YOU ARE NOT ALLOWED TO SHARE THE CONTENT WITH OTHERS OR DISSEMINATE THE CONTENT

NUS CS-CS5562: Trustworthy Machine Learning August 31, 2022

Assignment 2

Lecturer: Reza Shokri

The assignment contains the following five parts.

1. **Warm up:** Implement the label flipping poisoning attack.
2. **Task 1: Poisoning attack**
3. **Task 2: Robustness via data sanitization**
4. **Task 3: Design your adaptive attack**
5. **Task 4: Robust gradient descent algorithm**

You are asked to complete the code in **designated** places in the Jupyter Notebooks. You are **not** allowed to change any other parts of the notebook unless you are told to do so or you are running your own test cases. You also need to write a short report in the Notebook for all the four graded tasks.

- You are required to submit a `.zip` file containing the Jupyter notebooks only. You should name the zip file after your matriculation number, e.g. `A0123456X.zip`.

Setting up the environment

This section is duplicated from that in Assignment 1. If you have set up your local environment for Assignment 1, you can reuse it.

All the Jupyter notebooks can be run on Google Colab * or your local machines.

Google Colab

To use Google Colab, you need to first upload the entire assignment folder to your Google Drive. After launching individual notebooks, you can choose to use GPU as your runtime. The instructions to mount your Google Drive is included in the notebook. You should be able to run all the code without installing any packages in Colab.

Local Machine

Although we use mostly standard libraries in the assignment, there is a chance your local environment is missing one or two packages or have a version mismatch. We strongly recommend using Anaconda to install and manage the packages used in this assignment. Once Anaconda is installed, go to the assignment's root directory, where you will see `environment.yaml`. Install all the required packages by typing this command into the terminal

```
conda env create --file environment.yml
```

You should have an environment called `cs5562` with all packages needed to run the code. Switch to this environment by

```
conda activate cs5562
```

If installing all packages takes too long on your machine, you can specify this before creating the environment:

```
conda config --set channel_priority flexible
```

*https://colab.research.google.com/

Warm up: Label flipping attack

In the lecture, you have learned a few data poisoning attack algorithms. In untargeted poisoning attack, the objective of the adversary is as follows:

$$\begin{aligned} &\max_{D_p} L(\hat{\theta}, D_t) \\ &\text{where } \hat{\theta} = \arg\min_{\theta} L(\theta, D_c \cup D_p), \\ &\text{s.t., } |D_p| = \epsilon |D_c| \end{aligned} \tag{1}$$

where $L(\theta, D) := \frac{1}{|D|} \sum_{(x,y)\in D} \ell(\theta; x, y)$ is the average loss of the model h_θ on the dataset D and ϵ is the contamination fraction reflecting the power of the adversary. The adversary aims to find a poisoned dataset D_p to increase the average test loss of the model (loss on test set D_t) trained on clean training dataset D_c and poisoned dataset D_p.

In this warm up task, you need to implement the poisoning attack in the simplest setting, where the adversary can only modify the labels. In other words, the features in D_p are clean (without any adversarial noise), but the labels can be modified. The adversary is given the structure of the target model `target_model`, the clean training dataset `clean_dataset` of size n, and the test dataset `test_dataset`. The goal is to generate a poisoned dataset of size ϵn.

Implementation details

In this task, you will conduct poisoning attacks on two datasets, MNIST-17 dataset and dog-fish dataset. MNIST-17 dataset only contains images with class 1 and class 7 from the MNIST dataset (the label is -1 and 1, respectively). Dogfish dataset only contains dog and fish images (the label is -1 and 1, respectively).

You are given the `target_model`, the `test_dataset`, the `clean_dataset` and the contamination fraction `eps`.

1. **Random label flipping attack:** In this subtask, you need to implement a random label flipping attack. The adversary can add ϵn points to the training dataset and change the labels for those data points. Given a clean training dataset and the test dataset, the naive strategy for the adversary is to randomly select ϵn points from the clean training dataset or test dataset, flip their labels, and add them to the training dataset. You need the implement the algorithm in the `attack` function of the `Random_Label_Flip_Attack` class in the warm-ups notebook.

2. **Your label flipping attack:** In this subtask, you need to design your own label flipping attack for SVM models to further reduce the test accuracy. Instead of randomly selecting points from the clean dataset or test dataset, you can try to find the most "influential" points and flip their labels. Please implement your algorithm in `attack` function of the `Label_Flip_Attack` class in the warm-ups notebook.

Task 1: Poisoning Attacks

In this task, you are still working with SVM models. Different from the previous task, the adversary can now change the features and labels to launch poisoning attacks.

First, let's recall the optimization problem in Equation (1). This is a bi-level optimization problem that is usually hard to solve. The difficulty stems from how the optimization variable D_p only affects the objective (test loss $L(\theta, D_t)$ through the model parameters $\hat{\theta}$, which are themselves a complicated function of D_p.

To solve this problem, one approach is to use influence functions to measure the influence of individual training points on the prediction of the model Koh and Liang [2017]. In this task, you will learn an alternative method for approximately solving this problem, which is more efficient than the influence-function-based poisoning attack. Leveraging this method, you can implement your first attack where the adversary can change the features and labels to generate a poisoning dataset.

KKT Attack (see Koh et al. [2018])

In general, we do not know what poisoned model parameter $\hat{\theta}$ would lead to an attack that is both effective and realizable. However, suppose we do know which $\hat{\theta}$ we are after, the optimization problem of Equation (1) simplifies to finding D_p such that $\hat{\theta} = \arg\min_\theta L(\theta; D_c \cup D_p)$. Based on this, the new attack algorithm decomposes the attack into two parts:

1. Using heuristics to find parameters $\hat{\theta}$ that we want the target (victim) model to learn.

2. Finding poisoned data D_p such that the defender is indeed tricked into learning the decoy parameters $\hat{\theta}$.

Step 2 is easier to solve in general. Specifically, if the loss ℓ is strictly convex and differentiable in θ, we can rewrite the condition

$$\hat{\theta} = \arg\min_\theta L(\theta; D_c \cup D_p) \tag{2}$$

$$= \arg\min_\theta \frac{1}{|D_c \cup D_p|} \left(\sum_{(x,y)\in D_c} \ell(\theta; x, y) + \sum_{(x,y)\in D_p} \ell(\theta; x, y) \right) \tag{3}$$

as the equivalent KKT optimality condition

$$\sum_{(x,y)\in D_c} \nabla_\theta \ell(\hat{\theta}; x, y) + \sum_{(x,y)\in D_p} \nabla_\theta \ell(\hat{\theta}; x, y) = 0. \tag{4}$$

Note that if the loss ℓ is not differentiable, e.g., the hinge loss, you can replace it with a similar subgradient condition.

Implementation details

Given the clean training dataset `clean_dataset` the test dataset `test_dataset`, the structure of the target model `target_model` (SVM model) and the contamination fraction `eps`, you need to implement your KKT attack. You should implement the `attack` function of the `KKT_Attack` class in the Task 1 notebook.

Hints

Finding a good $\hat{\theta}$ is crucial. A good candidate for $\hat{\theta}$ should perform poorly on the test dataset.

Report

Please explain your attack algorithm in the report section of the notebook, specifying how you find $\hat{\theta}$ in Step 1 and whether the victim model trained on the poisoned training dataset has a similar parameter as $\hat{\theta}$ (the parameter you find in Step 1). In addition, you need to provide your answers to the following question in the report:

1. Compare the attack success rates of your KKT attack and your label flipping attack on the SVM models. Which attack has a higher success rate and why?

Task 2: Robustness via Data Sanitization

In this task, you take the role of the defender, and you need to build your defense model against poisoning attacks (including label flipping attacks and KKT attacks). The goal of the defender is to learn a high-accuracy model on the clean test dataset even when the training data is poisoned.

The first idea is to filter out the poisoned data or outliers before feeding the training dataset into the learning algorithm. This is the basic idea of *data sanitization*. However, the dimensionality of the training data is usually large (e.g., 784 for MNIST), which can make it harder for the defender to distinguish the poisoned data from the clean data by checking the images manually (recall that the adversarial examples are hard for us to detect).

To solve this problem, let's make a few observations about the poisoned data you generated in the previous tasks. You can compute the following values:

1. Call `loss_diff` in `utilities.py` to compare the loss distribution of a model on the poisoned dataset D_p and the clean dataset D_c.

2. Call `distance_to_center_diff` in `utilities.py`. The function computes the centroid of the clean data for each class and computes the distance of poisoned data and clean data to the centroids. You can compare the distribution of the distance on D_p and D_c.

Can you distinguish the poisoned data and the clean data from those statistics? Now, based on those two observations, please design your defense model without modifying the learning algorithm. More specifically, you need to sanitize the training dataset (which might be poisoned) to reduce the influence of the poisoned data.

Implementation details

You need to implement your defense model in the function `data_sanitizer` function in the Task 2 notebook. The goal is to filter out the poisoned data in the (potentially poisoned) training dataset `training_data`. Of course, the defender might not know the exact contamination fraction of the poisoned dataset. Instead, the defender can estimate it, which is indicated by the `estimate_eps` parameter. In this task, we assume that the contamination level of the training dataset is the same as the `estimate_eps`. Note that it can be 0.

Report

Please describe your defense algorithm in the report section of the notebook, specifying how you filter out the poisoned data and why it is a good approach. You can also inform us about any difficulties you faced and how you solved them. In addition, you need to provide your answers to the following questions in the report:

1. In the task, we assume that the defender can accurately estimate the contamination fraction of the training dataset. If the estimation is higher than the real contamination fraction, what would happen? What if the estimated contamination fraction is lower than the real fraction?

2. Compare the performance of the defense model in the adversarial setting (the test loss of the poisoned model) on different datasets. Does the defense have the same performance across all datasets? Why?

Task 3: Design your Adaptive Attack

From the lecture and Assignment 1, we hope you have already learned that the robustness of the model against one (or a few) specific attack(s) does not imply its robustness against all unseen attacks. Thus, the defense model without theoretical guarantee should be evaluated against an adaptive attack. Now, let's show the necessity of this principle again. In this task, you again take the role of the attacker, and you need to design your own adaptive attack to break the data sanitization defense.

Implementation details

You are required to implement the `attack` function of the `Adaptive_Attack` class in the Task 3 notebook. Similar to previous tasks, you are still working on the SVM model.

Report

Please describe your adaptive attack algorithm in the report, specifying how your attack bypasses the data sanitization. You can also inform us about any difficulties you faced and how you solved them.

Hint

You can read the paper (Koh et al. [2018]), which includes several attacks to bypass the data sanitization defenses.

Task 4: Robust Gradient Descent

In the previous task, we have seen issues of defense algorithms without theoretical guarantees. Thus, in this task, we shift our attention to provable defense algorithms.

The idea is based on the fact that the poisoned data can affect the learned model through the learning algorithm. Recall the learning algorithm we usually use, i.e., batch gradient descent. In each round, the algorithm selects a mini-batch and computes the gradient of the loss on each data point in the mini-batch with respect to the parameters. Then, the algorithm averages the gradients over all the points in the mini-batch and updates the model.

Instead of filtering out the poisoned data based on raw features, we can try to reduce the influence of the poisoned data on the final model by reducing the influence of its gradient on the aggregate gradient in each round. This is the idea of *robust gradient descent algorithms*. In those algorithms, the average aggregation algorithm is replaced by more robust aggregation algorithms, which are designed to reduce the effect of outliers in the input on aggregate results (see more details in Diakonikolas et al. [2019]) with provable guarantees (e.g., median).

In this task, you need to design your own robust gradient descent algorithm. Your algorithm will be evaluated both in the benign setting (when the training dataset is clean) and in the adversarial setting (when the training dataset is poisoned by different poisoning attacks). Evaluating your algorithm in the benign setting is because that the training dataset might not be poisoned in practice. Thus, you need to ensure that your robust algorithm can still achieve high accuracy when the training dataset is clean.

In addition, you also need to provide provable guarantees for your robust algorithm. As you have learned, defense algorithms without robustness guarantees may fail to be robust against adaptive attacks. Thus, provable robustness guarantees are usually necessary for robust algorithms.

You can make assumptions about the loss function, the target model, etc. However, the assumptions should be realistic. For example, you shouldn't assume that the adversary can not add any data to the training dataset.

Implementation details

You need to implement your robust gradient descent algorithm in the `robust_trainer` function in the Task 4 notebook.

Report

In the report, you need to describe your defense algorithm in detail. Furthermore, please provide the whole proof of your robustness analysis in the report. Note that the proof should be clear and self-contained. Incomplete or unclear analysis will be ignored. In addition, you need to provide your answers to the following question in the report:

1. Please train the robust logistic regression model, robust neural network models using your training algorithm and compare their accuracy in the benign and adversarial settings. Which model achieves a better test accuracy in each case? Why?

Hint

- You can learn the robust algorithm in the paper (Diakonikolas et al. [2019]) in which the high-dimensional robust aggregation algorithm replaces the average to aggregate the gradient in each round.
- Replacing the average with other robust aggregation algorithms (e.g., median, trimmed mean) are also possible solutions.

References

Ilias Diakonikolas, Gautam Kamath, Daniel Kane, Jerry Li, Jacob Steinhardt, and Alistair Stewart. Sever: A robust meta-algorithm for stochastic optimization. In *International Conference on Machine Learning*, pages 1596–1606. PMLR, 2019.

Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *International Conference on Machine Learning*, pages 1885–1894. PMLR, 2017.

Pang Wei Koh, Jacob Steinhardt, and Percy Liang. Stronger data poisoning attacks break data sanitization defenses. *arXiv preprint arXiv:1811.00741*, 2018.