NUS CS-CS5562: Trustworthy Machine Learning October 2, 2023

Assignment 3

Lecturer: Reza Shokri

Introduction

The objective of this assignment is to understand where privacy leakage comes from for a model and exploit this leakage to perform membership inference and training data reconstruction attacks. The assignment consists of the following parts:

1. **Warm-up**: Implement a naive membership inference attack to get familiar with the notation, attack setting, and code.

2. **Membership Inference Attacks using Loss Values**: Implement a membership inference attack on a given target model using loss values returned by it on its train and test datasets.

3. **Attacking the MLaaS Setting**: Implement a membership inference attack for a model in the MLaaS setting i.e. when you are not given the model's train and test datasets.

4. **Testing Unintended Memorization**: Assess the risk of data points being unintentionally memorized by models, taking classification and natural language processing tasks as examples.

5. **(Optional) Training Data Reconstruction**: Implement a reconstruction attack for a sequence model using a measure of memorization suited to natural language processing.

You will need to download some models for tasks 3 and 5. You can find them in the Google Drive folder here.*

You will need to implement the code in `privacy_assignment_1.ipynb` and write a report about the tasks. Details about the exact items to be reported are given in the corresponding task descriptions in this document.

The code in the notebook was tested using `tensorflow=2.4`. Please use the same version to avoid any errors. You can also run it on Google Colab.

- You need to use the LaTeX template we provided in the `report/report.tex`.
- **The report should ONLY contain FOUR pages. Anything that exceeds the limit will be ignored.**
- You are required to submit a zip containing the completed `privacy_assignment_1.ipynb`, your LaTeX file(s) and the compiled PDF file for the report (name it `report.pdf` and keep it in the `report` folder). Please name your zip file after you matriculation number.

*https://drive.google.com/drive/folders/1HYmGGkXpePKZgRBJJ82Q3tPLlsl2ry8q?usp=sharing

1 Warm-up

1.1 Dataset and Model Notation

We will define a classification dataset D to be a set of feature-label pairs (x_i, y_i), where $x_i \in \mathcal{X}$ and $y_i \in \{1, \ldots, k\}$, where k is the number of classes in the dataset. A classification model $h_\theta : \mathcal{X} \to \mathbb{R}^k$ is a mapping from the input space X to the output space $\mathbb{R}^k$. Here the output corresponds to the logit space, so these are real-valued numbers that can be positive or negative. θ represents all the parameters of the model i.e. convolutional filters, fully connected weights, biases, etc. These parameters are the ones that will be optimized over when a network is trained.

We optimize for the parameters θ by minimizing a loss function $l : \mathbb{R}^k \times \mathbb{Z}_+ \to \mathbb{R}_+$ which is a mapping from the model output and true label to a non-negative number. For a given input $x \in \mathcal{X}$ and its true label $y \in \mathbb{Z}$, the loss of the classifier on x is given by $l(h_\theta(x), y)$. The optimization procedure is parameterized by the order in which the data points are seen, the learning rate, number of epochs, etc. We can encapsulate these choices by representing them as the training algorithm $\mathcal{A}$.

You will now set up the dataset loading and model training code that will be used for the rest of the questions in the assignment.

Tasks

1. Complete the `Cifar10Dataset` class definition which loads and normalizes the CIFAR10 image dataset.

2. You have been provided with the architecture definition of a CNN classifier in the function `cnn_classifier`. Complete the function `get_trained_model_on_cifar10` to train the CNN classifier model h_θ on a given train dataset D_{tr}. Train the classifier *without* regularization. You can choose your own hyperparameters for the training algorithm $\mathcal{A}$. Note that `tensorflow` trains models without regularization by default.

3. Obtain the train and test accuracies of h_θ.

1.2 Membership Inference Attacks

We will now define membership inference attack setting, as given in the paper by Shokri et al. [2017]. Suppose we are given a model h_θ trained on a dataset D_{tr} and a data point (x, y). The objective of a membership inference attacker M is to determine the membership label of x i.e. whether x was part of the train dataset D_{tr} (i.e. "member") or not (i.e. "non-member"). Data points in the test dataset D_{ts} are considered non-members. In this assignment, we shall see how to use the loss values returned by h_θ for a membership inference attack.

1.3 The Naive Membership Inference Attack

You will now implement a naive membership inference attack M_{naive} that exploits the difference between the train and test accuracies of the model h_θ. M_{naive} uses the following rules to determine if a data point x is a member or a non-member:

1. If h_θ predicts the label of x correctly i.e. $y_{pred} = y$, then M_{naive} outputs "member".

2. If h_θ predicts the label of x incorrectly i.e $y_{pred} \neq y$, then M_{naive} outputs "non-member".

Given the rules above, can you already predict the accuracy of M_{naive} from the values of the train and test dataset accuracies?

Tasks

1. Complete the function `naive_membership_inference_attack` that performs the naive membership inference attack. You will be sending the datasets D_{tr}, D_{ts} and the model h_θ as input to the function. The function will use the rules

specified above to generate and return the membership labels ("member" or "non-member") for data points in D_{tr} and D_{ts}.

2. Since we know the true membership labels for data points in D_{tr} and D_{ts}, we can compute the True Positive (TP), False Positive (FP), True Negative (TN) and False Negative (FN) quantities for M_{naive}. Compute and report them in `report.pdf`. You can refer to this tutorial for an explanation of these quantities.

3. The accuracy of the attack can be computed as follows: $\frac{TP+TN}{TP+FP+TN+FN}$. Compute and report the attack accuracy of M_{naive} in `report.pdf`.

2 Membership Inference Attacks Using Loss Values

In this section, you will be implementing membership inference attacks M_{loss} and $M_{loss,per\ class}$ that use loss values returned by the model h_θ to generate membership labels. The details of both these attacks are given in the respective subsections below.

2.1 M_{loss} - Using Loss Values to Estimate a Single Threshold

M_{loss} observes the loss values on the entire train dataset D_{tr} and the test dataset D_{ts} and computes a threshold T_{loss} to determine whether a particular loss value was obtained on a member or a non-member. Given a data point x, M_{loss} generates its membership label using a precomputed threshold T_{loss} as follows:

1. Compute $loss = L(\theta(x), y)$ for the data point x.
2. If $loss < T_{loss}$, then M_{loss} outputs "member".
3. If $loss \geq T_{loss}$, then M_{loss} outputs "non-member".

Tasks

1. Complete the function `loss_membership_inference_attack` that takes as input a model h_θ, the loss function l used while training h_θ, the train and test datasets D_{tr} and D_{ts}, and returns the loss values of h_θ on all data points in D_{tr} and D_{ts}.
2. Plot a histogram using train and test dataset loss values. You should be able to distinguish between the training and test loss values in the plot.
3. Choose a threshold T_{loss} and compute the corresponding TP, FP, TN and FN quantities. Report the attack accuracy in `report.pdf`.

4. Observe how TP, FP, TN and FN change with a change in the threshold T_{loss}. Find a T_{loss} that maximises the attack accuracy. Explain how you calculated this threshold in `report.pdf`.

2.2 $M_{loss,per\ class}$ - Using Loss Values to Estimate a Threshold for Each Class

We will now improve the membership inference attack M_{loss} implemented in the previous section by utilising the fact that the model h_θ performs differently on different classes in the dataset. We will denote this attacker as $M_{loss,per\ class}$.

$M_{loss,per\ class}$ uses the same loss values as before but additionally computes a separate threshold $T_{loss,class\ i}$ for each class i of the data to determine whether a particular loss value was obtained on a member or a non-member. Given a data point x, $M_{loss,classwise}$ generates its membership label using the per-class threshold $T_{loss,class\ i}$ as follows:

1. Compute $loss = l(h_\theta(x), y)$ for the data point x.
2. Compute the class label i returned by h_θ for the data point x. This will fix the threshold $T_{loss,class\ i}$ to be used.
3. If $loss < T_{loss,class\ i}$, then $M_{loss,per\ class}$ outputs "member".
4. If $loss \geq T_{loss,class\ i}$, then $M_{loss,per\ class}$ outputs "non-member".

Tasks

1. Complete the function `per_class_loss_membership_inference_attack` that takes as input a model h_θ, the loss function L used while training h_θ, the train and test datasets D_{tr} and D_{ts}, and returns the loss values of h_θ on all data points in D_{tr} and D_{ts} separately for *each class*.

2. Plot a histogram for each class using the corresponding train and test dataset loss values. Plot the histograms of the 2 classes that you think show the clearest distinction between the loss values. Note that your answer might vary according to your chosen hyperparameters for the model's training algorithm $\mathcal{A}$.

3. Choose a threshold $T_{loss,class\ i}$ and compute the corresponding TP, FP, TN and FN quantities for each class. Report the attack accuracy for each class, and the overall attack accuracy in `report.pdf`.

4. Observe how TP, FP, TN and FN change with a change in the threshold $T_{loss,class\ i}$. Find the thresholds $T_{loss,class\ i}$ that maximize the overall attack accuracy. Explain how you found these thresholds in `report.pdf`. How does this attack accuracy compare with the accuracy of M_{loss} in the previous section?

2.3 Attacking a Model Trained With Regularization

Regularization is added to the training objective of a model to prevent it from overfitting on its train dataset. How would you expect the attack accuracy to change on such a regularized model, and why? Think about what membership inference attacks exploit! Write your reasons in `report.pdf`. Complete the function `regularized_cnn_classifier` and train a model with regularization $h_{\theta,reg}$. Perform the attacks M_{loss} and $M_{loss,per\ class}$ on $h_{\theta,reg}$ and report the attack accuracies in `report.pdf`.

3 Attacking the MLaaS Setting

Companies like Google, Amazon, and Microsoft now offer machine learning as a service (MLaaS) solutions on their cloud platforms. These services provide APIs for users to upload their data and then train and query models. However, the exact details of the model architectures and the training algorithms are hidden from the users.

Users performing inference using the trained model only have access to the model outputs, and do not have access to the model's train and test datasets. This means that an adversary who wants to attack this model does not have the ground truth membership labels to estimate the loss thresholds. Does this mean the adversary cannot perform a membership inference attack successfully? You shall see that this isn't the case!

3.1 Designing an Attacker for the MLaaS Setting

You will now design and perform your own membership inference attack $M_{challenge}$ on a given target model $h_{\theta,challenge}$. This model has been trained on a subset $D_{tr,challenge} \subset D$ of the CIFAR10 dataset. $D_{tr,challenge}$ will not be provided to you. Instead, you will get the input features and labels of another subset $D_{other,challenge} \subset D$ of the CIFAR10 dataset to use for your attack.

You are required to complete the function `challenge_membership_inference_attack`. This function will generate the membership labels for points from another subset of the CIFAR10 dataset $D_{eval,challenge} \subset D$. You are required to explain your attack design in `report.pdf`. We will grade you based on how many membership labels your attack $M_{challenge}$ predicts correctly.

The output logits on the evaluation dataset $D_{eval,challenge}$ are in `attacking_mlaas_data.npz` (downloaded from the Google Drive folder mentioned above). The code for loading this data is already provided to you.

Hints

1. Membership inference attacks use the fact that a model performs differently on its train and test datasets. Since you do not have the original train and test datasets, how will you mimic the behaviour of the target model $h_{\theta,challenge}$ on the two datasets?

2. In the previous sections, we saw that loss values leak membership information about data points. What other information from a model can be used for an attack?

4 Testing Unintended Memorization

We will now assess the risk of data points in a model's train dataset being unintentionally memorized by the model during the training process. This unintended memorization is of significant concern especially when the model is public but it has been trained on sensitive or private data. We will look at testing methodologies for an image classification task as well as a natural language processing task.

4.1 Assessing a CNN Classifier on CIFAR10

We will first check how our CNN classifier model h_θ is memorizing points from its CIFAR10 train dataset D_{tr}. Suppose we want to check if a particular data point $(x, y) \in D_{tr}$ is being memorized. For a classification model like our CNN classifier, we can define memorization according to Feldman [2020]:

$$mem(\mathcal{A}, D_{tr}, x) = \mathbf{Pr}_{h_\theta \sim \mathcal{A}(D_{tr})}[h_\theta(x) = y] - \mathbf{Pr}_{h'_\theta \sim \mathcal{A}(D_{tr} \setminus x)}[h'_\theta(x) = y] \quad (1)$$

where $\mathcal{A}$ is the training algorithm, D_{tr} is the train dataset, h_θ is the model trained on D_{tr}, h'_θ is the model trained on D_{tr} without the data point x, and y is the true label corresponding to x.

Memorization, as defined above, is measuring how much the model output differs when x is included in its train dataset and when it is not. The higher this quantity for x, the more the model has memorized it.

Tasks

Complete the function `check_memorization_for_point`, which needs to do the following tasks:

1. Choose a data point x to assess. Train two models h_θ, h'_θ on D_{tr} with and without this data point.

2. Compute $mem(\mathcal{A}, D_{tr}, x)$ for the point according to equation 1. To what extent has the model memorized x?

3. Additionally, compute the predicted labels and the loss values returned by the two models on x.

4. Are the predicted labels and loss values returned by the two models different? Repeat this experiment for different data points x and comment on the extent of the model's memorization in `report.pdf`.

4.2 Assessing an LSTM Next Character Predictor on Shakespeare Dataset

Now we will assess the risk of data points to be memorized by generative sequence models (used for natural language processing tasks), as outlined in the paper by Carlini et al. [2019]. Our task will be to train a model g_θ for the next character prediction task. g_θ will be trained to take in a sequence of characters $x_{seq} = x_1, \ldots, x_n$ as input and output a probability distribution $\mathbf{Pr}(x_{n+1}|x_1, \ldots, x_n)$ over what it thinks will be the next character in the sequence.

Threat Model: Carlini et al. [2019] assume "a threat model of curious or malevolent users that can query models a large number of times, adaptively, but only in a black-box fashion where they see only the models' output probabilities (or logits)".

For this threat model, we can check if g_θ is likely to memorize and potentially expose unique, sensitive sequences in train datasets using the testing methodology described below.

Testing Methodology: We will insert randomly chosen canary sequences n number of times. To see how much g_θ tends to memorize, we will measure the relative difference in *log-perplexity* (or 'perplexity' for short) between the inserted canaries and non-inserted random sequences. Perplexity measures the likelihood of data sequences i.e. how 'surprised' g_θ is to see a given sequence. We can interpret

the measure as the number of bits required to represent some sequence x_{seq} under a probability distribution defined by the generative sequence model g_θ. It is defined as follows:

$$P_{g_\theta}(x_{seq} = x_1, \ldots, x_n) = -\log_2 \mathbf{Pr}(x_{seq}|g_\theta) \tag{2}$$

$$= \sum_{i=2}^{n} \Big(-\log_2 \mathbf{Pr}(x_i|g_\theta(x_1, \ldots, x_{i-1})) \Big) \tag{3}$$

A higher perplexity implies that g_θ is more surprised to see x_{seq}, and a lower perplexity implies that x_{seq} is more likely to be a normal sequence. We could measure g_θ's memorization of its train dataset by directly computing the perplexity of sequences in it. However, like we have seen in equation 1, memorization depends on the specific model, dataset, and the training algorithm. A better approach would be to measure the *relative* difference in perplexity of some data between models g_θ (trained on that data) and g'_θ (trained without that data) to take these dependencies into account. You will now implement this testing methodology for the next character prediction task, whose details are provided below.

Dataset: The dataset contains text from works of Shakespeare. The code for loading the dataset has been provided to you in the class definition `ShakespeareDataset`. You can read more about the dataset here.

Architecture of g_θ (and g'_θ): One layer LSTM with 400 hidden units. The code for the model has been provided to you in the class definition `LstmCharGenerator`.

NLP Task: Next character prediction i.e. g_θ (and g'_θ) receives a sequence of characters as input and outputs a probability distribution over what it thinks will be the next character in the sequence.

Format of Canary: A canary c will be of the following format "The random sequence is ◯ ◯ ◯◯", where each placeholder ◯ is filled with a randomly chosen uppercase letter in $[A, J]$.

Tasks

1. Complete the function `perplexity` to compute the perplexity of a sequence according to equation 2.

2. Create a canary c according to the format.

3. Complete the function `get_augmented_dataset_with_canaries` in the class `ShakespeareDataset` to augment the original D_{tr} with k copies of c to form D'_{tr}. Train another LSTM g'_θ with the same hyperparameters as before on D'_{tr}, until g'_θ achieves minimum validation loss.

4. Report the relative difference between the perplexities returned by g_θ and g'_θ on the canary c.

5. Repeat this task with different canaries and different values for k. Report these results and comment on the relationship between k and the extent of memorization in `report.pdf`.

5 Training Data Reconstruction (Optional)

We will now try to reconstruct sequences (canaries) in the train dataset for a given generative sequence model using the perplexity term defined in equation 2. A brute-force approach to reconstructing canaries would be list all possible canaries, compute their perplexity, and return them in the order of decreasing perplexity. However, if the number of possible canaries is large, this approach will not scale well. This approach has been implemented in the function `brute_force_reconstruction` so you can experiment with different lengths of some example canaries.

We will provide you with the trained model $g_{\theta,challenge}$. Here $g_{\theta,challenge}$ has first been trained on $D_{tr,challenge}$. $D_{tr,challenge}$ is then augmented with 5 canaries, each of which will be added a varying k number of times. $g_{\theta,challenge}$ will then be trained for a few more epochs on the augmented dataset.

Using the trained model $g_{\theta,challenge}$, you are required to reconstruct the canaries that were added to $D_{tr,challenge}$ by modifying the function `efficient_reconstruction`. Explain your reconstruction algorithm and report the 5 most likely canaries according to this algorithm in `report.pdf`.

The details of the dataset, model, and canary formats are given below.

Dataset: The dataset contains text from works of Shakespeare. The code for loading the dataset has been provided to you in the class definition `ShakespeareDataset`. You can read more about the dataset here.

Architecture of $g_{\theta,challenge}$: One layer LSTM with 400 hidden units. The code for the model has been provided to you in the class definition `LstmCharGenerator`.

NLP Task: Next character prediction i.e. $g_{\theta,challenge}$ receives a sequence of characters as input and outputs a probability distribution over what it thinks will be the next character in the sequence.

Format of Canary: A canary c will be a sentence containing a mix of characters from the Shakespeare dataset vocabulary. The minimum length of an inserted canary is 29 characters, whereas the maximum length is 115 characters. The prefixes of the 5 canaries are: `['Reza', 'Some machine learning', 'Some models', 'To be or', 'Didst Romeo']`.

References

Nicholas Carlini, Chang Liu, Úlfar Erlingsson, Jernej Kos, and Dawn Song. The secret sharer: Evaluating and testing unintended memorization in neural networks. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pages 267–284, 2019.

Vitaly Feldman. Does learning require memorization? a short tale about a long tail. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 954–959, 2020.

Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 3–18. IEEE, 2017.