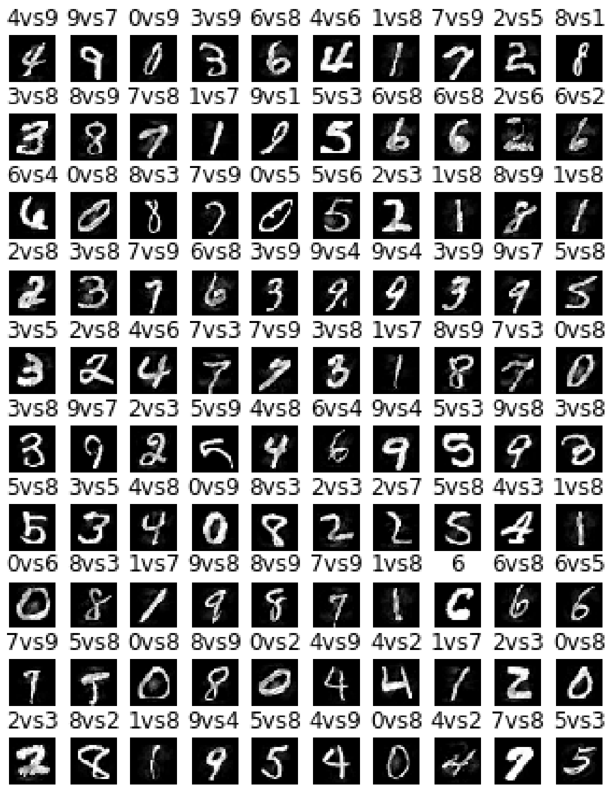

4vs9 9vs7 0vs9 3vs9 6vs8 4vs6 1vs8 7vs9 2vs5 8vs1
3vs8 8vs9 7vs5 1vs7 9vs1 5vs3 6vs2 6vs8 2 6vs2
6vs4 0vs8 8vs3 7vs9 0vs5 5vs3 2vs3 1vs8 8vs9 1vs8
2vs8 3vs8 7vs9 6vs8 3vs9 9vs4 9vs4 3vs9 9vs3 5vs8
3vs5 2vs8 4vs6 7vs1 7vs9 3vs8 1vs7 8vs9 7vs1 0vs8
3vs8 9vs7 2vs3 5vs9 4vs8 6vs4 9 5vs3 9vs8 3vs8
5vs8 3vs5 4vs8 0vs9 8vs3 2vs3 2vs7 5vs8 4vs3 1vs8
0vs6 8vs3 1vs7 9vs8 8vs9 7vs4 1vs8 6 6vs8 6vs5
7vs9 5vs8 0vs5 8vs9 0vs9 4vs9 4vs2 1vs7 2 0vs8
2 8vs2 1vs8 9vs4 5vs8 4vs9 0vs5 4vs2 7vs8 5vs9
($\epsilon = 0.9$, Fooling rate(96%))

2.8 MNIST: Adversarial examples generated by ART ElasticNet(EN) attack against SVM in the untargeted settings.

($\theta = 0.1, \gamma = 0.1$, Fooling rate(100%))

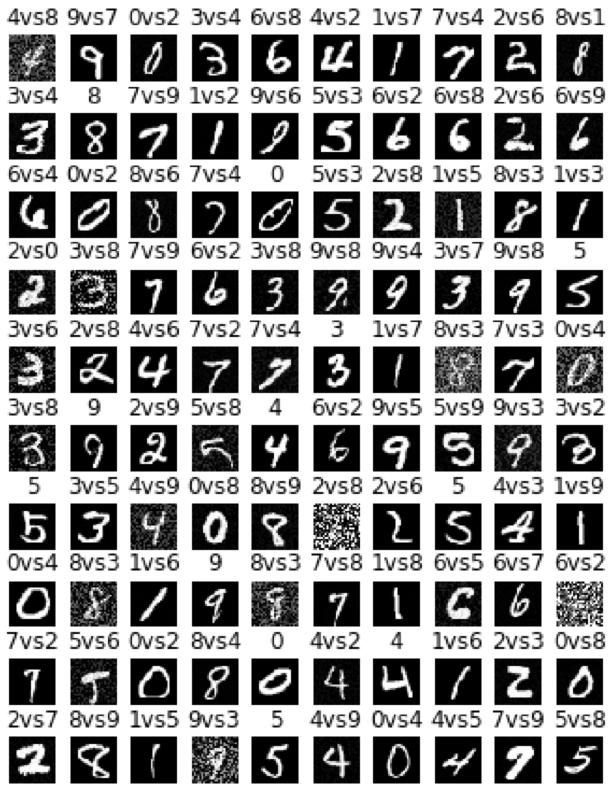

4vs9 9vs7 0vs9 3vs9 6vs8 4vs6 1vs8 7vs9 2vs5 8vs1
3vs8 8vs9 7vs8 1vs7 9vs1 5vs3 6vs8 6vs8 2vs6 6vs2
6vs4 0vs8 8vs3 7vs9 0vs5 5vs6 2vs3 1vs8 8vs9 1vs8
2vs8 3vs8 7vs9 6vs8 3vs9 9vs4 9vs4 3vs9 9vs7 5vs8
3vs5 2vs8 4vs6 7vs3 7vs9 3vs8 1vs7 8vs9 7vs3 0vs8
3vs8 9vs7 2vs3 5vs9 4vs8 6vs4 9vs4 5vs3 9vs8 3vs8
5vs8 3vs5 4vs8 0vs9 8vs3 2vs3 2vs7 5vs8 4vs3 1vs8
0vs6 8vs9 1vs7 9vs8 8vs9 7vs9 1vs8 6 6vs8 6vs5
7vs9 5vs8 0vs8 8vs9 0vs2 4vs9 4vs2 1vs7 2vs3 0vs8
2vs3 8vs2 1vs8 9vs4 5vs8 4vs9 0vs8 4vs2 7vs8 5vs3

($\theta = 0.3, \gamma = 0.1$, Fooling rate(100%))

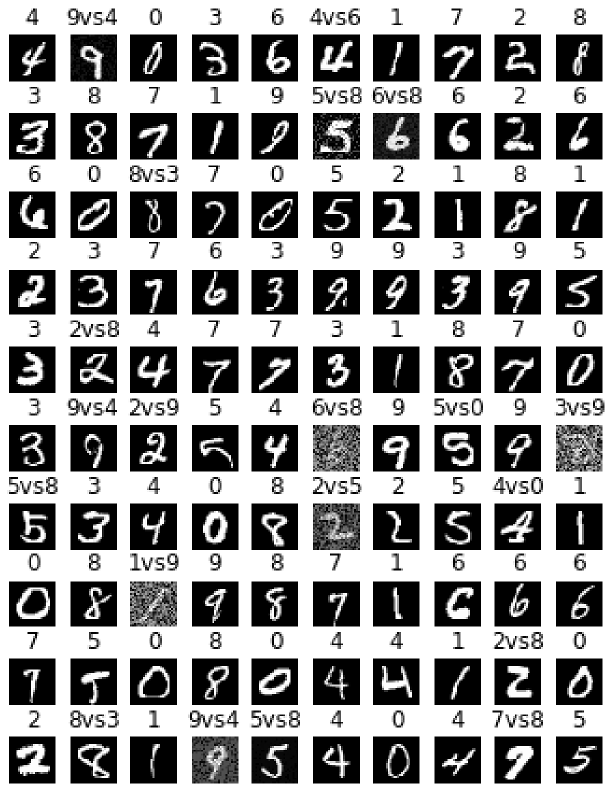

4vs9 9vs7 0vs9 3vs9 6vs8 4vs6 1vs8 7vs9 2vs5 8vs1
3vs8 8vs9 7vs8 1vs7 9vs1 5vs3 6vs8 6vs8 2vs6 6vs2
6vs4 0vs8 8vs3 7vs9 0vs5 5vs6 2vs3 1vs8 8vs9 1vs8
2vs8 3vs8 7vs9 6vs8 3vs9 9vs4 9vs4 3vs9 9vs7 5vs8
3vs5 2vs8 4vs6 7vs3 7vs9 3vs8 1vs7 8vs9 7vs3 0vs8
3vs8 9vs7 2vs3 5vs9 4vs8 6vs4 9vs4 5vs3 9vs8 3vs8
5vs8 3vs5 4vs8 0vs9 8vs3 2vs3 2vs7 5vs8 4vs3 1vs8
0vs6 8vs3 1vs7 9vs8 8vs9 7vs9 1vs8 6 6vs8 6vs5
7vs9 5vs8 0vs8 8vs9 0vs2 4vs9 4vs2 1vs7 2vs3 0vs8
2vs3 8vs2 1vs8 9vs4 5vs8 4vs9 0vs8 4vs2 7vs8 5vs3

3.1.1 MNIST: Adversarial examples generated by ART HopSkipJump attack against Decision Trees in the targeted and untargeted settings.

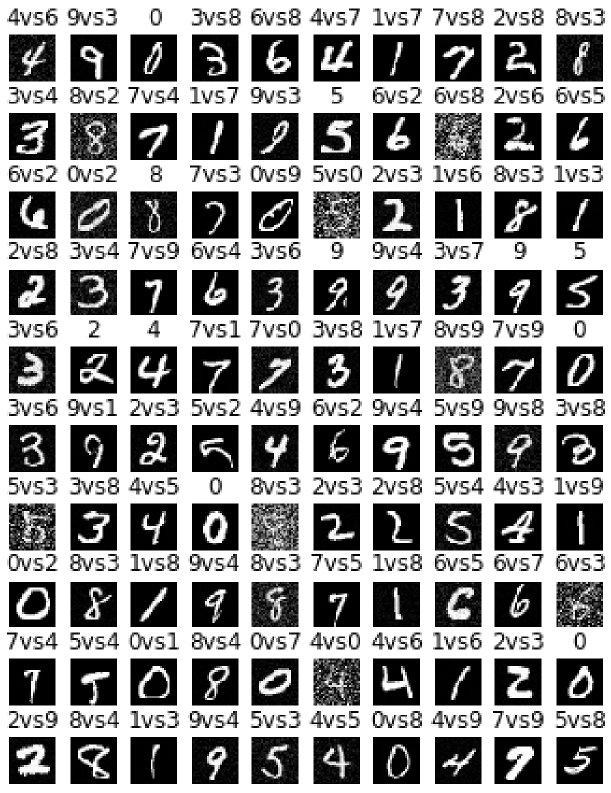
(Untargeted, ℓ_∞ norm, 87% fooling rate)


4vs8 9vs7 0vs2 3vs4 6vs8 4vs2 1vs7 7vs4 2vs6 8vs1
3vs4 8 7vs9 1vs2 9vs6 5vs3 6vs2 6vs8 2vs6 6vs9
6vs4 0vs2 8vs6 7vs4 0 5vs3 2vs8 1vs5 8vs3 1vs3
2vs0 3vs8 7vs9 6vs2 3vs8 9vs8 9vs4 3vs7 9vs8 5
3vs6 2vs8 4vs6 7vs2 7vs4 3 1vs7 8vs3 7vs3 0vs4
3vs8 9 2vs9 5vs8 4 6vs2 9vs5 5vs9 9vs3 3vs2
5 3vs5 4vs9 0vs8 8vs9 2vs8 2vs6 5 4vs3 1vs9
0vs4 8vs3 1vs6 9 8vs3 7vs8 1vs8 6vs5 6vs7 6vs2
7vs2 5vs6 0vs2 8vs4 0 4vs2 4 1vs6 2vs3 0vs8
2vs7 8vs9 1vs5 9vs3 5 4vs9 0vs4 4vs5 7vs9 5vs8

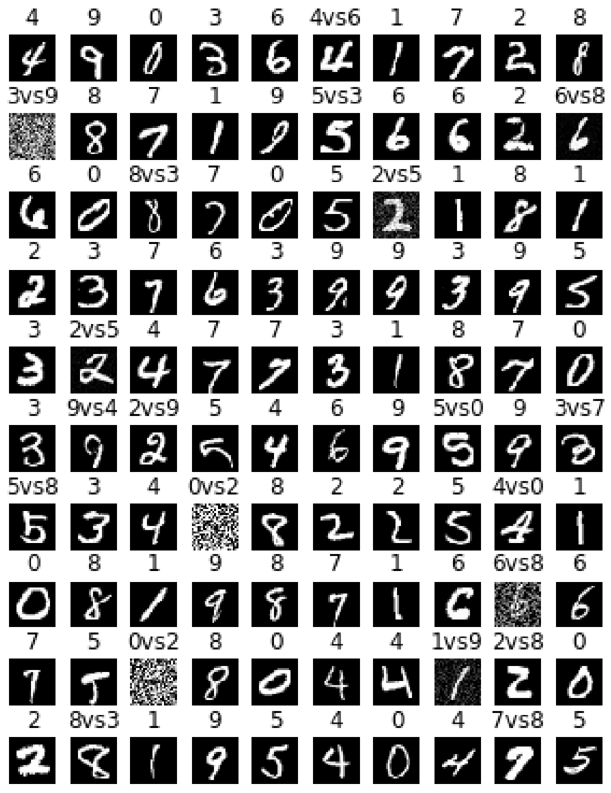
(Targeted, ℓ_∞ norm, 11% fooling rate)


4 9vs4 0 3 6 4vs6 1 7 2 8
3 8 7 1 9 5vs8 6vs8 6 2 6
6 0 8vs3 7 0 5 2 1 8 1
2 3 7 6 3 9 9 3 9 5
3 2vs8 4 7 7 3 1 8 7 0
3 9vs4 2vs9 5 4 6vs8 9 5vs0 9 3vs9
5vs8 3 4 0 8 2vs5 2 5 4vs0 1
0 8 1vs9 9 8 7 1 6 6 6
7 5 0 8 0 4 4 1 2vs8 0
2 8vs3 1 9vs4 5vs8 4 0 4 7vs8 5

(Untargeted, ℓ_2 norm, 92% fooling rate)

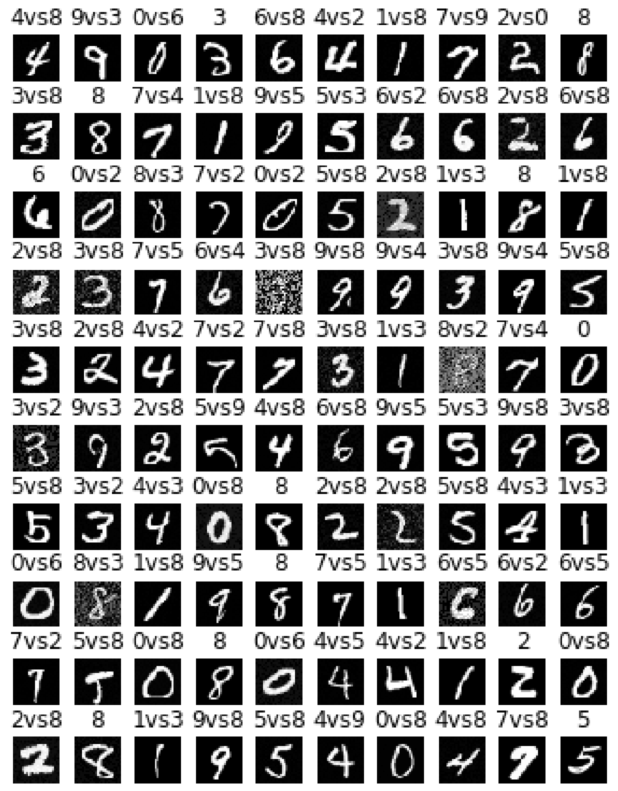

4vs6 9vs3 0 3vs8 6vs8 4vs7 1vs7 7vs8 2vs8 8vs3
3vs4 8vs2 7vs4 1vs7 9vs3 5 6vs2 6vs8 2vs6 6vs5
6vs2 0vs2 8 7vs3 0vs9 5vs0 2vs3 1vs6 8vs3 1vs3
2vs8 3vs4 7vs9 6vs4 3vs6 9 9vs4 3vs7 9 5
3vs6 2 4 7vs1 7vs0 3vs8 1vs7 8vs9 7vs9 0
3vs6 9vs1 2vs3 5vs2 4vs9 6vs2 9vs4 5vs9 9vs8 3vs8
5vs3 3vs8 4vs5 0 8vs3 2vs3 2vs8 5vs4 4vs3 1vs9
0vs2 8vs3 1vs8 9vs4 8vs3 7vs5 1vs8 6vs5 6vs7 6vs3
7vs4 5vs4 0vs1 8vs4 0vs7 4vs0 4vs6 1vs6 2vs3 0
2vs9 8vs4 1vs3 9vs4 5vs3 4vs5 0vs8 4vs9 7vs9 5vs8

(Targeted, ℓ_2 norm, 10% fooling rate)


4 9 0 3 6 4vs6 1 7 2 8
3vs9 8 7 1 9 5vs3 6 6 2 6vs8
6 0 8vs3 7 0 5 2vs5 1 8 1
2 3 7 6 3 9 9 3 9 5
3 2vs5 4 7 7 3 1 8 7 0
3 9vs4 2vs9 5 4 6 9 5vs0 9 3vs7
5vs8 3 4 0vs2 8 2 2 5 4vs0 1
0 8 1 9 8 7 1 6 6vs8 6
7 5 0vs2 8 0 4 4 1vs9 2vs8 0
2 8vs3 1 9 5 4 0 4 7vs8 5

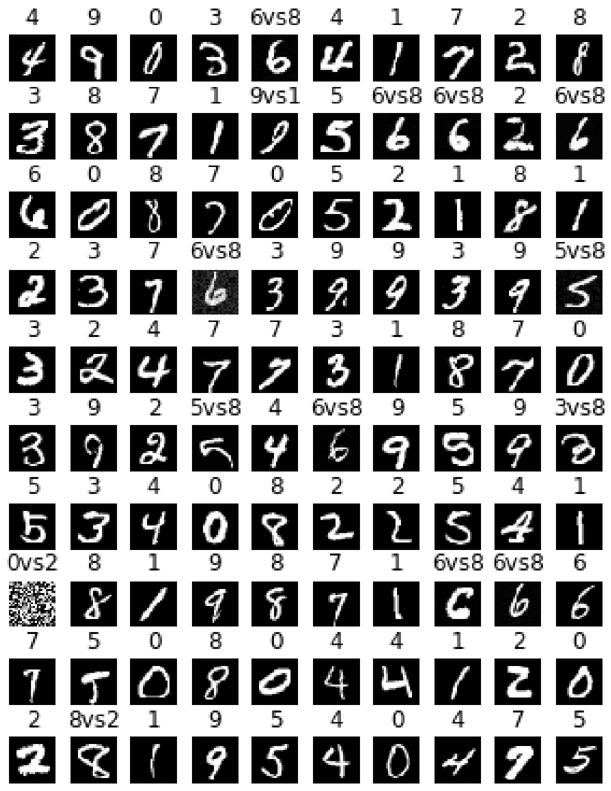
3.1.2 MNIST: Adversarial examples generated by ART HopSkipJump attack against Random Forest in the targeted and untargeted settings.

(Untargeted, ℓ_∞ norm, 87% fooling rate)

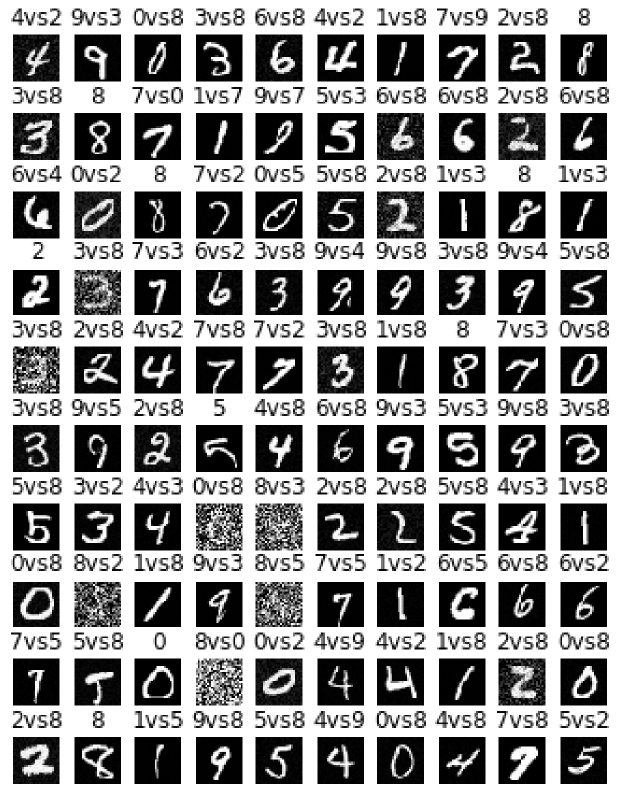


4vs8 9vs3 0vs6 3 6vs8 4vs2 1vs8 7vs9 2vs0 8
3vs8 8 7vs4 1vs8 9vs5 5vs3 6vs2 6vs8 2vs8 6vs8
6 0vs2 8vs3 7vs2 0vs2 5vs8 2vs8 1vs3 8 1vs8
2vs8 3vs8 7vs5 6vs4 3vs8 9vs8 9vs4 3vs8 9vs4 5vs8
3vs8 2vs8 4vs2 7vs2 7vs8 3vs8 1vs3 8vs2 7vs4 0
3vs2 9vs3 2vs8 5vs9 4vs8 6vs8 9vs5 5vs3 9vs8 3vs8
5vs8 3vs2 4vs3 0vs8 8 2vs8 2vs8 5vs8 4vs3 1vs3
0vs6 8vs3 1vs8 9vs5 8 7vs5 1vs3 6vs5 6vs2 6vs5
7vs2 5vs8 0vs8 8 0vs6 4vs5 4vs2 1vs8 2 0vs8
2vs8 8 1vs3 9vs8 5vs8 4vs9 0vs8 4vs8 7vs8 5

(Targeted, ℓ_∞ norm, 11% fooling rate)

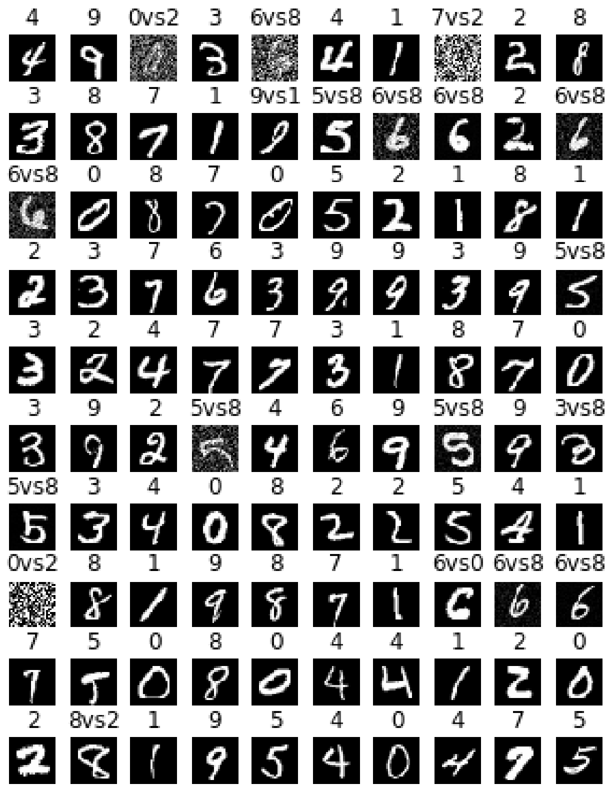


(Untargeted, ℓ_2 norm, 92% fooling rate)



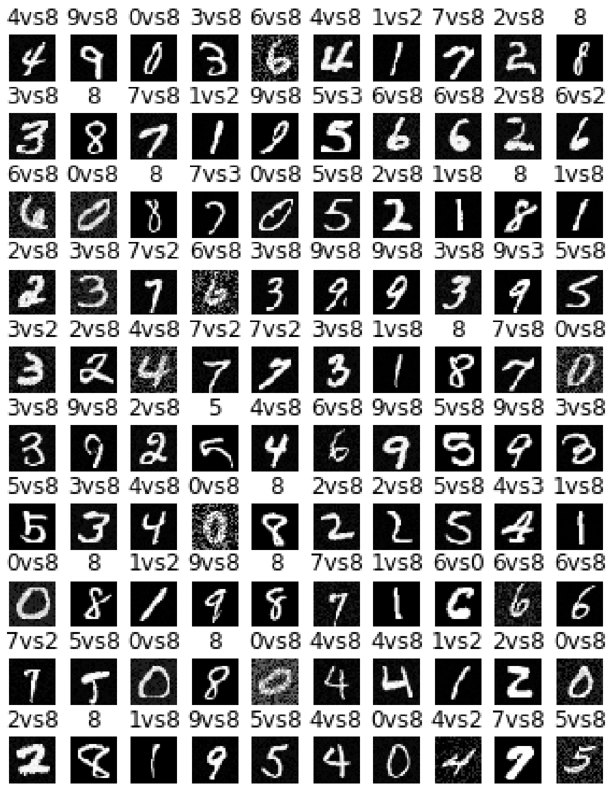
4vs2 9vs3 0vs8 3vs8 6vs8 4vs2 1vs8 7vs9 2vs8 8
3vs8 8 7vs0 1vs7 9vs7 5vs3 6vs8 6vs8 2vs8 6vs8
6vs4 0vs2 8 7vs2 0vs5 5vs8 2vs8 1vs3 8 1vs3
2 3vs8 7vs3 6vs2 3vs8 9vs4 9vs8 3vs8 9vs4 5vs8
3vs8 2vs8 4vs2 7vs8 7vs2 3vs8 1vs8 8 7vs3 0vs8
3vs8 9vs5 2vs8 5 4vs8 6vs8 9vs3 5vs3 9vs8 3vs8
5vs8 3vs2 4vs3 0vs8 8vs3 2vs8 2vs8 5vs8 4vs3 1vs8
0vs8 8vs2 1vs8 9vs3 8vs5 7vs5 1vs2 6vs5 6vs8 6vs2
7vs5 5vs8 0 8vs0 0vs2 4vs9 4vs2 1vs8 2vs8 0vs8
2vs8 8 1vs5 9vs8 5vs8 4vs9 0vs8 4vs8 7vs8 5vs2

(Targeted, ℓ_2 norm, 15% fooling rate)

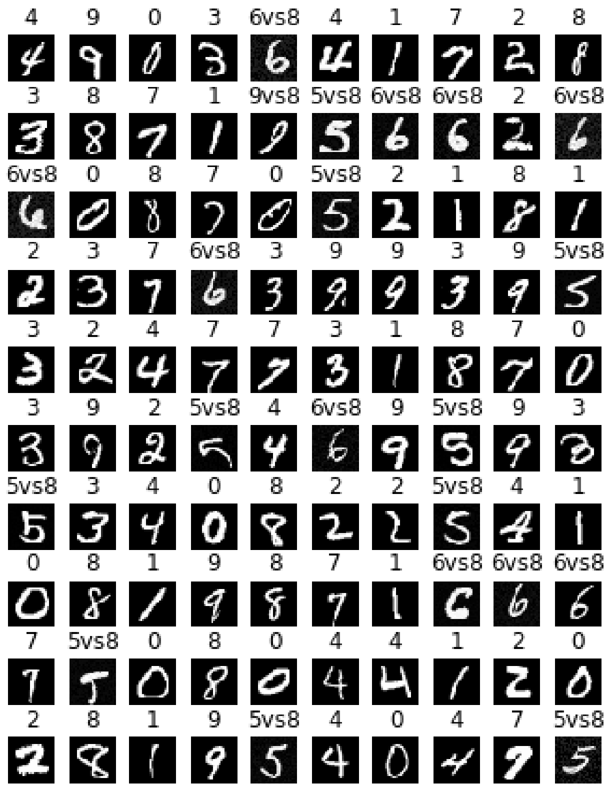


3.1.3 MNIST: Adversarial examples generated by ART HopSkipJump attack against GBDT in the targeted and untargeted settings.

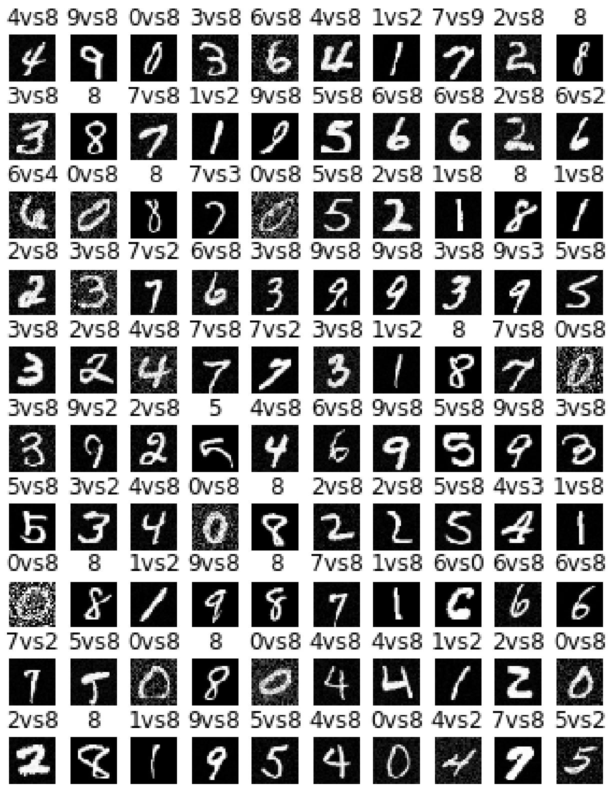
(Untargeted, ℓ_∞ norm, 89% fooling rate)


4vs8 9vs8 0vs8 3vs8 6vs8 4vs8 1vs2 7vs8 2vs8 8
3vs8 8 7vs8 1vs2 9vs8 5vs3 6vs8 6vs8 2vs8 6vs2
6vs8 0vs8 8 7vs3 0vs8 5vs8 2vs8 1vs8 8 1vs8
2vs8 3vs8 7vs2 6vs8 3vs8 9vs8 9vs8 3vs8 9vs3 5vs8
3vs2 2vs8 4vs8 7vs2 7vs2 3vs8 1vs8 8 7vs8 0vs8
3vs8 9vs8 2vs8 5 4vs8 6vs8 9vs8 5vs8 9vs8 3vs8
5vs8 3vs8 4vs8 0vs8 8 2vs8 2vs8 5vs8 4vs3 1vs8
0vs8 8 1vs2 9vs8 8 7vs8 1vs8 6vs0 6vs8 6vs8
7vs2 5vs8 0vs8 8 0vs8 4vs8 4vs8 1vs2 2vs8 0vs8
2vs8 8 1vs8 9vs8 5vs8 4vs8 0vs8 4vs2 7vs8 5vs8

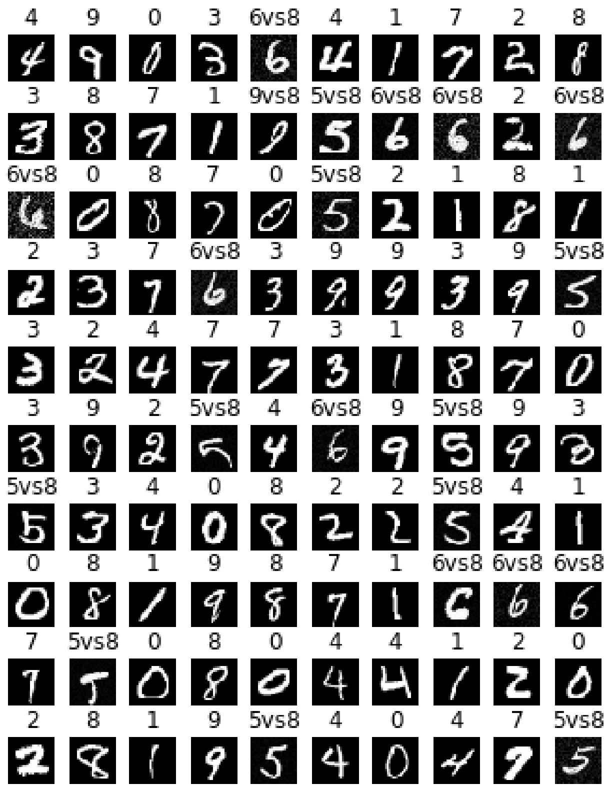
(Targeted, ℓ_∞ norm, 20% fooling rate)


4 9 0 3 6vs8 4 1 7 2 8
3 8 7 1 9vs8 5vs8 6vs8 6vs8 2 6vs8
6vs8 0 8 7 0 5vs8 2 1 8 1
2 3 7 6vs8 3 9 9 3 9 5vs8
3 2 4 7 7 3 1 8 7 0
3 9 2 5vs8 4 6vs8 9 5vs8 9 3
5vs8 3 4 0 8 2 2 5vs8 4 1
0 8 1 9 8 7 1 6vs8 6vs8 6vs8
7 5vs8 0 8 0 4 4 1 2 0
2 8 1 9 5vs8 4 0 4 7 5vs8

(Untargeted, ℓ_2 norm, 89% fooling rate)

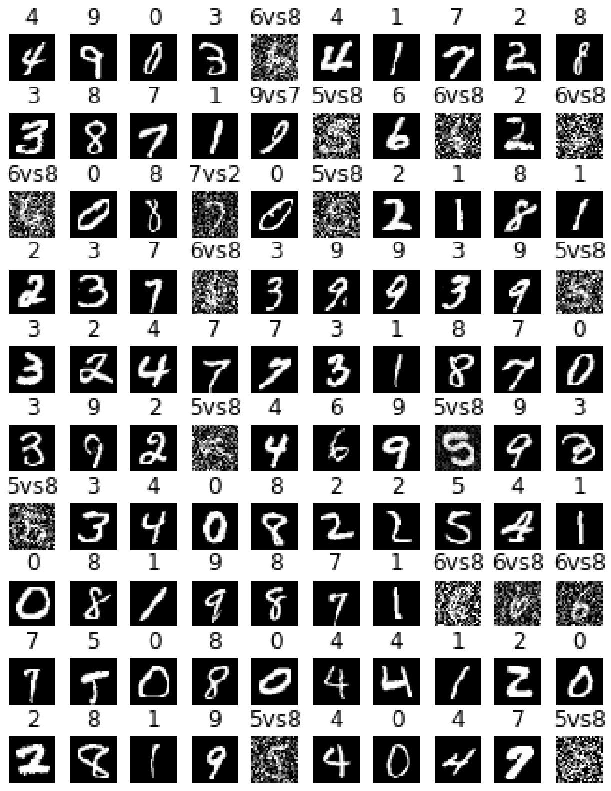

4vs8 9vs8 0vs8 3vs8 6vs8 4vs8 1vs2 7vs9 2vs8 8
3vs8 8 7vs8 1vs2 9vs8 5vs8 6vs8 6vs8 2vs8 6vs2
6vs4 0vs8 8 7vs3 0vs8 5vs8 2vs8 1vs8 8 1vs8
2vs8 3vs8 7vs2 6vs8 3vs8 9vs8 9vs8 3vs8 9vs3 5vs8
3vs8 2vs8 4vs8 7vs8 7vs2 3vs8 1vs2 8 7vs8 0vs8
3vs8 9vs2 2vs8 5 4vs8 6vs8 9vs8 5vs8 9vs8 3vs8
5vs8 3vs2 4vs8 0vs8 8 2vs8 2vs8 5vs8 4vs3 1vs8
0vs8 8 1vs2 9vs8 8 7vs8 1vs8 6vs0 6vs8 6vs8
7vs2 5vs8 0vs8 8 0vs8 4vs8 4vs8 1vs2 2vs8 0vs8
2vs8 8 1vs8 9vs8 5vs8 4vs8 0vs8 4vs2 7vs8 5vs2

(Targeted, ℓ_2 norm, 20% fooling rate)

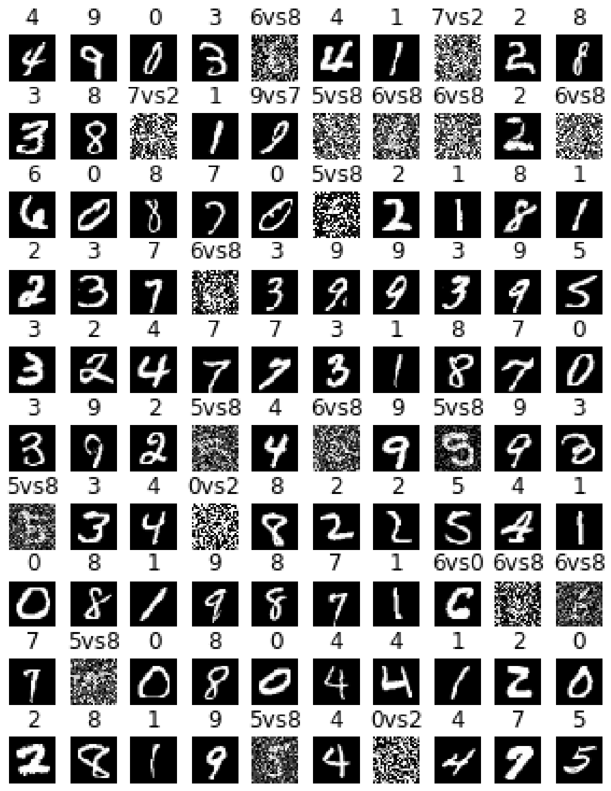

4 9 0 3 6vs8 4 1 7 2 8
3 8 7 1 9vs8 5vs8 6vs8 6vs8 2 6vs8
6vs8 0 8 7 0 5vs8 2 1 8 1
2 3 7 6vs8 3 9 9 3 9 5vs8
3 2 4 7 7 3 1 8 7 0
3 9 2 5vs8 4 6vs8 9 5vs8 9 3
5vs8 3 4 0 8 2 2 5vs8 4 1
0 8 1 9 8 7 1 6vs8 6vs8 6vs8
7 5vs8 0 8 0 4 4 1 2 0
2 8 1 9 5vs8 4 0 4 7 5vs8

3.1.4 MNIST: Adversarial examples generated by ART HopSkipJump attack against SVM in the targeted and untargeted settings.

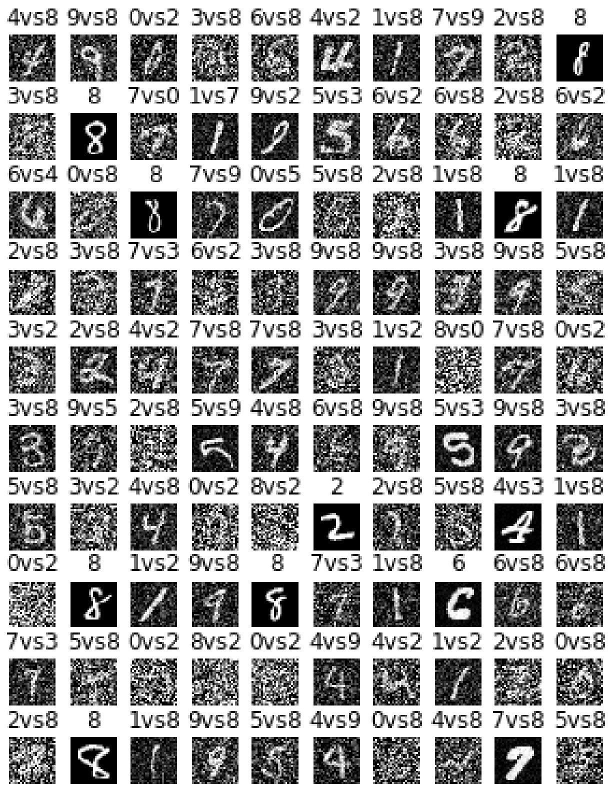
(Untargeted, ℓ_∞ norm, 91% fooling rate)


4 9 0 3 6vs8 4 1 7 2 8
3 8 7 1 9vs7 5vs8 6 6vs8 2 6vs8
6vs8 0 8 7vs2 0 5vs8 2 1 8 1
2 3 7 6vs8 3 9 9 3 9 5vs8
3 2 4 7 7 3 1 8 7 0
3 9 2 5vs8 4 6 9 5vs8 9 3
5vs8 3 4 0 8 2 2 5 4 1
0 8 1 9 8 7 1 6vs8 6vs8 6vs8
7 5 0 8 0 4 4 1 2 0
2 8 1 9 5vs8 4 0 4 7 5vs8

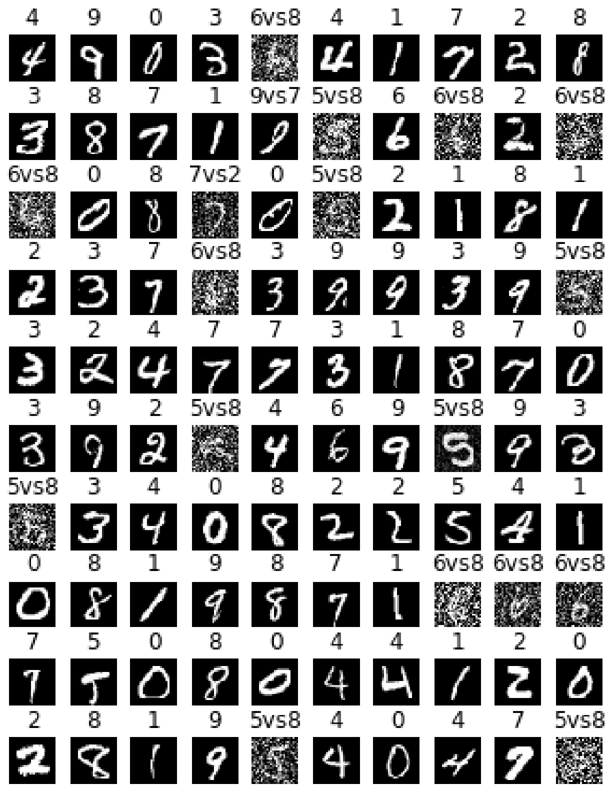
(Targeted, ℓ_∞ norm, 19% fooling rate)


4 9 0 3 6vs8 4 1 7vs2 2 8
3 8 7vs2 1 9vs7 5vs8 6vs8 6vs8 2 6vs8
6 0 8 7 0 5vs8 2 1 8 1
2 3 7 6vs8 3 9 9 3 9 5
3 2 4 7 7 3 1 8 7 0
3 9 2 5vs8 4 6vs8 9 5vs8 9 3
5vs8 3 4 0vs2 8 2 2 5 4 1
0 8 1 9 8 7 1 6vs0 6vs8 6vs8
7 5vs8 0 8 0 4 4 1 2 0
2 8 1 9 5vs8 4 0vs2 4 7 5

(Untargeted, ℓ_2 norm, 92% fooling rate)

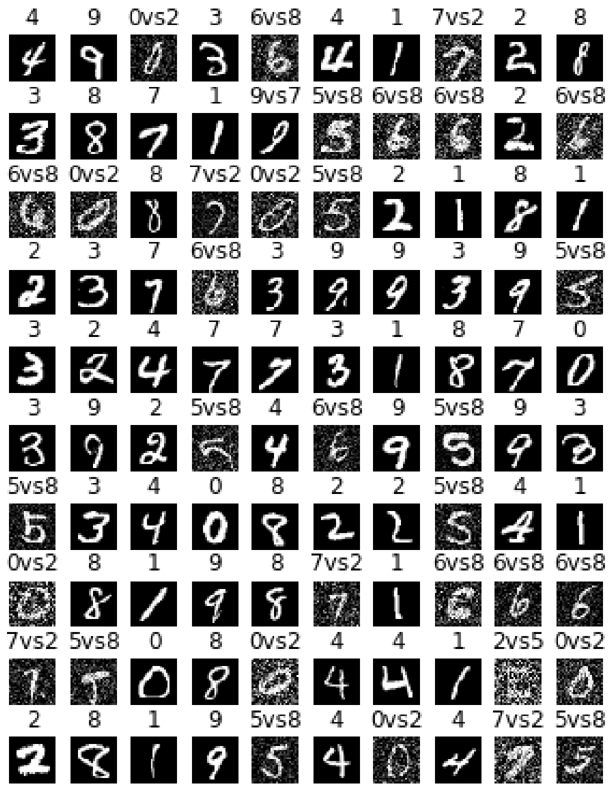

4vs8 9vs8 0vs2 3vs8 6vs8 4vs2 1vs8 7vs9 2vs8 8
3vs8 8 7vs0 1vs7 9vs2 5vs3 6vs2 6vs8 2vs8 6vs2
6vs4 0vs8 8 7vs9 0vs5 5vs8 2vs8 1vs8 8 1vs8
2vs8 3vs8 7vs3 6vs2 3vs8 9vs8 9vs8 3vs8 9vs8 5vs8
3vs2 2vs8 4vs2 7vs8 7vs8 3vs8 1vs2 8vs0 7vs8 0vs2
3vs8 9vs5 2vs8 5vs9 4vs8 6vs8 9vs8 5vs3 9vs8 3vs8
5vs8 3vs2 4vs8 0vs2 8vs2 2 2vs8 5vs8 4vs3 1vs8
0vs2 8 1vs2 9vs8 8 7vs3 1vs8 6 6vs8 6vs8
7vs3 5vs8 0vs2 8vs2 0vs2 4vs9 4vs2 1vs2 2vs8 0vs8
2vs8 8 1vs8 9vs8 5vs8 4vs9 0vs8 4vs8 7vs8 5vs8

(Targeted, ℓ_2 norm, 17% fooling rate)

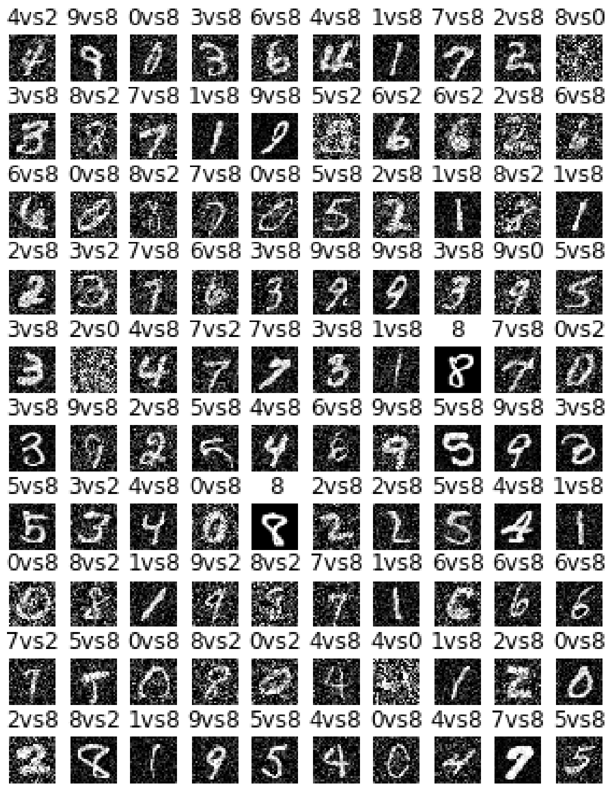

4 9 0 3 6vs8 4 1 7 2 8
3 8 7 1 9vs7 5vs8 6 6vs8 2 6vs8
6vs8 0 8 7vs2 0 5vs8 2 1 8 1
2 3 7 6vs8 3 9 9 3 9 5vs8
3 2 4 7 7 3 1 8 7 0
3 9 2 5vs8 4 6 9 5vs8 9 3
5vs8 3 4 0 8 2 2 5 4 1
0 8 1 9 8 7 1 6vs8 6vs8 6vs8
7 5 0 8 0 4 4 1 2 0
2 8 1 9 5vs8 4 0 4 7 5vs8

3.2 MNIST: Adversarial examples generated by ART BA(ℓ_2) with $\delta = 0.01$, $\varepsilon = 0.01$, and 100 iterations in the untargeted setting.

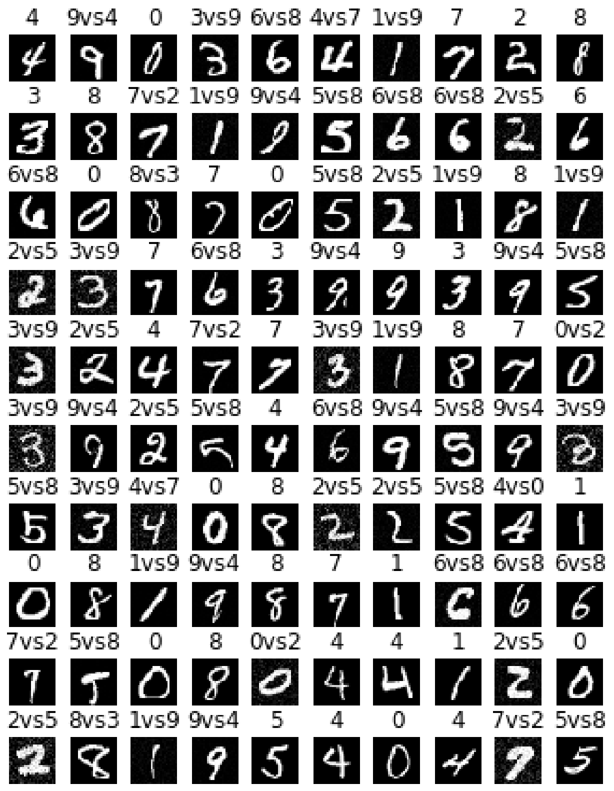
(SVM, Targeted , Fooling rate of 33%)


4 9 0vs2 3 6vs8 4 1 7vs2 2 8
3 8 7 1 9vs7 5vs8 6vs8 6vs8 2 6vs8
6vs8 0vs2 8 7vs2 0vs2 5vs8 2 1 8 1
2 3 7 6vs8 3 9 9 3 9 5vs8
3 2 4 7 7 3 1 8 7 0
3 9 2 5vs8 4 6vs8 9 5vs8 9 3
5vs8 3 4 0 8 2 2 5vs8 4 1
0vs2 8 1 9 8 7vs2 1 6vs8 6vs8 6vs8
7vs2 5vs8 0 8 0vs2 4 4 1 2vs5 0vs2
2 8 1 9 5vs8 4 0vs2 4 7vs2 5vs8

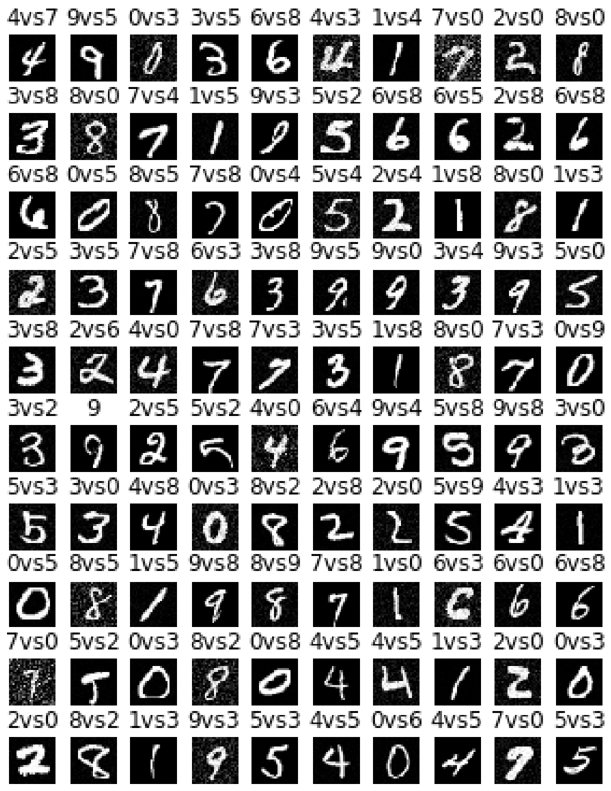
(SVM, Untargeted, Fooling rate of 98%)


4vs2 9vs8 0vs8 3vs8 6vs8 4vs8 1vs8 7vs8 2vs8 8vs0
3vs8 8vs2 7vs8 1vs8 9vs8 5vs2 6vs2 6vs2 2vs8 6vs8
6vs8 0vs8 8vs2 7vs8 0vs8 5vs8 2vs8 1vs8 8vs2 1vs8
2vs8 3vs2 7vs8 6vs8 3vs8 9vs8 9vs8 3vs8 9vs0 5vs8
3vs8 2vs0 4vs8 7vs2 7vs8 3vs8 1vs8 8 7vs8 0vs2
3vs8 9vs8 2vs8 5vs8 4vs8 6vs8 9vs8 5vs8 9vs8 3vs8
5vs8 3vs2 4vs8 0vs8 8 2vs8 2vs8 5vs8 4vs8 1vs8
0vs8 8vs2 1vs8 9vs2 8vs2 7vs8 1vs8 6vs8 6vs8 6vs8
7vs2 5vs8 0vs8 8vs2 0vs2 4vs8 4vs0 1vs8 2vs8 0vs8
2vs8 8vs2 1vs8 9vs8 5vs8 4vs8 0vs8 4vs8 7vs8 5vs8

(DT, Targeted, Fooling rate of 57.99%)


4 9vs4 0 3vs9 6vs8 4vs7 1vs9 7 2 8
3 8 7vs2 1vs9 9vs4 5vs8 6vs8 6vs8 2vs5 6
6vs8 0 8vs3 7 0 5vs8 2vs5 1vs9 8 1vs9
2vs5 3vs9 7 6vs8 3 9vs4 9 3 9vs4 5vs8
3vs9 2vs5 4 7vs2 7 3vs9 1vs9 8 7 0vs2
3vs9 9vs4 2vs5 5vs8 4 6vs8 9vs4 5vs8 9vs4 3vs9
5vs8 3vs9 4vs7 0 8 2vs5 2vs5 5vs8 4vs0 1
0 8 1vs9 9vs4 8 7 1 6vs8 6vs8 6vs8
7vs2 5vs8 0 8 0vs2 4 4 1 2vs5 0
2vs5 8vs3 1vs9 9vs4 5 4 0 4 7vs2 5vs8

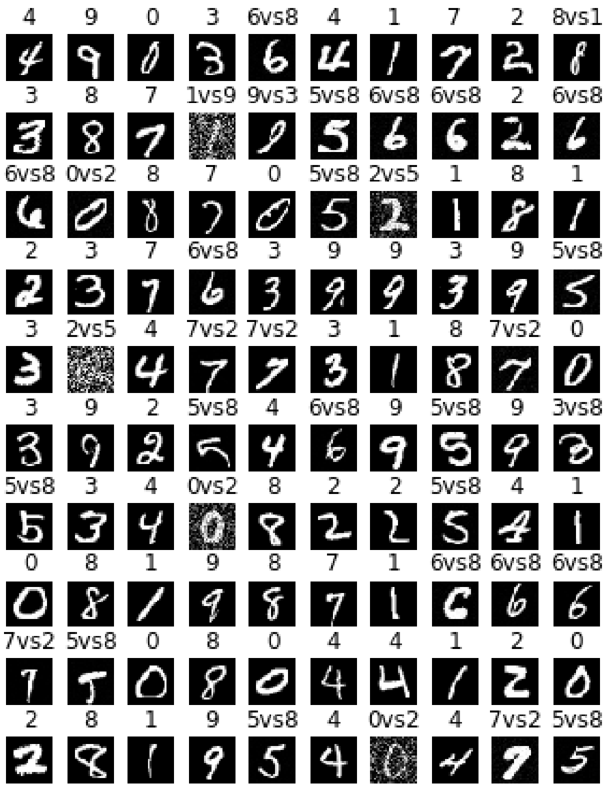
(DT, Untargeted, Fooling rate of 100%)

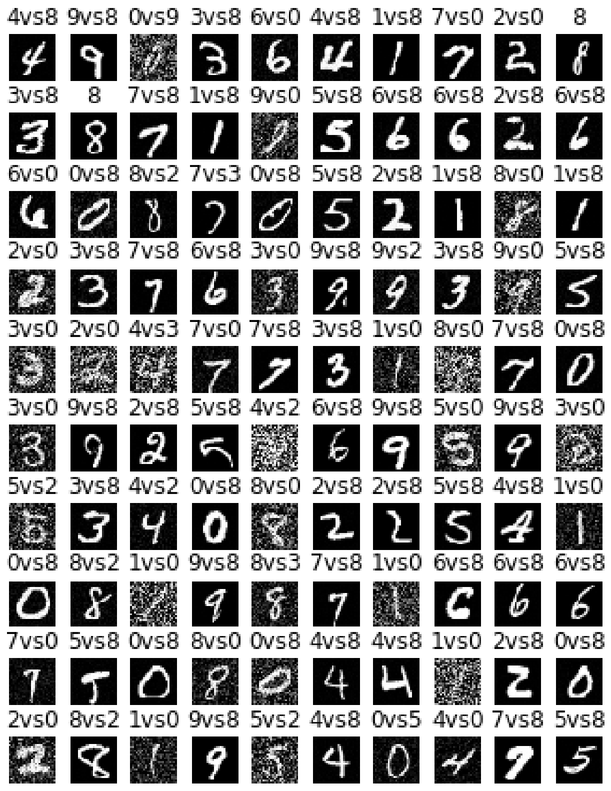

4vs7 9vs5 0vs3 3vs5 6vs8 4vs3 1vs4 7vs0 2vs0 8vs0
3vs8 8vs0 7vs4 1vs5 9vs3 5vs2 6vs8 6vs5 2vs8 6vs8
6vs8 0vs5 8vs5 7vs8 0vs4 5vs4 2vs4 1vs8 8vs0 1vs3
2vs5 3vs5 7vs8 6vs3 3vs8 9vs5 9vs0 3vs4 9vs3 5vs0
3vs8 2vs6 4vs0 7vs8 7vs3 3vs5 1vs8 8vs0 7vs3 0vs9
3vs2 9 2vs5 5vs2 4vs0 6vs4 9vs4 5vs8 9vs8 3vs0
5vs3 3vs0 4vs8 0vs3 8vs2 2vs8 2vs0 5vs9 4vs3 1vs3
0vs5 8vs5 1vs5 9vs8 8vs9 7vs8 1vs0 6vs3 6vs0 6vs8
7vs0 5vs2 0vs3 8vs2 0vs8 4vs5 4vs5 1vs3 2vs0 0vs3
2vs0 8vs2 1vs3 9vs3 5vs3 4vs5 0vs6 4vs5 7vs0 5vs3

3.2 MNIST: cont.

(RF, Targeted, Fooling rate of 31%)

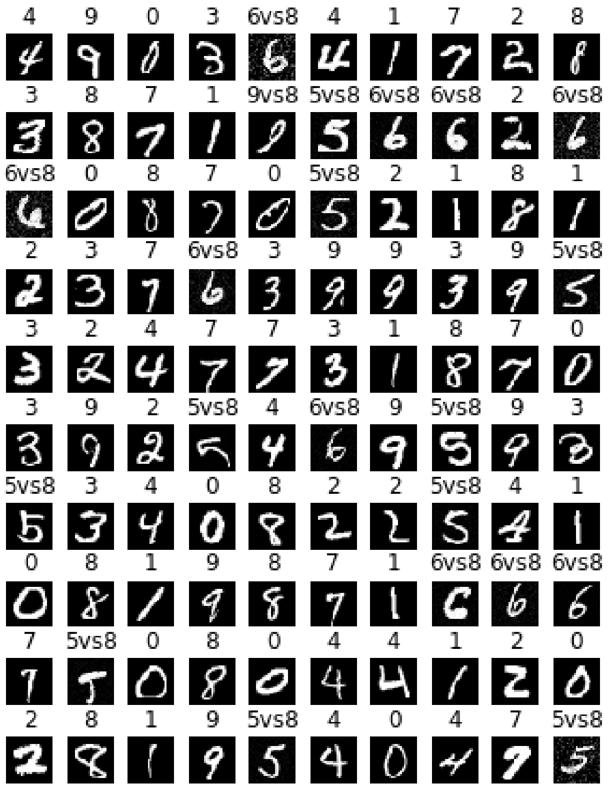
(RF, Untargeted, Fooling rate of 99%)

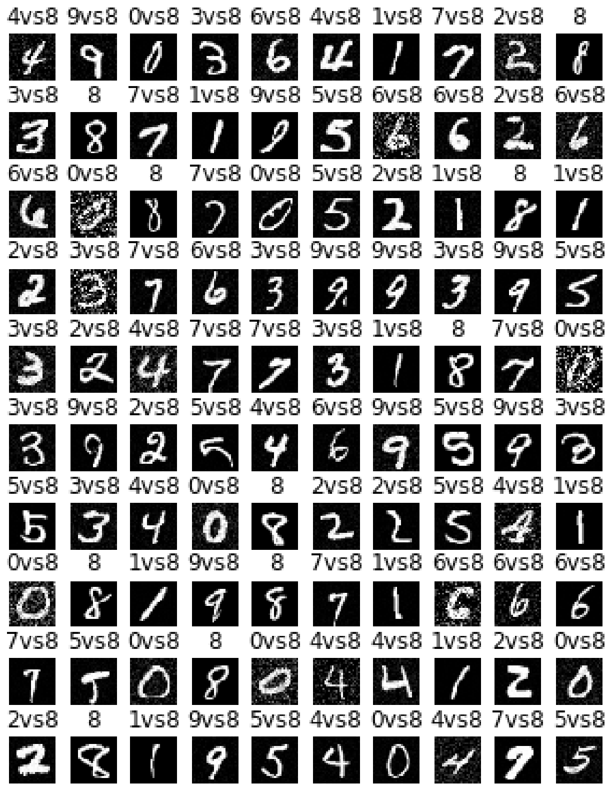

4 9 0 3 6vs8 4 1 7 2 8vs1
3 8 7 1vs9 9vs3 5vs8 6vs8 6vs8 2 6vs8
6vs8 0vs2 8 7 0 5vs8 2vs5 1 8 1
2 3 7 6vs8 3 9 9 3 9 5vs8
3 2vs5 4 7vs2 7vs2 3 1 8 7vs2 0
3 9 2 5vs8 4 6vs8 9 5vs8 9 3vs8
5vs8 3 4 0vs2 8 2 2 5vs8 4 1
0 8 1 9 8 7 1 6vs8 6vs8 6vs8
7vs2 5vs8 0 8 0 4 4 1 2 0
2 8 1 9 5vs8 4 0vs2 4 7vs2 5vs8


4vs8 9vs8 0vs9 3vs8 6vs0 4vs8 1vs8 7vs0 2vs0 8
3vs8 8 7vs8 1vs8 9vs0 5vs8 6vs8 6vs8 2vs8 6vs8
6vs0 0vs8 8vs2 7vs3 0vs8 5vs8 2vs8 1vs8 8vs0 1vs8
2vs0 3vs8 7vs8 6vs8 3vs0 9vs8 9vs2 3vs8 9vs0 5vs8
3vs0 2vs0 4vs3 7vs0 7vs8 3vs8 1vs0 8vs0 7vs8 0vs8
3vs0 9vs8 2vs8 5vs8 4vs2 6vs8 9vs8 5vs0 9vs8 3vs0
5vs2 3vs8 4vs2 0vs8 8vs0 2vs8 2vs8 5vs8 4vs8 1vs0
0vs8 8vs2 1vs0 9vs8 8vs3 7vs8 1vs0 6vs8 6vs8 6vs8
7vs0 5vs8 0vs8 8vs0 0vs8 4vs8 4vs8 1vs0 2vs8 0vs8
2vs0 8vs2 1vs0 9vs8 5vs2 4vs8 0vs5 4vs0 7vs8 5vs8

(GBDT, Targeted, Fooling rate of 20%)

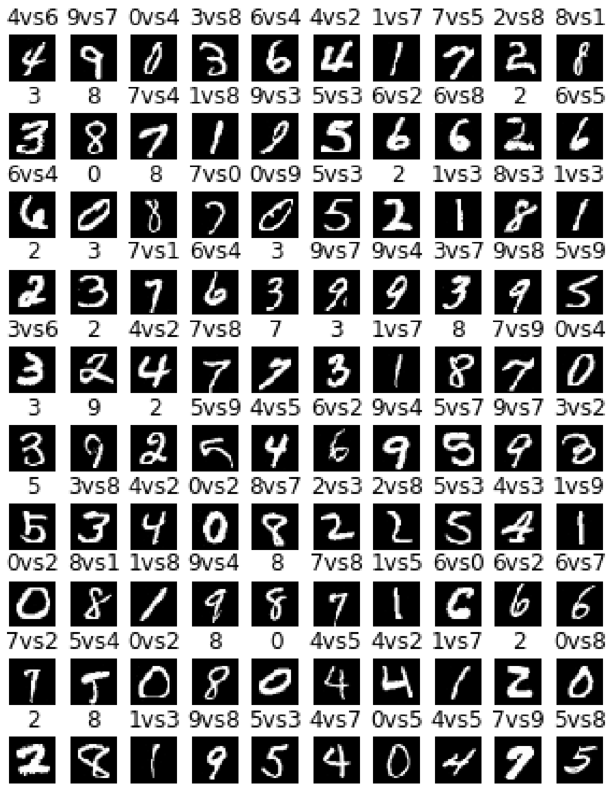
(GBDT, Untargeted, Fooling rate of 89%)


4 9 0 3 6vs8 4 1 7 2 8
3 8 7 1 9vs8 5vs8 6vs8 6vs8 2 6vs8
6vs8 0 8 7 0 5vs8 2 1 8 1
2 3 7 6vs8 3 9 9 3 9 5vs8
3 2 4 7 7 3 1 8 7 0
3 9 2 5vs8 4 6vs8 9 5vs8 9 3
5vs8 3 4 0 8 2 2 5vs8 4 1
0 8 1 9 8 7 1 6vs8 6vs8 6vs8
7 5vs8 0 8 0 4 4 1 2 0
2 8 1 9 5vs8 4 0 4 7 5vs8

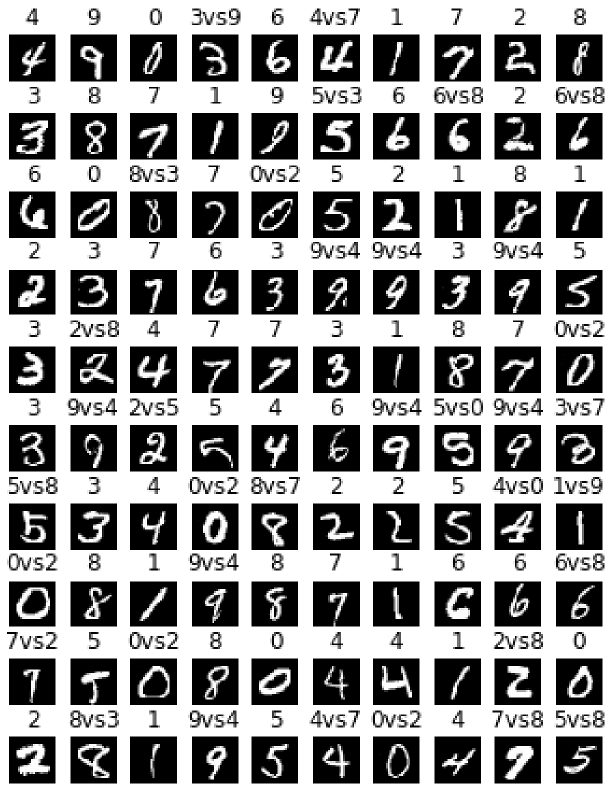

4vs8 9vs8 0vs8 3vs8 6vs8 4vs8 1vs8 7vs8 2vs8 8
3vs8 8 7vs8 1vs8 9vs8 5vs8 6vs8 6vs8 2vs8 6vs8
6vs8 0vs8 8 7vs8 0vs8 5vs8 2vs8 1vs8 8 1vs8
2vs8 3vs8 7vs8 6vs8 3vs8 9vs8 9vs8 3vs8 9vs8 5vs8
3vs8 2vs8 4vs8 7vs8 7vs8 3vs8 1vs8 8 7vs8 0vs8
3vs8 9vs8 2vs8 5vs8 4vs8 6vs8 9vs8 5vs8 9vs8 3vs8
5vs8 3vs8 4vs8 0vs8 8 2vs8 2vs8 5vs8 4vs8 1vs8
0vs8 8 1vs8 9vs8 8 7vs8 1vs8 6vs8 6vs8 6vs8
7vs8 5vs8 0vs8 8 0vs8 4vs8 4vs8 1vs8 2vs8 0vs8
2vs8 8 1vs8 9vs8 5vs8 4vs8 0vs8 4vs8 7vs8 5vs8

4.1 MNIST: Adversarial examples generated by ART ZOO(ℓ_2) attack against Decision Trees.

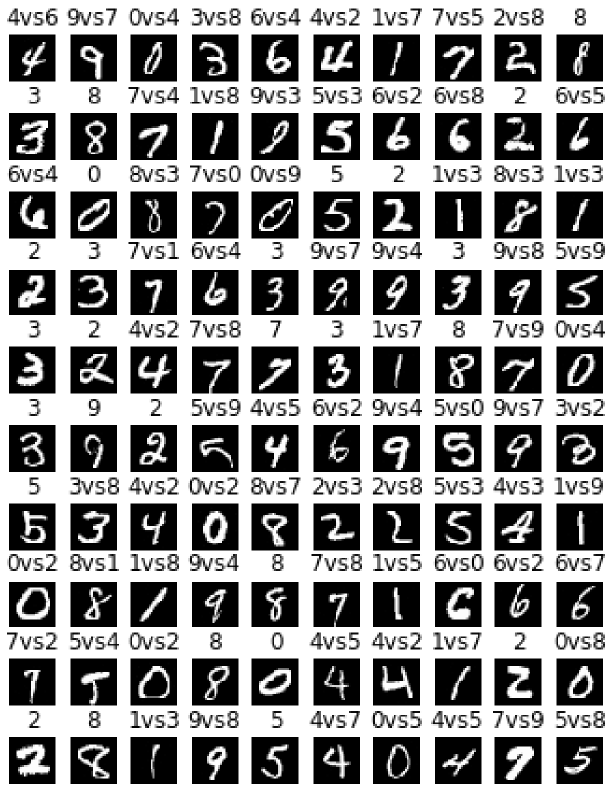
(Untargeted, $\varepsilon = 0.1$, 83% fooling rate)


4vs6 9vs7 0vs4 3vs8 6vs4 4vs2 1vs7 7vs5 2vs8 8vs1
3 8 7vs4 1vs8 9vs3 5vs3 6vs2 6vs8 2 6vs5
6vs4 0 8 7vs0 0vs9 5vs3 2 1vs3 8vs3 1vs3
2 3 7vs1 6vs4 3 9vs7 9vs4 3vs7 9vs8 5vs9
3vs6 2 4vs2 7vs8 7 3 1vs7 8 7vs9 0vs4
3 9 2 5vs9 4vs5 6vs2 9vs4 5vs7 9vs7 3vs2
5 3vs8 4vs2 0vs2 8vs7 2vs3 2vs8 5vs3 4vs3 1vs9
0vs2 8vs1 1vs8 9vs4 8 7vs8 1vs5 6vs0 6vs2 6vs7
7vs2 5vs4 0vs2 8 0 4vs5 4vs2 1vs7 2 0vs8
2 8 1vs3 9vs8 5vs3 4vs7 0vs5 4vs5 7vs9 5vs8

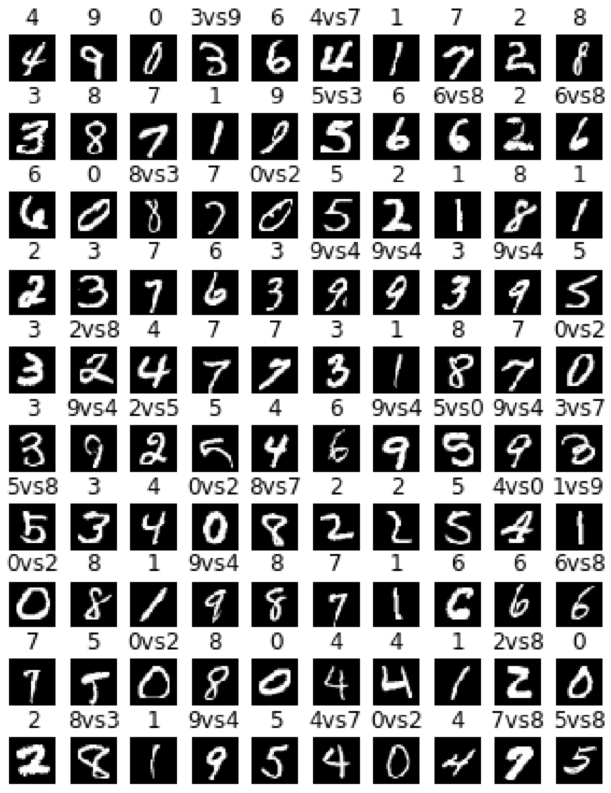
(Targeted, $\varepsilon = 0.1$, 26% fooling rate)


4 9 0 3vs9 6 4vs7 1 7 2 8
3 8 7 1 9 5vs3 6 6vs8 2 6vs8
6 0 8vs3 7 0vs2 5 2 1 8 1
2 3 7 6 3 9vs4 9vs4 3 9vs4 5
3 2vs8 4 7 7 3 1 8 7 0vs2
3 9vs4 2vs5 5 4 6 9vs4 5vs0 9vs4 3vs7
5vs8 3 4 0vs2 8vs7 2 2 5 4vs0 1vs9
0vs2 8 1 9vs4 8 7 1 6 6 6vs8
7vs2 5 0vs2 8 0 4 4 1 2vs8 0
2 8vs3 1 9vs4 5 4vs7 0vs2 4 7vs8 5vs8

(Untargeted, $\varepsilon = 0.3$, 76% fooling rate)

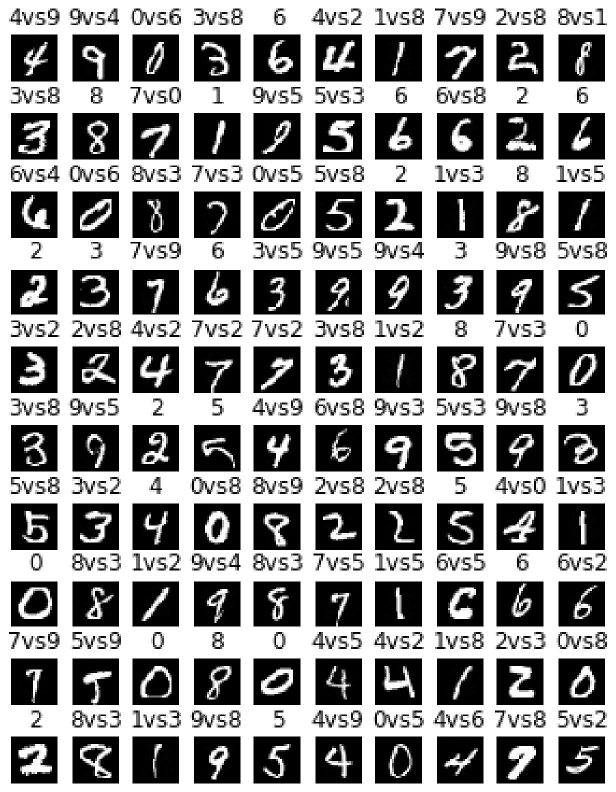

4vs6 9vs7 0vs4 3vs8 6vs4 4vs2 1vs7 7vs5 2vs8 8
3 8 7vs4 1vs8 9vs3 5vs3 6vs2 6vs8 2 6vs5
6vs4 0 8vs3 7vs0 0vs9 5 2 1vs3 8vs3 1vs3
2 3 7vs1 6vs4 3 9vs7 9vs4 3 9vs8 5vs9
3 2 4vs2 7vs8 7 3 1vs7 8 7vs9 0vs4
3 9 2 5vs9 4vs5 6vs2 9vs4 5vs0 9vs7 3vs2
5 3vs8 4vs2 0vs2 8vs7 2vs3 2vs8 5vs3 4vs3 1vs9
0vs2 8vs1 1vs8 9vs4 8 7vs8 1vs5 6vs0 6vs2 6vs7
7vs2 5vs4 0vs2 8 0 4vs5 4vs2 1vs7 2 0vs8
2 8 1vs3 9vs8 5 4vs7 0vs5 4vs5 7vs9 5vs8

(Targeted, $\varepsilon = 0.3$, 25% fooling rate)

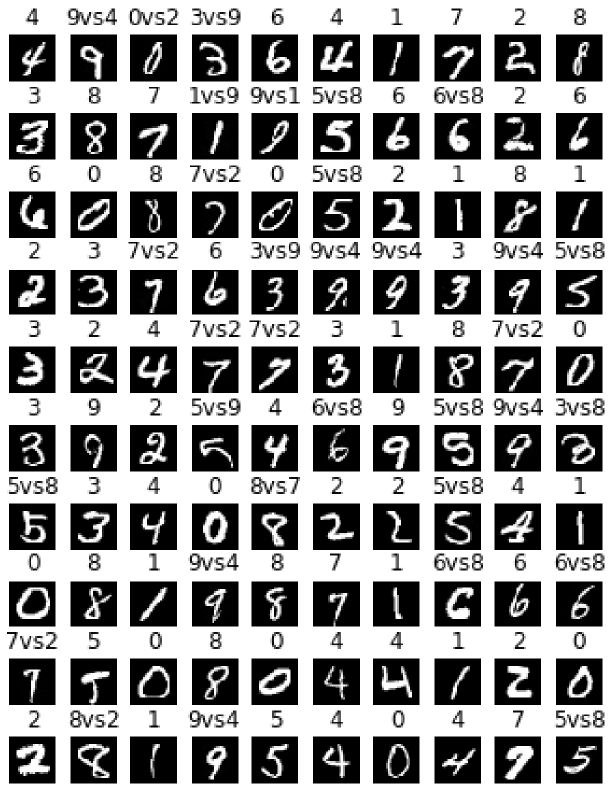

4 9 0 3vs9 6 4vs7 1 7 2 8
3 8 7 1 9 5vs3 6 6vs8 2 6vs8
6 0 8vs3 7 0vs2 5 2 1 8 1
2 3 7 6 3 9vs4 9vs4 3 9vs4 5
3 2vs8 4 7 7 3 1 8 7 0vs2
3 9vs4 2vs5 5 4 6 9vs4 5vs0 9vs4 3vs7
5vs8 3 4 0vs2 8vs7 2 2 5 4vs0 1vs9
0vs2 8 1 9vs4 8 7 1 6 6 6vs8
7 5 0vs2 8 0 4 4 1 2vs8 0
2 8vs3 1 9vs4 5 4vs7 0vs2 4 7vs8 5vs8

4.2 MNIST: Adversarial examples generated by ART ZOO(ℓ_2) attack against Random Forest.

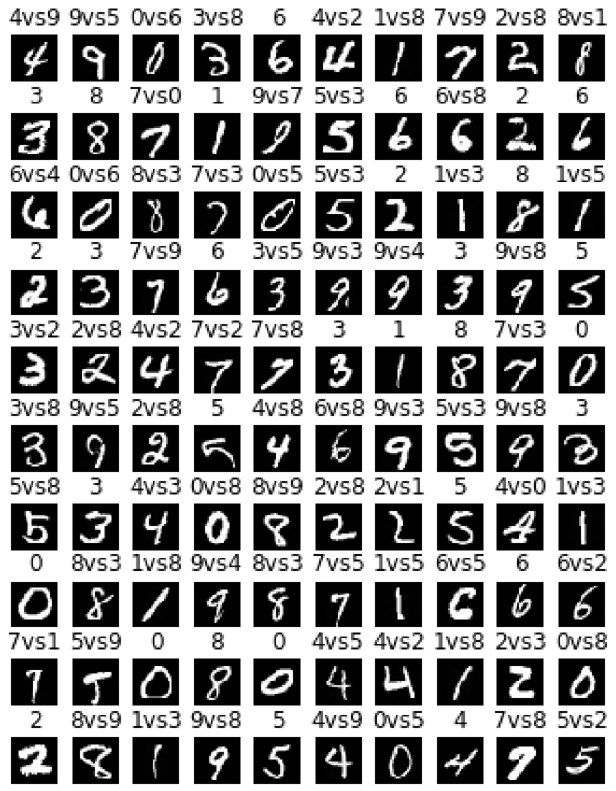
(Untargeted, $\varepsilon = 0.1$, 76% fooling rate)


4vs9 9vs4 0vs6 3vs8 6 4vs2 1vs8 7vs9 2vs8 8vs1
3vs8 8 7vs0 1 9vs5 5vs3 6 6vs8 2 6
6vs4 0vs6 8vs3 7vs3 0vs5 5vs8 2 1vs3 8 1vs5
2 3 7vs9 6 3vs5 9vs5 9vs4 3 9vs8 5vs8
3vs2 2vs8 4vs2 7vs2 7vs2 3vs8 1vs2 8 7vs3 0
3vs8 9vs5 2 5 4vs9 6vs8 9vs3 5vs3 9vs8 3
5vs8 3vs2 4 0vs8 8vs9 2vs8 2vs8 5 4vs0 1vs3
0 8vs3 1vs2 9vs4 8vs3 7vs5 1vs5 6vs5 6 6vs2
7vs9 5vs9 0 8 0 4vs5 4vs2 1vs8 2vs3 0vs8
2 8vs3 1vs3 9vs8 5 4vs9 0vs5 4vs6 7vs8 5vs2

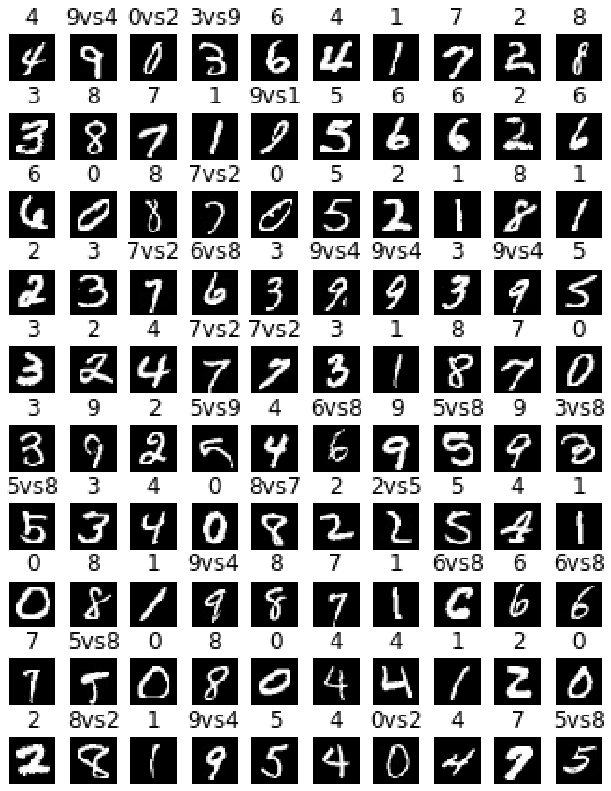
(Targeted, $\varepsilon = 0.1$, 28.3% fooling rate)


4 9vs4 0vs2 3vs9 6 4 1 7 2 8
3 8 7 1vs9 9vs1 5vs8 6 6vs8 2 6
6 0 8 7vs2 0 5vs8 2 1 8 1
2 3 7vs2 6 3vs9 9vs4 9vs4 3 9vs4 5vs8
3 2 4 7vs2 7vs2 3 1 8 7vs2 0
3 9 2 5vs9 4 6vs8 9 5vs8 9vs4 3vs8
5vs8 3 4 0 8vs7 2 2 5vs8 4 1
0 8 1 9vs4 8 7 1 6vs8 6 6vs8
7vs2 5 0 8 0 4 4 1 2 0
2 8vs2 1 9vs4 5 4 0 4 7 5vs8

(Untargeted, $\varepsilon = 0.3$, 72% fooling rate)

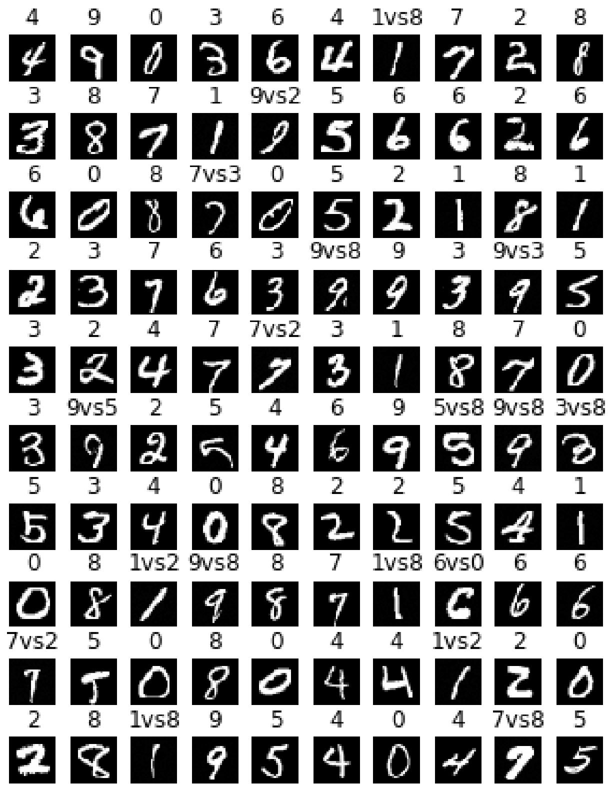

4vs9 9vs5 0vs6 3vs8 6 4vs2 1vs8 7vs9 2vs8 8vs1
3 8 7vs0 1 9vs7 5vs3 6 6vs8 2 6
6vs4 0vs6 8vs3 7vs3 0vs5 5vs3 2 1vs3 8 1vs5
2 3 7vs9 6 3vs5 9vs3 9vs4 3 9vs8 5
3vs2 2vs8 4vs2 7vs2 7vs8 3 1 8 7vs3 0
3vs8 9vs5 2vs8 5 4vs8 6vs8 9vs3 5vs3 9vs8 3
5vs8 3 4vs3 0vs8 8vs9 2vs8 2vs1 5 4vs0 1vs3
0 8vs3 1vs8 9vs4 8vs3 7vs5 1vs5 6vs5 6 6vs2
7vs1 5vs9 0 8 0 4vs5 4vs2 1vs8 2vs3 0vs8
2 8vs9 1vs3 9vs8 5 4vs9 0vs5 4 7vs8 5vs2

(Targeted, $\varepsilon = 0.3$, 23% fooling rate)

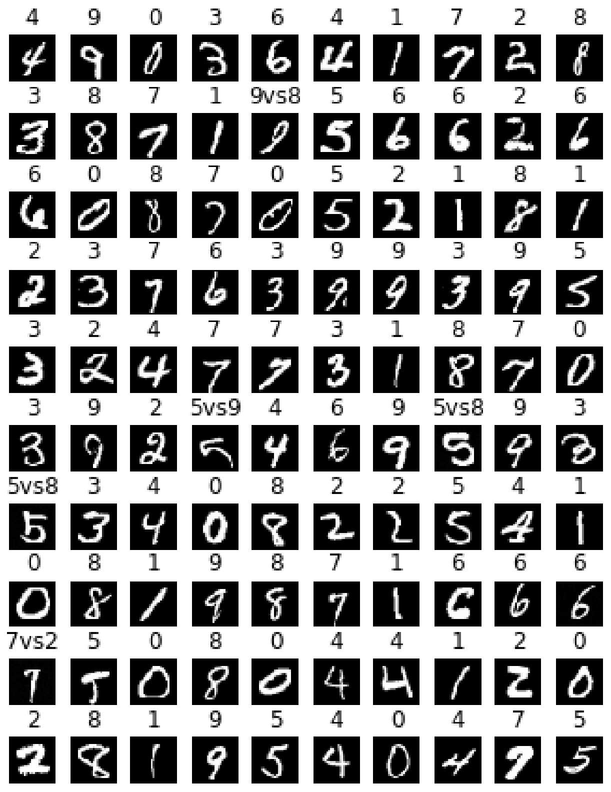

4 9vs4 0vs2 3vs9 6 4 1 7 2 8
3 8 7 1 9vs1 5 6 6 2 6
6 0 8 7vs2 0 5 2 1 8 1
2 3 7vs2 6vs8 3 9vs4 9vs4 3 9vs4 5
3 2 4 7vs2 7vs2 3 1 8 7 0
3 9 2 5vs9 4 6vs8 9 5vs8 9 3vs8
5vs8 3 4 0 8vs7 2 2vs5 5 4 1
0 8 1 9vs4 8 7 1 6vs8 6 6vs8
7 5vs8 0 8 0 4 4 1 2 0
2 8vs2 1 9vs4 5 4 0vs2 4 7 5vs8

4.3 MNIST: Adversarial examples generated by ART ZOO(ℓ_2) attack against GBDT.

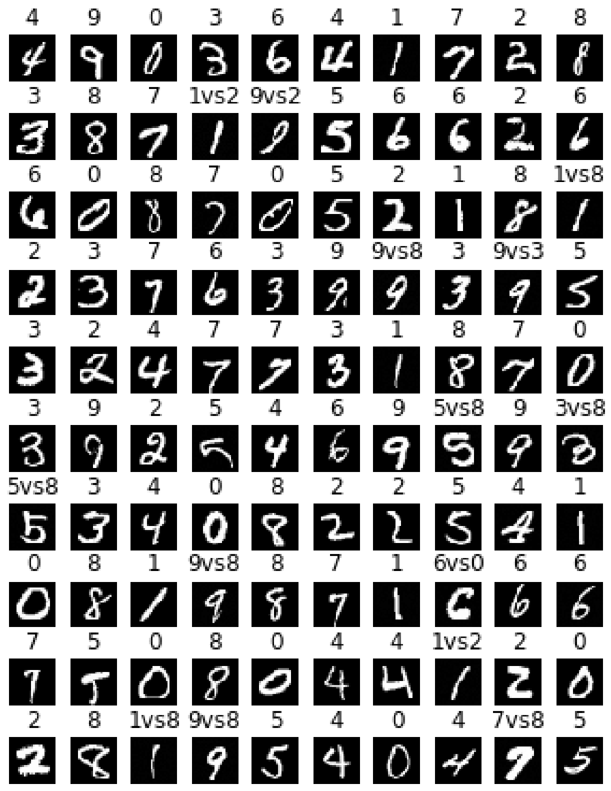
(Untargeted, $\varepsilon = 0.1$, 19% fooling rate)


4 9 0 3 6 4 1vs8 7 2 8
3 8 7 1 9vs2 5 6 6 2 6
6 0 8 7vs3 0 5 2 1 8 1
2 3 7 6 3 9vs8 9 3 9vs3 5
3 2 4 7 7vs2 3 1 8 7 0
3 9vs5 2 5 4 6 9 5vs8 9vs8 3vs8
5 3 4 0 8 2 2 5 4 1
0 8 1vs2 9vs8 8 7 1vs8 6vs0 6 6
7vs2 5 0 8 0 4 4 1vs2 2 0
2 8 1vs8 9 5 4 0 4 7vs8 5

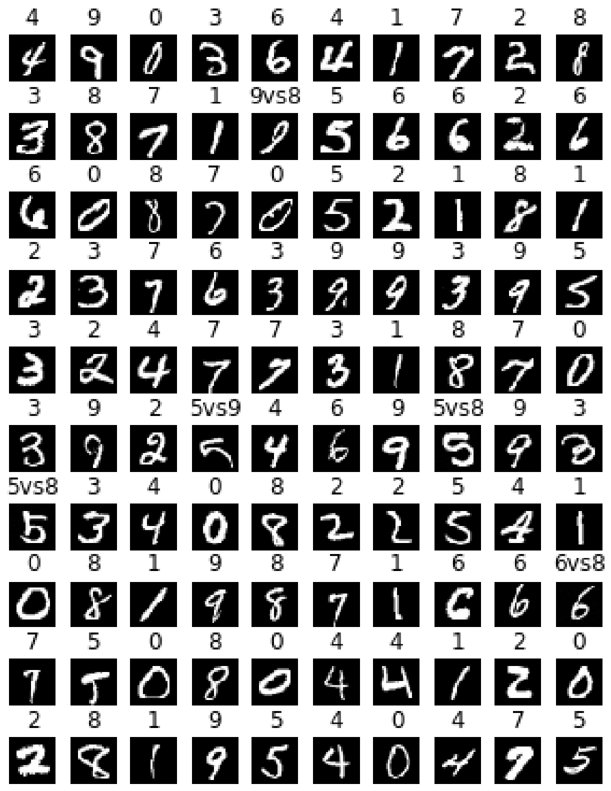
(Targeted, $\varepsilon = 0.1$, 3% fooling rate)



(Untargeted, $\varepsilon = 0.3$, 15% fooling rate)

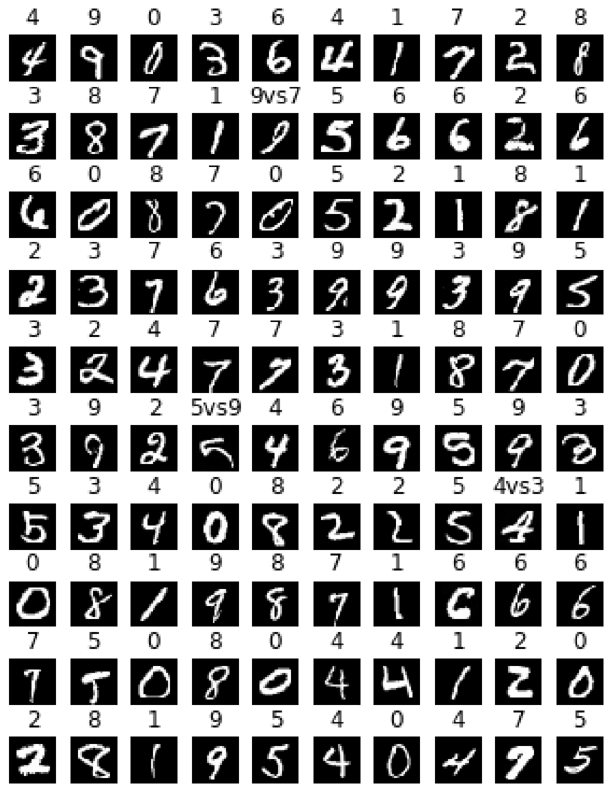

4 9 0 3 6 4 1 7 2 8
3 8 7 1vs2 9vs2 5 6 6 2 6
6 0 8 7 0 5 2 1 8 1vs8
2 3 7 6 3 9 9vs8 3 9vs3 5
3 2 4 7 7 3 1 8 7 0
3 9 2 5 4 6 9 5vs8 9 3vs8
5vs8 3 4 0 8 2 2 5 4 1
0 8 1 9vs8 8 7 1 6vs0 6 6
7 5 0 8 0 4 4 1vs2 2 0
2 8 1vs8 9vs8 5 4 0 4 7vs8 5

(Targeted, $\varepsilon = 0.3$, 3% fooling rate)

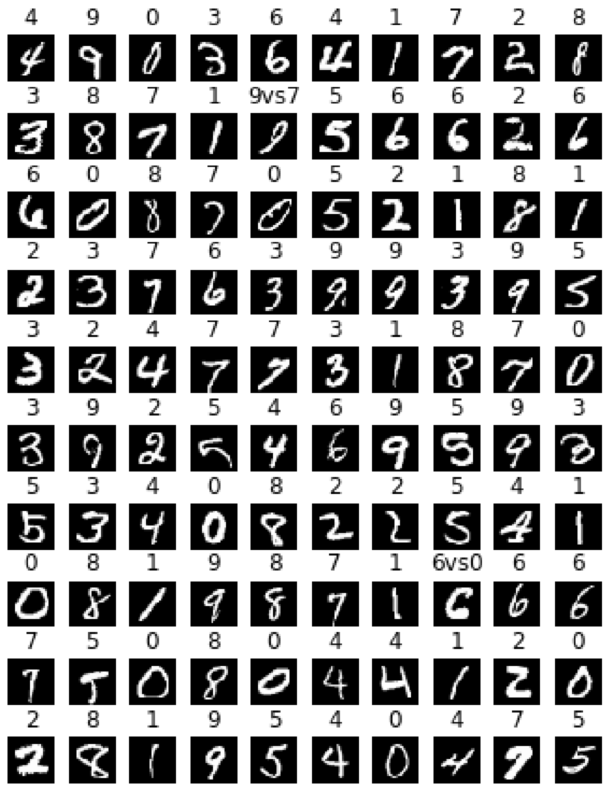


4.4 MNIST: Adversarial examples generated by ART ZOO(ℓ_2) attack against SVM.

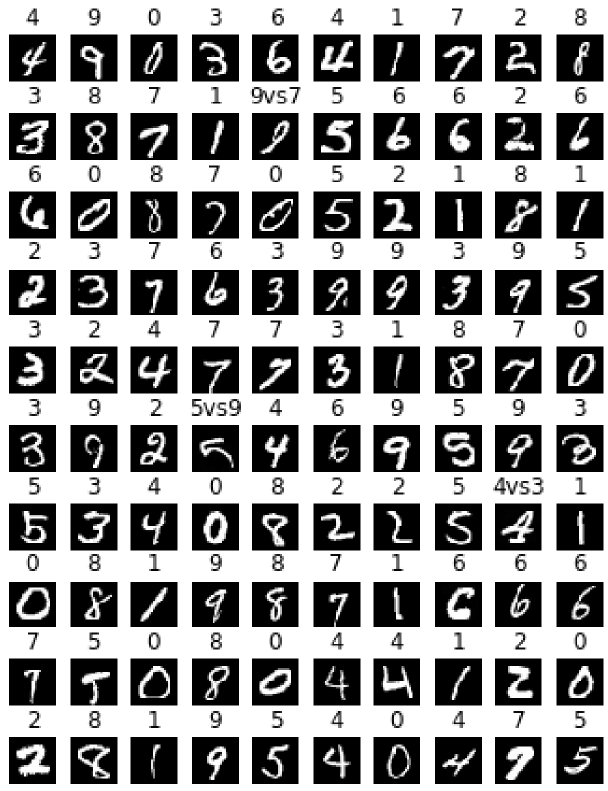
(Untargeted, $\varepsilon = 0.1$, 0.3% fooling rate)



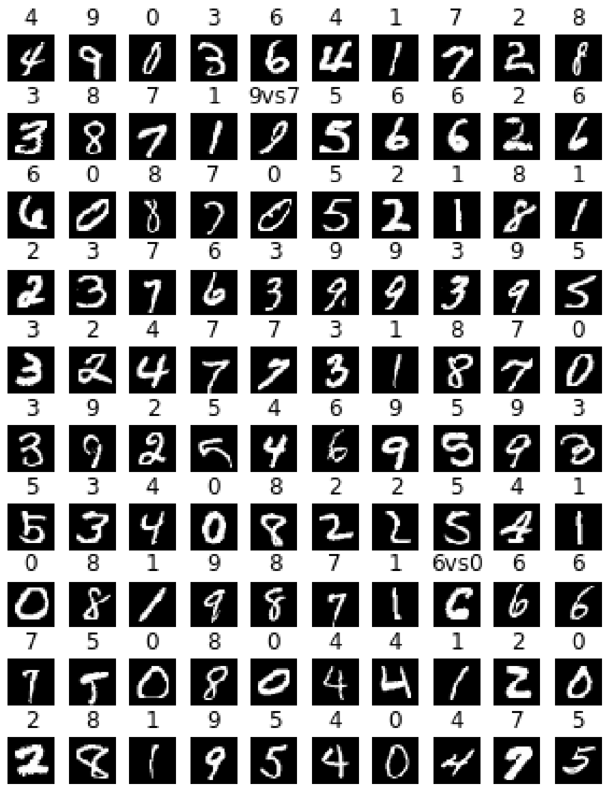
(Targeted, $\varepsilon = 0.1$, 0% fooling rate)



(Untargeted, $\varepsilon = 0.3$, % fooling rate)

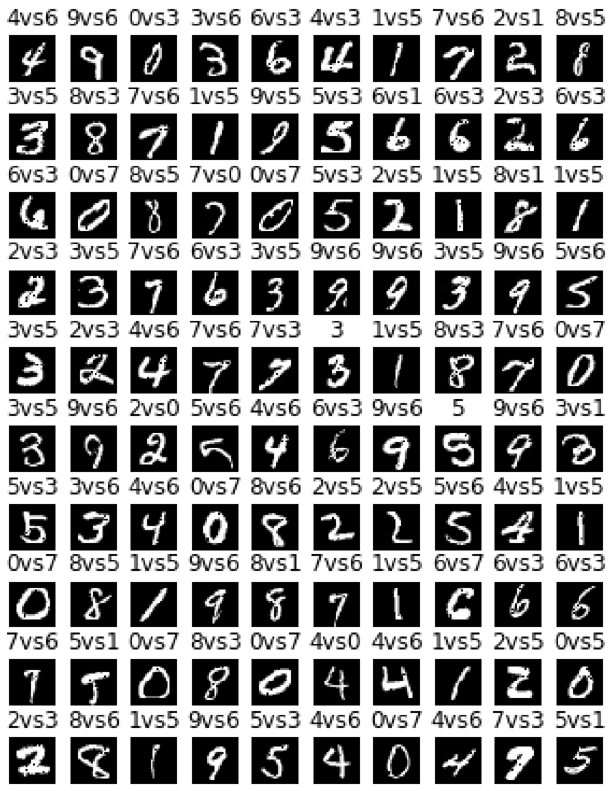


(Targeted, $\varepsilon = 0.3$, 0% fooling rate)

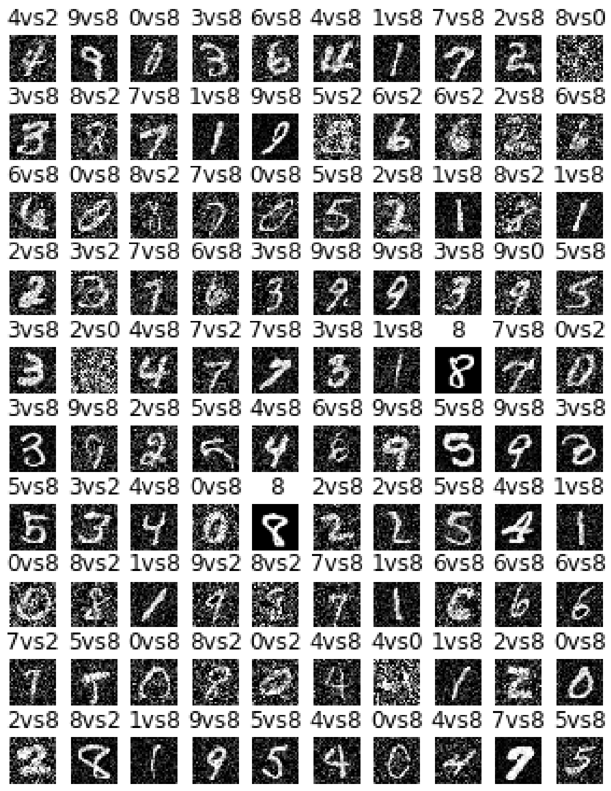


MNIST: Adversarial examples generated by ART Decision Tree, Boundary and HopSkipJump(HSJA), zerothorder optimization attack(ZOO) attacks against decision trees in the untargeted setting.

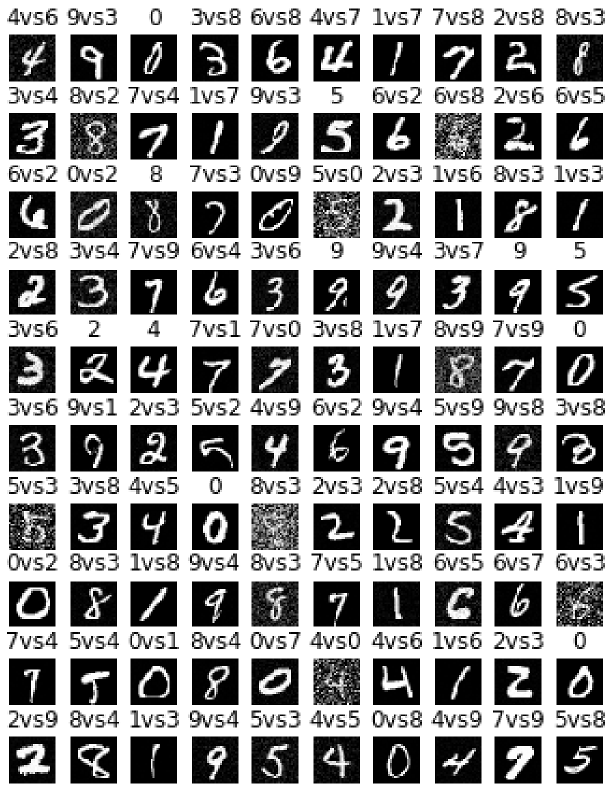
(ART DecisionTree attack, 100% fooling rate)


4vs6 9vs6 0vs3 3vs6 6vs3 4vs3 1vs5 7vs6 2vs1 8vs5
3vs5 8vs3 7vs6 1vs5 9vs5 5vs3 6vs1 6vs3 2vs3 6vs3
6vs3 0vs7 8vs5 7vs0 0vs7 5vs3 2vs5 1vs5 8vs1 1vs5
2vs3 3vs5 7vs6 6vs3 3vs5 9vs6 9vs6 3vs5 9vs6 5vs6
3vs5 2vs3 4vs6 7vs6 7vs3 3 1vs5 8vs3 7vs6 0vs7
3vs5 9vs6 2vs0 5vs6 4vs6 6vs3 9vs6 5 9vs6 3vs1
5vs3 3vs6 4vs6 0vs7 8vs6 2vs5 2vs5 5vs6 4vs5 1vs5
0vs7 8vs5 1vs5 9vs6 8vs1 7vs6 1vs5 6vs7 6vs3 6vs3
7vs6 5vs1 0vs7 8vs3 0vs7 4vs0 4vs6 1vs5 2vs5 0vs5
2vs3 8vs6 1vs5 9vs6 5vs3 4vs6 0vs7 4vs6 7vs3 5vs1

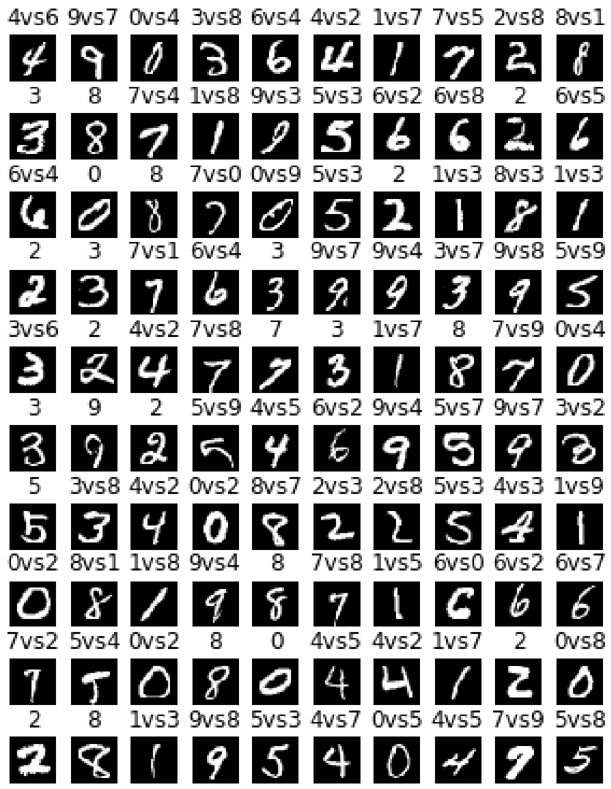
(ART Boundary attack, 100% fooling rate)


4vs2 9vs8 0vs8 3vs8 6vs8 4vs8 1vs8 7vs8 2vs8 8vs0
3vs8 8vs2 7vs8 1vs8 9vs8 5vs2 6vs2 6vs2 2vs8 6vs8
6vs8 0vs8 8vs2 7vs8 0vs8 5vs8 2vs8 1vs8 8vs2 1vs8
2vs8 3vs2 7vs8 6vs8 3vs8 9vs8 9vs8 3vs8 9vs0 5vs8
3vs8 2vs0 4vs8 7vs2 7vs8 3vs8 1vs8 8 7vs8 0vs2
3vs8 9vs8 2vs8 5vs8 4vs8 6vs8 9vs8 5vs8 9vs8 3vs8
5vs8 3vs2 4vs8 0vs8 8 2vs8 2vs8 5vs8 4vs8 1vs8
0vs8 8vs2 1vs8 9vs2 8vs2 7vs8 1vs8 6vs8 6vs8 6vs8
7vs2 5vs8 0vs8 8vs2 0vs2 4vs8 4vs0 1vs8 2vs8 0vs8
2vs8 8vs2 1vs8 9vs8 5vs8 4vs8 0vs8 4vs8 7vs8 5vs8

(ART HSJA(ℓ_2) attack, 92% fooling rate)


4vs6 9vs3 0 3vs8 6vs8 4vs7 1vs7 7vs8 2vs8 8vs3
3vs4 8vs2 7vs4 1vs7 9vs3 5 6vs2 6vs8 2vs6 6vs5
6vs2 0vs2 8 7vs3 0vs9 5vs0 2vs3 1vs6 8vs3 1vs3
2vs8 3vs4 7vs9 6vs4 3vs6 9 9vs4 3vs7 9 5
3vs6 2 4 7vs1 7vs0 3vs8 1vs7 8vs9 7vs9 0
3vs6 9vs1 2vs3 5vs2 4vs9 6vs2 9vs4 5vs9 9vs8 3vs8
5vs3 3vs8 4vs5 0 8vs3 2vs3 2vs8 5vs4 4vs3 1vs9
0vs2 8vs3 1vs8 9vs4 8vs3 7vs5 1vs8 6vs5 6vs7 6vs3
7vs4 5vs4 0vs1 8vs4 0vs7 4vs0 4vs6 1vs6 2vs3 0
2vs9 8vs4 1vs3 9vs4 5vs3 4vs5 0vs8 4vs9 7vs9 5vs8

(ART ZOO attack, 83% fooling rate)


4vs6 9vs7 0vs4 3vs8 6vs4 4vs2 1vs7 7vs5 2vs8 8vs1
3 8 7vs4 1vs8 9vs3 5vs3 6vs2 6vs8 2 6vs5
6vs4 0 8 7vs0 0vs9 5vs3 2 1vs3 8vs3 1vs3
2 3 7vs1 6vs4 3 9vs7 9vs4 3vs7 9vs8 5vs9
3vs6 2 4vs2 7vs8 7 3 1vs7 8 7vs9 0vs4
3 9 2 5vs9 4vs5 6vs2 9vs4 5vs7 9vs7 3vs2
5 3vs8 4vs2 0vs2 8vs7 2vs3 2vs8 5vs3 4vs3 1vs9
0vs2 8vs1 1vs8 9vs4 8 7vs8 1vs5 6vs0 6vs2 6vs7
7vs2 5vs4 0vs2 8 0 4vs5 4vs2 1vs7 2 0vs8
2 8 1vs3 9vs8 5vs3 4vs7 0vs5 4vs5 7vs9 5vs8