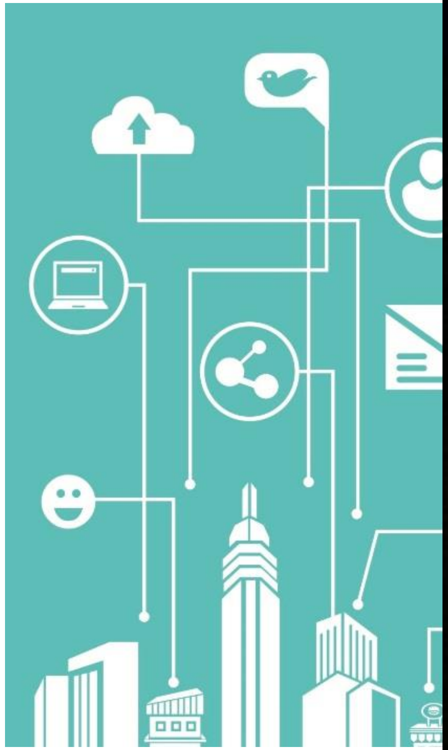
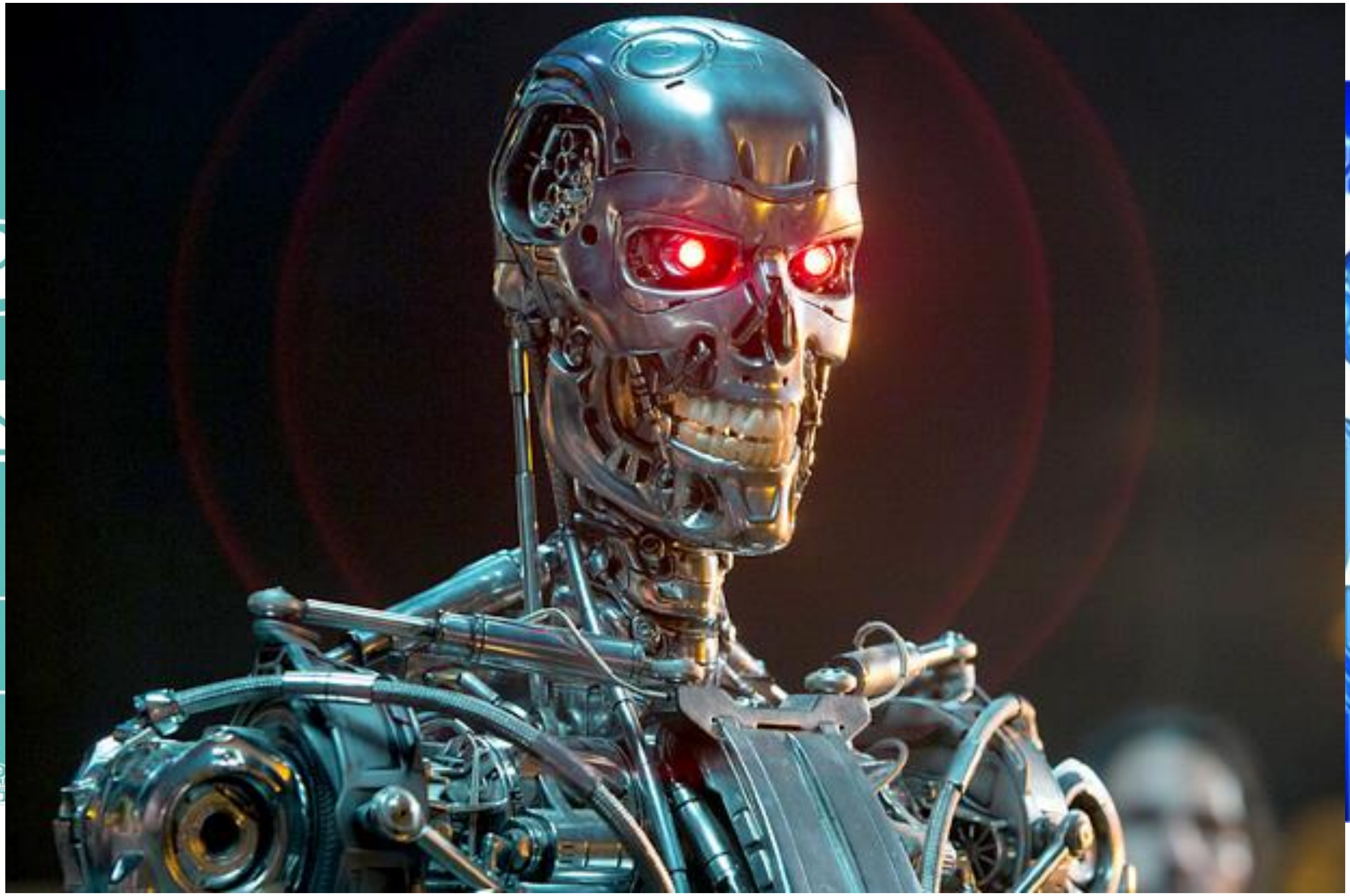
Poisoning Attacks through Back-Gradient Optimization

Luis Muñoz-González

RA Symposium
London, June 2017





The Security of Machine Learning

Machine Learning systems can be compromised:

- Proliferation and sophistication of attacks and threats.
- Machine learning systems are one of the weakest parts in the security chain.
- Attackers can also use machine learning as a weapon.

Adversarial Machine Learning:

- Security of machine learning algorithms.
- Understanding the weaknesses of the algorithms.
- Proposing more resilient techniques.

Threats

Evasion Attacks:

- Attacks at test time.
- The attacker aims to find the blind spots and weaknesses of the ML system to evade it.

Poisoning Attacks:

- Compromise data collection.
- The attacker subverts the learning process.
- Degrades the performance of the system.
- Can facilitate future evasion.

Evasion Attacks

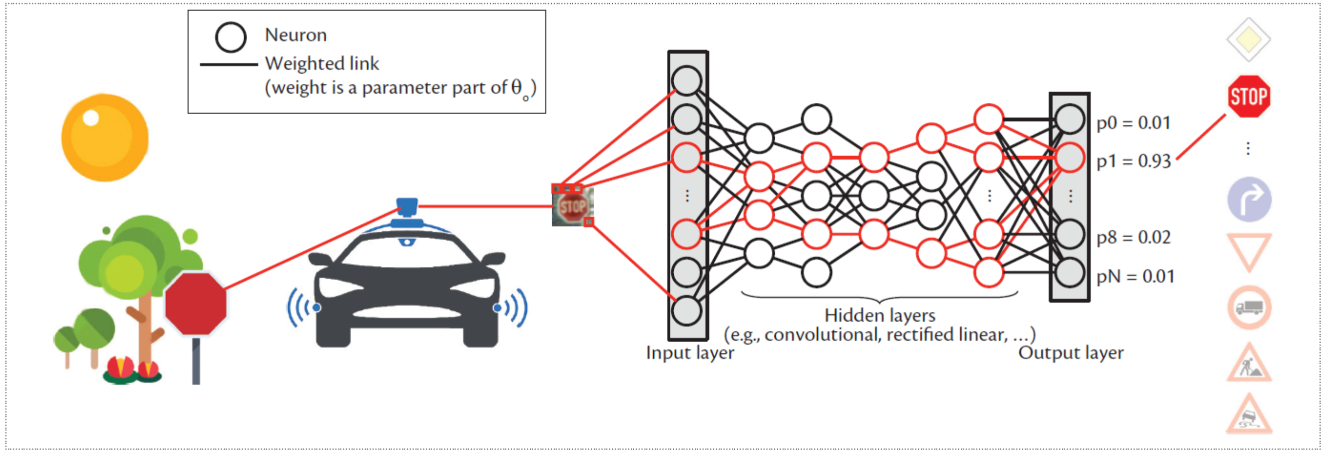


Figure 1. An autonomous vehicle uses a camera to identify and recognize roadside signs. Once a sign has been identified, its image is fed to a neural network for classification in one of the predefined sign classes. Here, the neural network identifies the sign as a stop sign.

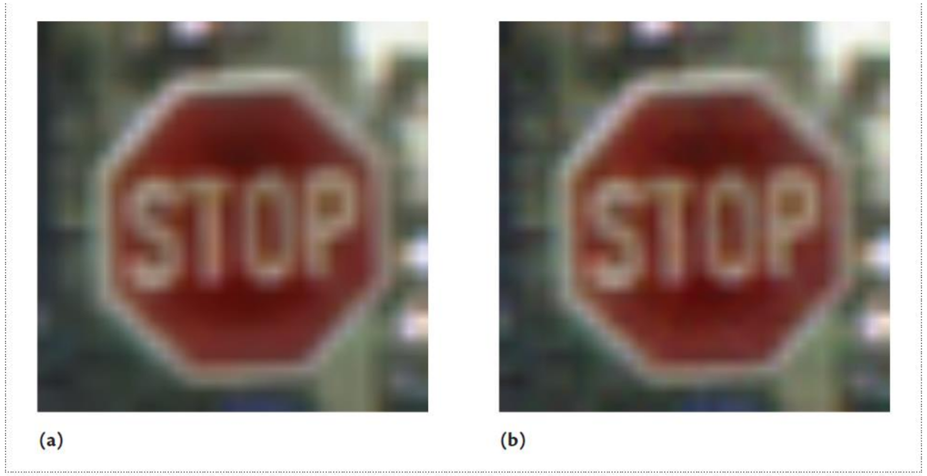
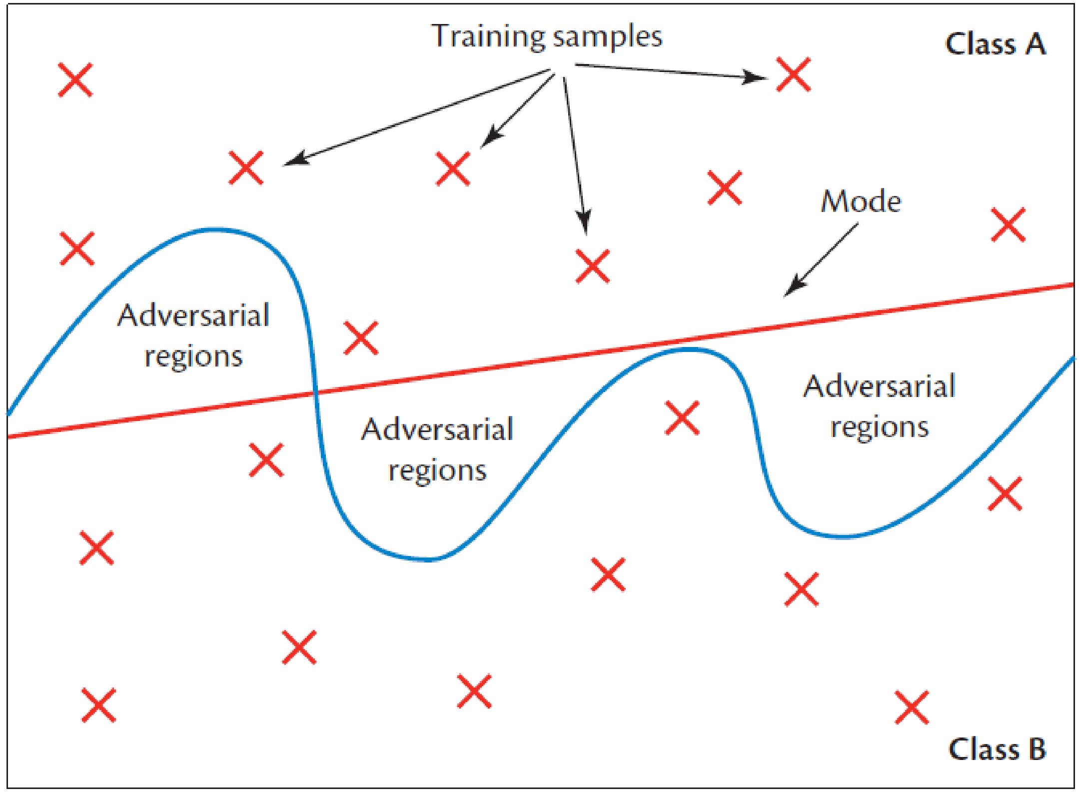
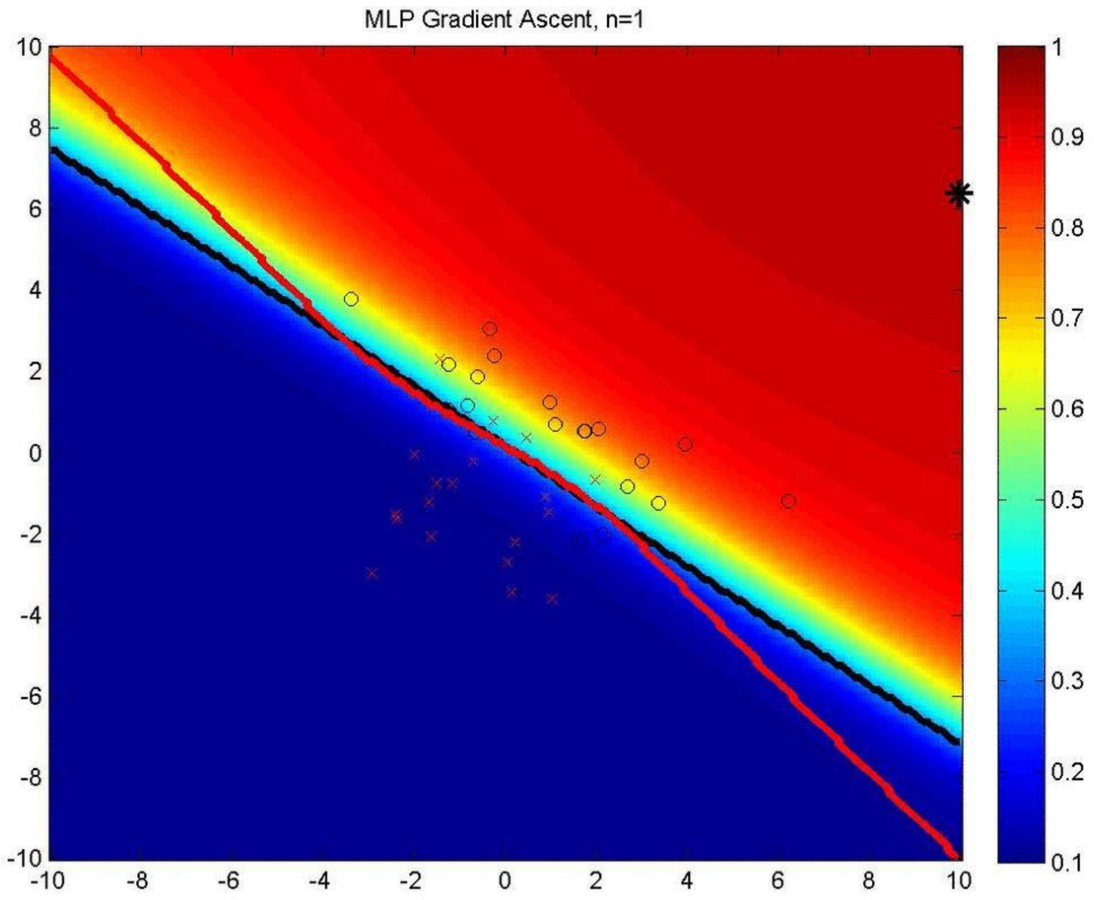


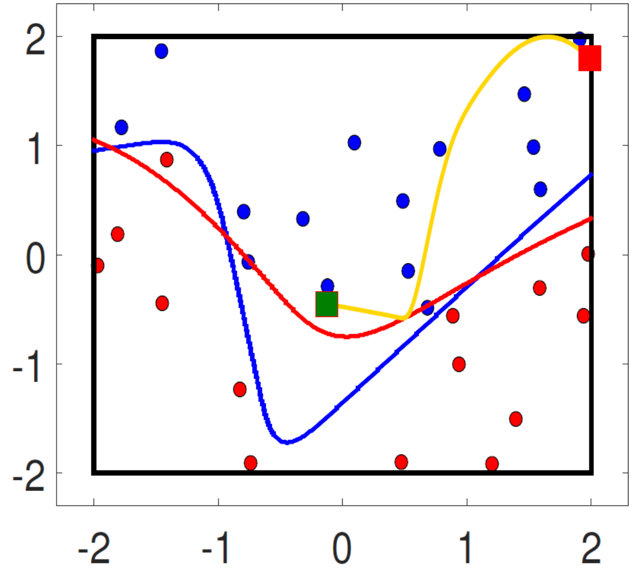
Figure 2. To humans, adversarial samples are indistinguishable from original samples. (a) An ordinary image of a stop sign. (b) An image crafted by an adversary.

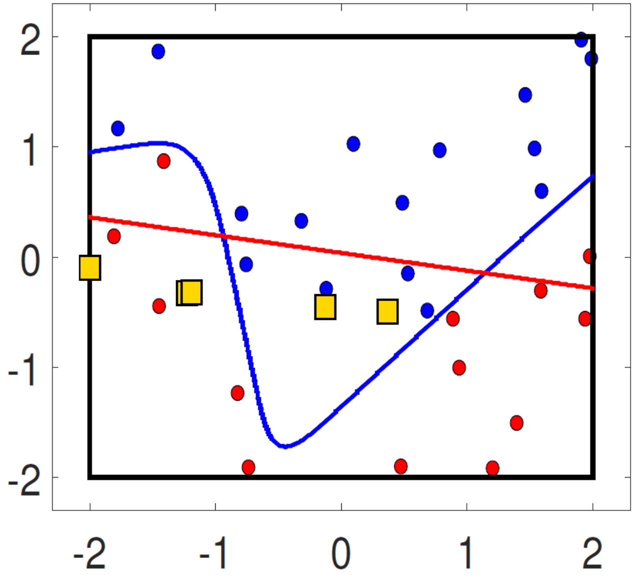


P. McDaniel, N. Papernot, Z.B. Celik. "*Machine Learning in Adversarial Settings.*" IEEE Security & Privacy, 14(3), pp. 68-72, 2016.

Poisoning Attacks







Optimal Poisoning Attacks

General formulation of the problem:

- The attacker aims to optimize some objective function (evaluated on a validation dataset) by introducing malicious examples in the training dataset used by the defender.
- The defender aims to learn the parameters of the model that optimise some objective function evaluated on the (poisoned) training dataset.
- The attacker's problem can be modelled as a **bi-level optimization problem**:

$$\mathcal{D}_p^* \in \arg\max_{\mathcal{D}_p} \; \mathcal{A}_{\text{val}}(\mathbf{w}^*, \mathcal{D}_{\text{val}}) \,,$$

$$\text{s.t.} \quad \mathbf{w}^* \in \arg\min_{\mathbf{w}} \; \mathcal{C}_{\text{tr}}(\mathbf{w}, \mathcal{D}_{\text{tr}} \cup \mathcal{D}_p)$$

Optimal Poisoning Attacks for Classification

$$\mathbf{x_p}^* \in \arg\max_{\mathbf{x_p}\in\mathcal{X}} \mathcal{C}_{\text{val}}(\mathbf{w}^*),$$
$$\text{s.t. } \mathbf{w}^* \in \arg\min_{\mathbf{w}} \mathcal{C}_{\text{tr}}(\mathbf{w}, \mathcal{D}_{\text{tr}} \cup \{\mathbf{x_p}, y_p\})$$

- Biggio et al. "*Poisoning Attacks against Support Vector Machines*." ICML 2012.
- Mei and Zhu. "*Using Machine Teaching to Identify Optimal Training-Set Attacks on Machine Learners*." AAAI 2015.
- Xiao et al. "*Is Feature Selection Secure against Training Data Poisoning?*" ICML 2015.

- Poisoning points are learned following a **gradient ascent** strategy: $\nabla_{\mathbf{x_p}} \mathcal{C}_{\text{val}}(\mathbf{w}^*) = \left(\frac{\partial \mathbf{w}}{\partial \mathbf{x_p}}\right)^T \nabla_{\mathbf{w}} \mathcal{C}_{\text{val}}(\mathbf{w}^*)$
- Applying **Karush-Kuhn-Tucker** conditions $\nabla_{\mathbf{w}} \mathcal{C}_{\text{tr}}(\mathbf{w}, \mathbf{x_p}) = \mathbf{0}$ and **the implicit function theorem**:

$$\nabla_{\mathbf{x_p}}\mathcal{C}_{\text{val}} = -(\nabla_{\mathbf{x_p}}\nabla_{\mathbf{w}}\mathcal{C}_{\text{tr}})(\nabla^2_{\mathbf{w}}\mathcal{C}_{\text{tr}})^{-1}\nabla_{\mathbf{w}}\mathcal{C}_{\text{val}}$$

- Limited to a **restricted family of classifiers**.
- **Poor scalability** with the number of parameters of the model.

Optimal Poisoning Attacks for Classification

More efficient solution:

1) Don't invert matrices, use **conjugate gradient** instead:
   - More Stable.
   - Allows avoiding the computation of the Hessian.
2) **Divide and Conquer**:
   - Instead of computing $\nabla_{\mathbf{x_p}} \mathcal{C}_{\text{val}} = -(\nabla_{\mathbf{x_p}} \nabla_{\mathbf{w}} \mathcal{C}_{\text{tr}})(\nabla^2_{\mathbf{w}} \mathcal{C}_{\text{tr}})^{-1} \nabla_{\mathbf{w}} \mathcal{C}_{\text{val}}$
   - Compute: $\nabla^2_{\mathbf{w}} \mathcal{C}_{\text{tr}} \; \mathbf{v} = \nabla_{\mathbf{w}} \mathcal{C}_{\text{val}}$

$$\nabla_{\mathbf{x_p}} \mathcal{C}_{\text{val}} = -\nabla_{\mathbf{x_p}} \nabla_{\mathbf{w}} \mathcal{C}_{\text{tr}} \; \mathbf{v}$$

3) **Don't compute the Hessian!**

$$\frac{\partial^2 f(\mathbf{u}, \mathbf{v})}{\partial \mathbf{u} \partial \mathbf{v}^T} \mathbf{z} = \lim_{h \to 0} \frac{1}{h} \left(\nabla_{\mathbf{v}} f\left(\mathbf{u} + h\mathbf{z}, \mathbf{v}\right) - \nabla_{\mathbf{v}} f\left(\mathbf{u}, \mathbf{v}\right) \right)$$

$$\frac{\partial^2 f(\mathbf{u}, \mathbf{v})}{\partial \mathbf{u} \partial \mathbf{u}^T} \mathbf{z} = \lim_{h \to 0} \frac{1}{h} \left(\nabla_{\mathbf{u}} f\left(\mathbf{u} + h\mathbf{z}, \mathbf{v}\right) - \nabla_{\mathbf{u}} f\left(\mathbf{u}, \mathbf{v}\right) \right)$$

Poisoning with Back-Gradient Optimization

- J. Domke. "*Generic Methods for Optimization-Based Modelling*." AISTATS 2012.
- D. Maclaurin, D.K. Duvenaud, R.P. Adams. "*Gradient-based Hyperparameter Optimization through Reversible Learning*." ICML 2015.

Algorithm 1 Gradient Descent

Input: initial weights $\mathbf{w}_0$, learning rate α, $\mathcal{D}_{\rm tr}$, loss function $\mathcal{L}(\mathbf{w}, \mathbf{x}, y)$

1: **for** $t = 0, \dots, T-1$ **do**
2: $\quad \mathbf{g}_t = \nabla_{\mathbf{w}} \mathcal{C}_{\rm tr}(\mathbf{w}_t)$
3: $\quad \mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \alpha\, \mathbf{g}_t$
4: **end for**

Output: trained parameters $\mathbf{w}_T$

Algorithm 2 Back-gradient Descent

Input: $\mathbf{w}_T$, α, $\mathcal{L}(\mathbf{w}, \mathbf{x}, y)$, $\mathcal{D}_{\rm tr}$, $\mathcal{D}_{\rm val}$
initialize $d\mathbf{x}_\mathbf{p} \leftarrow \mathbf{0}$, $d\mathbf{w} \leftarrow \nabla_{\mathbf{w}} \mathcal{C}_{\rm val}(\mathbf{w}_T)$

1: **for** $t = T, \dots, 1$ **do**
2: $\quad d\mathbf{x}_\mathbf{p} \leftarrow d\mathbf{x}_\mathbf{p} - \alpha\, d\mathbf{w} \nabla_{\mathbf{w}} \nabla_{\mathbf{x}_\mathbf{p}} \mathcal{C}_{\rm tr}(\mathbf{w}_t, \mathbf{x}_\mathbf{p})$
3: $\quad d\mathbf{w} \leftarrow d\mathbf{w} - \alpha\, d\mathbf{w} \nabla_{\mathbf{w}} \nabla_{\mathbf{w}} \mathcal{C}_{\rm tr}(\mathbf{w}_t, \mathbf{x}_\mathbf{p})$
4: $\quad \mathbf{g}_{t-1} = \nabla_{\mathbf{w}_t} \mathcal{C}_{\rm tr}(\mathbf{w}_t, \mathbf{x}_\mathbf{p})$
5: $\quad \mathbf{w}_{t-1} = \mathbf{w}_t + \alpha\, \mathbf{g}_{t-1}$
6: **end for**

Output: $\nabla_{\mathbf{x}_\mathbf{p}} \mathcal{C}_{\rm val} \leftarrow d\mathbf{x}_\mathbf{p}$

Greedy Attack Strategy

Algorithm 3 Greedy Poisoning Attack

Input: $\mathcal{D}_{\text{tr}}$, $\mathcal{D}_{\text{val}}$, iterations gradient descent T, set of initial poisoning points $\{\mathbf{x}_{p_j}^{(0)}, y_{p_j}\}_{j=0}^{n_p}$, grad. ascent learning rate β, small positive constant ε

initialize $\mathcal{D}_p \leftarrow \{\emptyset\}$, $\hat{\mathcal{D}}_{\text{tr}} \leftarrow \mathcal{D}_{\text{tr}}$

for $j = 1, \dots, n_p$ **do**
    $i \leftarrow 0$
    repeat
        $\mathbf{w}_T \leftarrow$ Gradient Descent on $\hat{\mathcal{D}}_{\text{tr}}$ (T iterations)
        $\nabla_{\mathbf{x}_{p_j}} \mathcal{C}_{\text{val}}(\mathbf{x}_{p_j}^{(i)}, y_{p_j}) \leftarrow$ back-grad. descent with $\mathbf{w}_T$ (Alg. 2)
        $\mathbf{x_p}_j^{(i+1)} \leftarrow \Pi_{\mathcal{X}}(\mathbf{x_p}_j^{(i)} + \beta \nabla_{\mathbf{x}_{p_j}} \mathcal{C}_{\text{val}})$
        $i \leftarrow i + 1$
    until $\mathcal{C}_{\text{val}}(\mathbf{x_p}_j^{(i)}) - \mathcal{C}_{\text{val}}(\mathbf{x_p}_j^{(i-1)}) < \varepsilon$
    $\hat{\mathcal{D}}_{\text{tr}} \leftarrow \hat{\mathcal{D}}_{\text{tr}} \cup (\mathbf{x}_{p_j}^{(i)}, y_{p_j})$
    $\mathcal{D}_p \leftarrow \mathcal{D}_p \cup (\mathbf{x}_{p_j}^{(i)}, y_{p_j})$
end for

Output: set of poisoning points $\mathcal{D}_p$

- Learn one poisoning point at a time.
- Performance comparable to coordinated attack strategies.

Types of Poisoning Attacks

$$\mathbf{x_p}^* \in \arg\max_{\mathbf{x_p} \in \mathcal{X}} \; \mathcal{C}_{\mathrm{val}}(\mathbf{w}^*) \,,$$

$$\text{s.t.} \;\; \mathbf{w}^* \in \arg\min_{\mathbf{w}} \; \mathcal{C}_{\mathrm{tr}}(\mathbf{w}, \mathcal{D}_{\mathrm{tr}} \cup \{\mathbf{x_p}, y_p\})$$

Attacker's Objective:

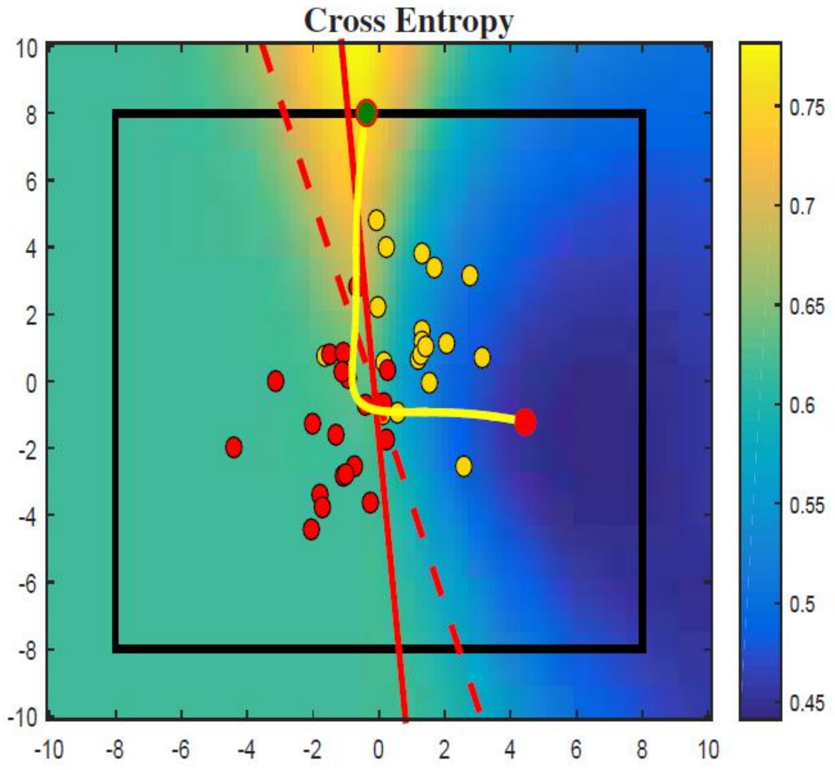
The attacker's cost function $\mathcal{C}_{\mathrm{val}}$ determines the objective of the attack:

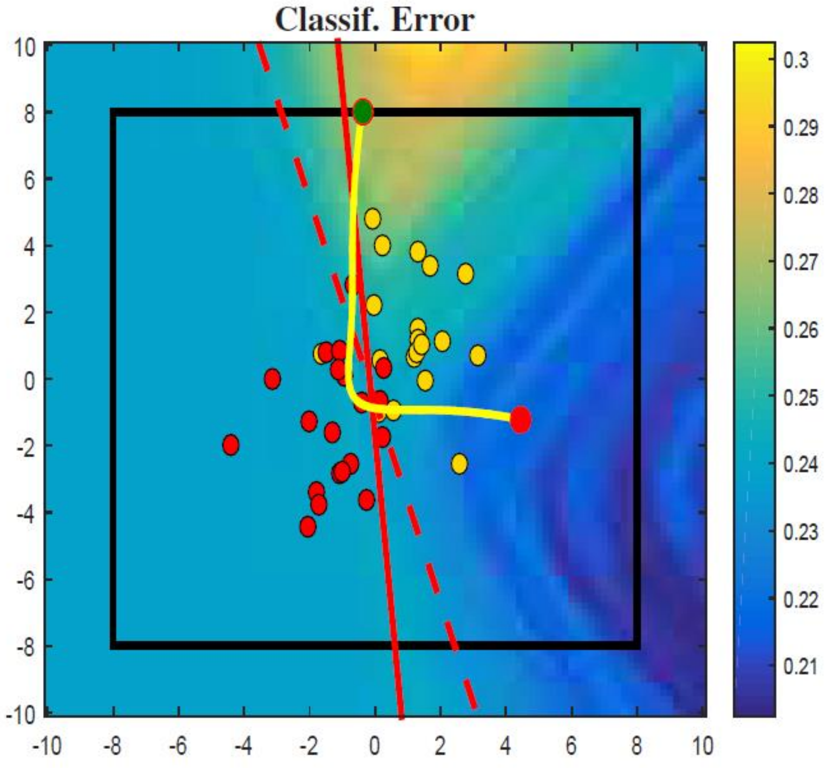
- **Targeted Attacks**: the attacker aims to cause some concrete error: particular classes, instances or features to be selected/discarded by the learning algorithm.
- **Indiscriminate Attacks**: the attacker aims to increase the overall classification error.

Attacker's Capabilities:

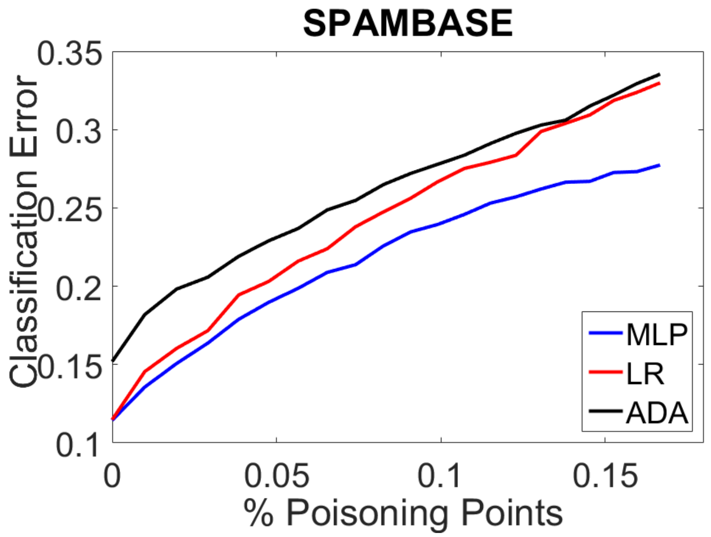
- The **labels** of the poisoning points y_p determine the attacker capabilities.
- Different modes: **insertion**, modification, deletion.
- Attacker's capabilities also have an **impact on the attacker objective**.
- **Full knowledge** vs Partial Knowledge.

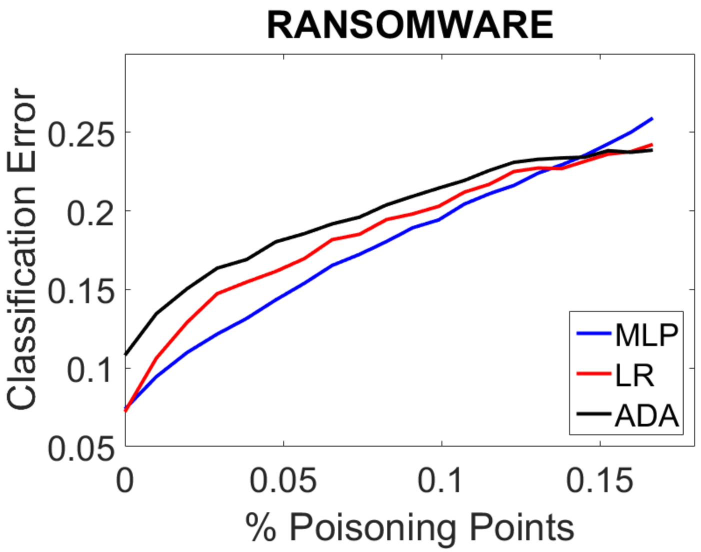
Synthetic Example

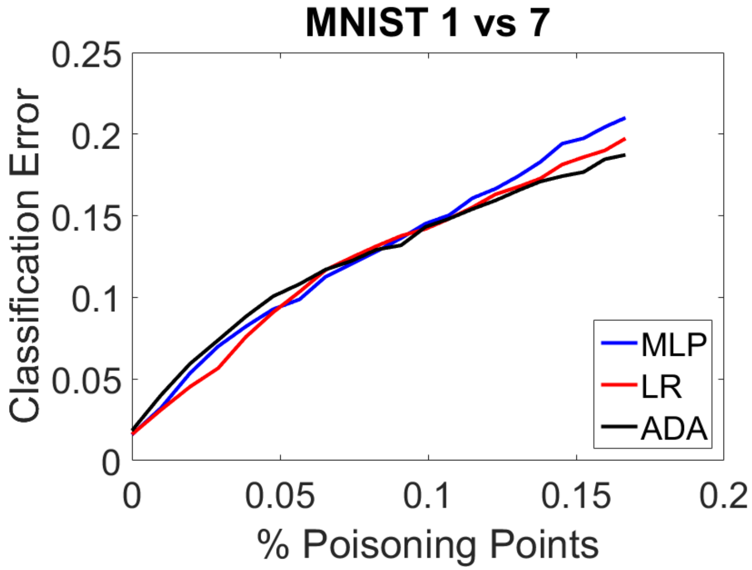




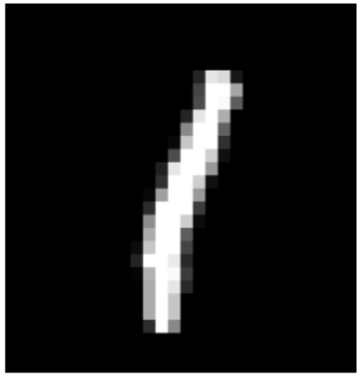
Indiscriminate Attacks against Binary Classifiers



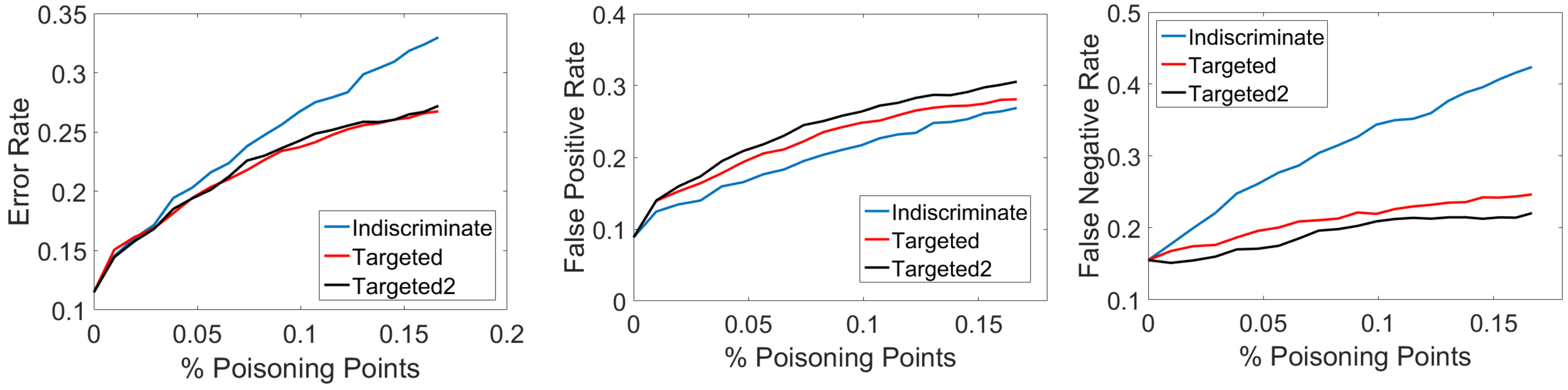




- **Spambase**: Spam filtering application (54 features)
- **Ransomware**: Malware detection (400 features)
- **MNIST 1vs 7**: Computer vision (784 features, 28 x 28)



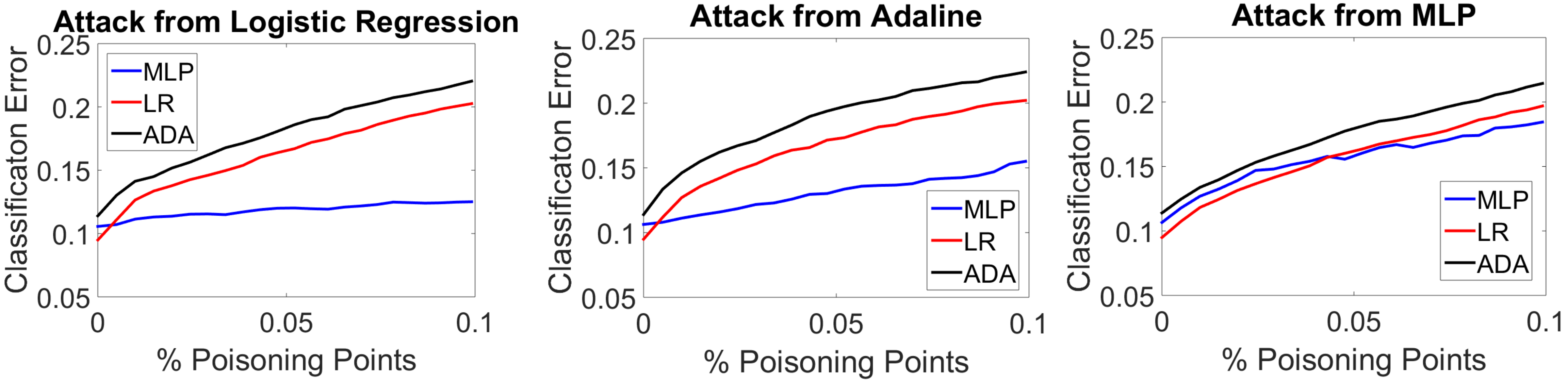
Targeted vs Indiscriminate Attacks



- Spambase dataset, Logistic Regression.

Attack	Labels of poisoning points	Attacker's objective function
Indiscriminate	Positive and negative	Cross Entropy
Targeted	Positive	Cross Entropy
Targeted 2	Positive	Cross Entropy only on positive samples

Transferability

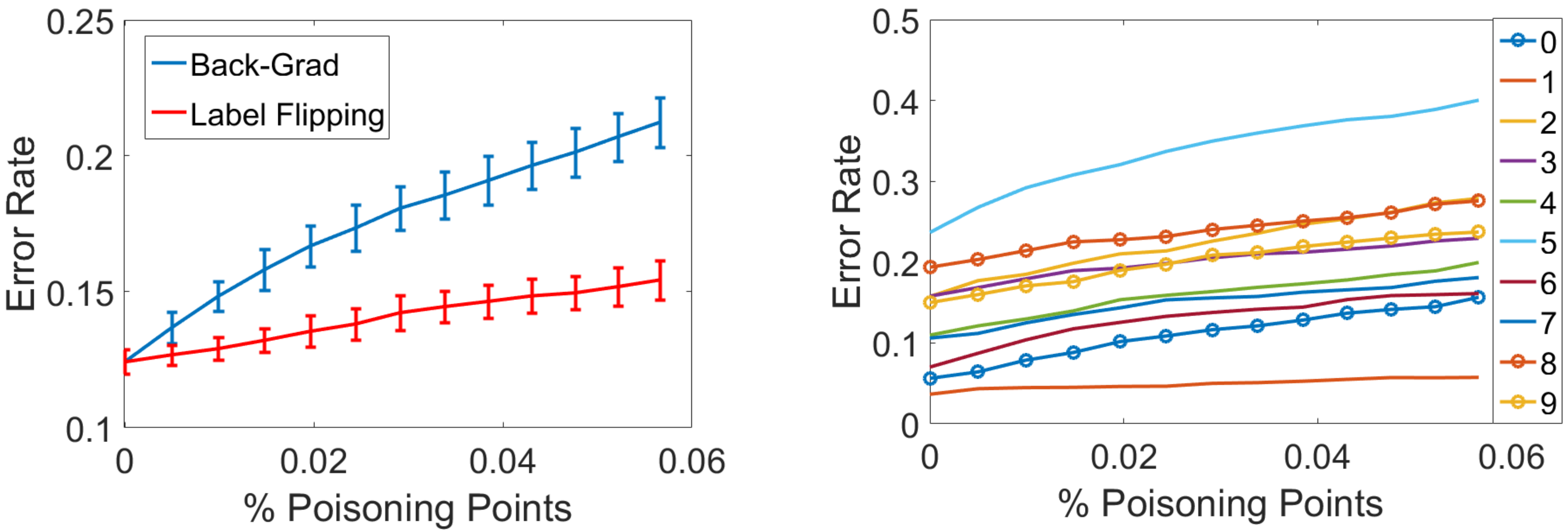


- Spambase dataset
- Attack points between linear classifiers are transferable
- Attack points generated from the non-linear classifier are transferable to linear classifiers

Poisoning Multi-Class Classifiers

Indiscriminate Attack:

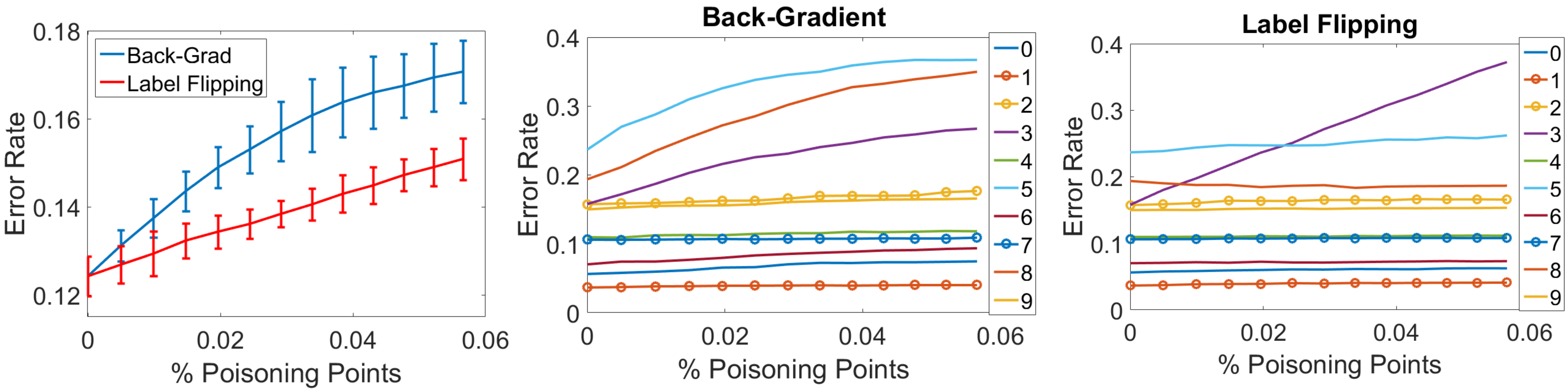
- MNIST dataset, Multi-class Logistic regression
- Selection of initial poisoning points: at random from the validation set, and then, flip the label randomly
- Comparison with random label flipping



Poisoning Multi-Class Classifiers

Targeted Attack:

- MNIST dataset, Multi-class Logistic regression
- Selection of initial poisoning points: at random from samples of digit 3, then, flip the label to 8.
- Comparison with random label flipping (flipping the labels of samples from digit 3 to 8).



Summary

- Machine learning algorithms are **vulnerable to data poisoning**.
- Optimal Poisoning Attacks can be modelled as **bi-level optimization problems**.
- **Back-Gradient optimization** is efficient to compute poisoning points:
  - Better scalability
  - No KKT conditions required: can be applied to a broader range of algorithms
- **Transferability**: poisoning points generated to attack one particular algorithm can also be harmful to other algorithms.
- Interrelation between the **attacker's capabilities and objective**.
- **Ongoing work**: Deep Networks, Hyperparameters.

Collaborators

Imperial College London

RISS Resilient Information Systems Security

Università di Cagliari

Pattern Recognition and Applications Lab

- Dr Emil C. Lupu
- Andrea Paudice
- Vasin Wongrassamee
- Kenneth Co
- Lisa Tse

- Prof Fabio Roli
- Dr Battista Biggio
- Ambra Demontis

Thank you!

Contact: l.munoz@imperial.ac.uk
Room 502 (Huxley Building)