CIS 6930: Privacy & Machine Learning (Fall 2019)
Homework 2 — Privacy Attacks on Machine Learning Models

Name: Your Name Here

October 8, 2019

This is an individual assignment!

Instructions

Please read the instructions and questions carefully. Write your answers directly in the space provided. Compile the tex document and hand in the resulting PDF.

In this assignment you will implement and evaluate several membership inference attacks in Python. Use the code skeleton provided and submit the completed source file(s) alongside with the PDF.[1] *Note: bonus points you get on this homework *do* carry across assignments/homework.*

Assignment Files

The assignment archive contains the following Python source files:

- `hw2.py`. This file is the main assignment source file.
- `nets.py`. This file defines the neural network architectures and some useful related functions.
- `attacks.py`. This file contains attack code used in the assignment.

Note: You are encouraged to take a look at the provided files. This may help you successfully complete the assignment.

[1] You are encouraged to use Python3. You may use HiPerGator or your own system. This assignment can be done with or without GPUs. If you want to use a GPU, please adjust `setup_session()` in `hw2.py` accordingly.

Problem 1: Getting Familiar with Neural Networks (25 pts)

For this problem, you will train neural networks to do MNIST classification. You will get familiar with the provided code skeleton and answer some questions about what you observe when running the code.

To run the code for this problem, use the following command.

```
python3 hw2.py problem1 <nn_desc> <target_train_size> <num_epoch>
```

Here `<nn_desc>` is a neural network description string (no whitespaces). It can take two forms: `simple,<num_hidden>,<l2_reg_const>` or `deep`. The latter specifies the deep neural network architecture (see `get_deeper_classifier()` in `nets.py` for details), whereas the former specifies a simple neural network architecture (see `get_simple_classifier()` in `nets.py` for details) with one hidden layer with `<num_hidden>` neurons and an L_2 regularization constant of `<l2_reg_const>`. Also, `<target_train_size>` is the number of records in the target model's training dataset and `<num_epoch>` is the number of epoch to train for.

For example, suppose you run the following command.

```
python3 hw2.py problem1 simple,32,0.001 1000 50
```

This will train the target model on 1000 MNIST data records (i.e., images) for 50 epochs. The target model architecture is a neural network with a single hidden layer of 32 neurons which uses L_2 regularization with a constant of 0.001.[2] (The loss function is the categorical cross entropy.)

1. (5 pts) Run the code using the command provided above. What is the training accuracy? What is the test accuracy? Propose a simple metric of overfitting. How overfitted is the target model you just trained (according to your metric)?

   Your answer here.

2. (10 pts) Run the code to train the target model while varying the number of training epochs (from 10 to 1000), the size of the training data (from 100 to 2000), the number of hidden layer neurons (from 32 to 1024), and the regularization constant (from 0.0 to 0.01).

   What do you observe? For what combination of parameters is the train/test accuracy the highest? For each parameter you varied, explain its impact on accuracy and overfitting (according to your metric).

   Your answer here.

3. (10 pts) Take the best performing architecture you found above. For this question, we are interested in inputs that are misclassified by the target network. Use the provided `plot_image()` function to plot some examples of test images that are misclassified. Paste the plot below. Do you notice anything?

   Your answer here.

   Now repeat the same procedure with the deep architecture. Does it change your answer?

   Your answer here.

[2] By default, for problem 1, the code will provide detailed output about the training process and the accuracy of the target model. To change this behavior, edit line 180 of `hw2.py`.

Problem 2: Implementing the Shokri et al. Attack (30 pts)

For this problem you will implement the Shokri et al. [1] attack. Put your code in the `shokri_attack_models()` function which is located in `attacks.py`. The companion function `do_shokri_attack()` is already provided so you do not have to implement it.

Refer to the course slides and/or the paper [1] for instructions on how to implement the attack. Comments in the provided code skeleton are here to guide your implementation.

To run the code for this problem, use the following.

```
python3 hw2.py problem2 <nn_desc> <train_sz> <num_epoch> <num_shadow> <attack_model>
```

Here `<num_shadow>` is the number of shadow models to use and `<attack_model>` is a string denoting the attack model type (scikit-learn). See the code for options.

Answer the following questions.

1. (10 pts)

   Once you have implemented the attack, run the following command:

   ```
   python3 hw2.py problem2 simple,32,0.0 1000 50 10 LR
   ```

   What is the accuracy of the attack? What is the advantage?

   Your answer here.

2. (10 pts) Run the attack on the most overfitted target model (according to the parameters you found in the previous problems and your overfitting metric). Also run the attack on the least overfitted but best performing target model. Compare the attack accuracy and advantage in both cases. What do you notice? According to your experiments what are the factors that lead to attack success? (Justify your answer.)

   Your answer here.

3. (10 pts)

   Now vary the attack model type. What do you notice? What are the best/worst performing attack model types?

   Your answer here.

Problem 3: More attacks (30 pts)

For this problem you will implement three more attacks.

- Loss attack (`do_loss_attack()`): for this attack you are given the mean and standard deviation of the target model's training loss and testing loss. You can assume that the loss follows a Gaussian distribution.

  The attack works as follows: given a target record, measure its loss on the target model and then decide (IN or OUT) based on which loss distribution the sample most likely comes from.

- Loss attack2 (`do_loss_attack2()`): for this attack you are given only the mean and standard deviation of the target model's training loss. In addition, the attack function takes a threshold parameter. Again, you can assume that the loss follows a Gaussian distribution.

  The attack works as follows: given a target record, measure its loss on the target model and then decide (IN or OUT) based on whether the probability of obtaining a loss value as extreme or less extreme than that is *lower* than the threshold.

- Posterior attack (`do_posterior_attack()`): for this attack you only have the ability to query the target model. You are also given a threshold parameter.

  The attack works as follows: given a target record, query it on the target model to obtain its posterior (i.e., the predicted probability over the true class label) and then decide (IN or OUT) based on whether the posterior is *greater* than the threshold.

Implement all three attacks and answer the following questions.

1. (5 pts) Pick a set of parameters. What is the attack accuracy and advantage of the loss attack? Specify the command you ran!

   Your answer here.

2. (10 pts) Loss attack2 takes a threshold parameter. What is the optimal threshold value? Justify your answer. (Hint: you can write some code to find out experimentally.)

   Your answer here.

3. (10 pts) The posterior attack takes a threshold parameter. What is the optimal threshold value? Justify your answer. (Hint: you can write some code to find out experimentally.)

   Your answer here.

4. (5 pts) Which of the three attacks performs best?

   Your answer here.

Problem 4: Overfitting & Other factors (15 pts)

For this problem, we are interested in understanding how much of a role overfitting plays in the success of the attack (compared to other factors).

1. (10 pts) Play with the neural network architecture and other parameters to train target models with varying level of overfitting (according to your metric from Problem 1). In each case, run all four attacks and record the accuracy and advantage. Paste the results below as plot(s) or table(s). What do you observe?

   Your answer here.

   Can you find significantly different architectures / sets of parameters with a similar overfitting level but with different attack success? What do you conclude?

   Your answer here.

2. (5 pts) Now consider target images which your attacks successfully infer the membership status of. Plot some of them with `plot_image()` and paste the plot below. What do you notice?

   Your answer here.

[Bonus] Problem 5: Membership Inference on CIFAR-100 (20 pts)

Repeat problems 1 through 4 but using CIFAR-100 as a dataset. For this, you will need to use suitable neural network architectures. Explain/Justify your choices!

1. (15 pts) Repeat Problems 1, 2, 3, and 4 here.

   Your answer here.

2. (5 pts) Explain how your answers have changed. Are CIFAR-100 records more or less vulnerable to membership inference attacks (than MNIST)? Why? (Justify your answer.)

   Your answer here.

References

[1] Shokri, R., Stronati, M., Song, C., and Shmatikov, V. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)* (2017), IEEE, pp. 3–18.