CY 7790, Model Extraction Attacks and Memorization in Machine Learning

John Abascal and Harsh Chaudhari

November 15, 2021

1 Stealing Machine Learning Models via Prediction APIs

Problem Statement. Machine learning models are sometimes deemed confidential due to their sensitive training data, commercial value, or security applications. Because machine-learning-as-a-service is becoming an increasingly popular revenue source for companies such as Amazon, Google, and Microsoft, the potentially sensitive training data is at risk. Even though many of these services are typically offered via a publicly accessible query interface (black-box), an adversary can potentially extract information about the model parameters or training data. In this paper, the authors' goal is to create a model extraction attack (i.e. the adversary's goal is to generate a model that is nearly identical to the private target model). The authors consider private models when constructing their attacks because many models because:

1. Model is proprietary or monetized per query
2. Training data is sensitive and white-box access to model leaks info about training data.
3. Model is used for security purposes where attacker knowledge would improve evasion attacks (e.g. spam detection)

Threat Model. The goal of the adversary is to create a machine learning model that "closely matches" the private target model. Because every machine learning service is different, the capabilities of the adversary can vary depending on which service it is targeting. The attack can be considered "black-box" since the main attribute of the service that the adversary requires is query access. The knowledge of the adversary can potentially include the model class, training algorithm, hyper-parameters, etc.

Definitions The authors measure similarity between the two models using the following metrics:

- **Test Error:** The average test error over a set ,D, is given by $R_{\text{test}}(f, \hat{f}) = \sum_{(\mathbf{x},y)\in D} d(f(\mathbf{x}), \hat{f}(\mathbf{x}))/|D|$
  A low test error implies that $\hat{f}$ matches f well for inputs distributed like the training data samples
- **Uniform Error:** For a set of vectors uniformly chosen in $\mathcal{X}$, let $R_{\text{unif}}(f, \hat{f}) = \sum_{\mathbf{x}\in U} d(f(\mathbf{x}), \hat{f}(\mathbf{x}))/|U|$. Thus, R_{unif} estimates the fraction of the full feature space on which f and $\hat{f}$ disagree.

Methodology. The authors consider several scenarios where model extraction attacks can be performed.

- **Equation Solving Attacks**
  Binary logistic regression: Develop a system of equations to solve for parameters of a model. This method has a closed form solution

  Multiclass Logistic Regression and Neural Networks: This method seeks to minimize the loss with respect to the parameters without using a closed form solution. The population for this training algorithm is attained using the oracle and the loss function depends on the model class.

– **Path Finding Attacks**
Decision trees do not compute probability; instead, they partition the space. So, the authors attempt to identify the path the input traversed.

Basic Algorithm Intuition

1. Choose random input **x** and get leaf ID of the tree output
2. Find the set features (continuous or categorical) such that **x'** in the feature set results in the same leaf ID
3. For each leaf ID, find this associated feature set

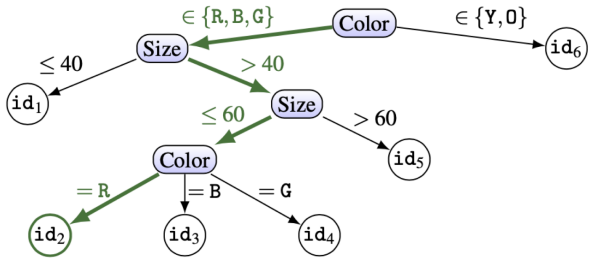

Figure 3: **Decision tree over features Color and Size.** Shows the path (thick green) to leaf $\texttt{id}_2$ on input $\mathbf{x} = \{Size = 50, Color = R\}$.

– **Class Label Only Attacks**
Lowd Meek Attack

- Binary linear classifiers: Find points near decisions boundary, solve for parameters
- Kernel methods with computable and invertible projection functions: Solve for parameters in linear space

Re-training using query responses as labels

- Uniform queries: sample m points at random
- Line search: sample m adaptive queries close to decision boundary
- Adaptive retraining: query m/r points at random, in batches of m/r, retrain on low-confidence points

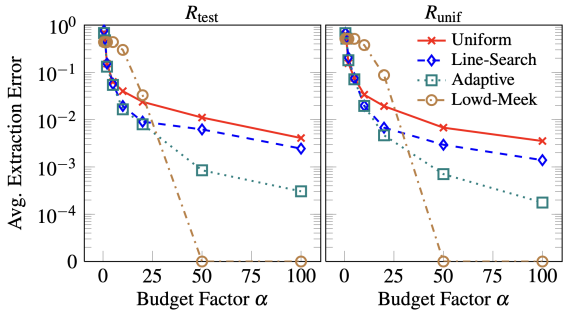

Figure 4: **Average error of extracted linear models.** Results are for different extraction strategies applied to models trained on all binary data sets from Table 3. The left shows R_{test} and the right shows R_{unif}.

The authors also discuss the following extraction countermeasures:

– **Rounding Confidences**
Eliminates guarantees in equation solving and increases leaf collision in path finding

– **Differential Privacy**
Add noise to model weights when query – prevent distinguishing between close model parameters

– **Ensemble Methods**
Not studied, but may prevent extraction attacks

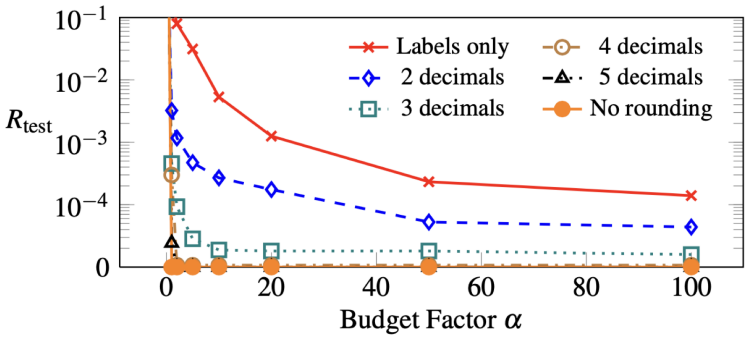

Figure 7: **Effect of rounding on model extraction.** Shows the average test error of equation-solving attacks on softmax models trained on the benchmark suite (Table 3), as we vary the number of significant digits in reported class probabilities. Extraction with no rounding and with class labels only (adaptive retraining) are added for comparison.

Class Discussion.

– **Have hosted models changed?**
Yes, all companies offer more complex models nowadays

– **Why does this work for logistic regression?**
For logistic regression, the output is exactly the sigmoid function

– **Are architecture and confidence known for multiclass cases?**
Confidence is, but architecture may not be.

– **Why do we need to extract/mimic model if we already have access to it?**

– **Does top down method for decision trees require exponential number of queries?**
No, because binary search is being done

2 Extracting Training Data from Large Language Models.

Problem Statement. The paper shows that an adversary can perform a training data extraction attack to recover individual training examples by querying a large language model (with billion parameters) that was trained on a private

dataset. The authors demonstrate their attack on GPT-2 language model which was trained on scrapes of the public Internet, and extract hundreds of verbatim text sequences from the model's training data. The authors then study the effect of model size, and string frequency on memorization and consequently propose various mitigation strategies against data memorization and their respective trade-offs.

Threat Model. The goal of the adversary is to indiscriminately extract memorized training from the language model as opposed to extracting targeted pieces of training data. The adversary has black-box input-output access to a language model, such that the adversary can query the model to get the probability distribution of next word in sequence by supplying prior tokens in sequence.

Definitions.

- **Model Knowledge Extraction:** A string s is extractable from a language model f_θ if there exists a prefix c such that:

$$s \longleftarrow \underset{s':|s'|=N}{\text{argmax}} f_\theta(fs'|c)$$

  where $f_\theta(fs'|c)$ denotes the likelihood of all sequences of length N. Since computing the most likely sequence is intractable for large N, the argmax is replaced with an appropriate sampling strategy.

- **k-Eidetic Memorization:** A string s is k-eidetic memorized (for $k \geq 1$) by f_θ if s is extractable from f_θ and s appears in at most k examples in the training data $X : |\{x \in X : s \subset x\}| \leq k$.

Methodology. The attack is a two-step process:

- **Step 1:** Repeatedly sample tokens in an autoregressive fashion from the model to generate sequences that the model considers highly likely.
- **Step 2:** Use membership inference attack which distinguishes if a sample was part of the training set or not. The attack is conducted by using the knowledge of the likelihood of the sample according to the attacked model. We discuss the sampling schemes and attack variations conducted by the authors next.

The authors propose three different sampling schemes which can be used in the aforementioned *Step 1*:

- Use top-n sampling strategy with start-of-sequence token as input to the language model.
- Divide the logit values of the language model by a decaying temperature 't' before applying softmax. Allows model to explore diverse set of prefixes, before pursuing a high confidence path.
- Seed model with random prefixes from a disjoint set scraped from internet, then proceed with top-n sampling

The authors also propose five different variants of Membership Inference attacks used in *Step 2*:

- Predict top-n lowest perplexity samples as memorized based on the assumption that the language model will assign a high likelihood to sequences seen in its training data.
- Predict samples with highest ratio of log-perplexities on smaller language models compared to the attack model as memorized.
- Compare sample log-perplexity on attack model to number of bits of entropy when sample is compressed with zlib compression.
- Compare to log-perplexity of lowercase version of sample on attack model itself.
- Same as first attack but instead of lowest perplexity, use minimum perplexity across any sliding window of 50 tokens.

Evaluation. We first discuss the methodology of how the datasets were generated by the authors:

- The authors build three datasets of $200,000$ generated samples (each of which is 256 tokens long) using one of the three mentioned strategies.
- The authors then order each of the three datasets according to each of the six membership inference metrics and select top 100 (deduplicated) samples according to each metric.
- A total of 1800 such samples were then manually determined whether the sample contains memorized text and then using limited query access to GPT-2 training data which internally used fuzzy trigram match with training data samples, a subset of them were confirmed to be memorized.

Figures 1 and 2 provide the number of unique samples memorized by GPT-2 and the manual categorizations of these strings respectively. Figure 3 gives nine examples of $k = 1$ eidetic memorized content, each of which is a random sequences between 10 and 87 characters long. Figure 4 shows how different model sizes of GPT-2 language model affect memorization. The authors provide more detailed analysis in their paper for the case of longer token sequences. The authors then discuss possible mitigation strategies against memorization and their tradeoffs:

- Training with Differential Privacy: The training procedure is slow with trade-off in test accuracy.
- Curating Training Data: Limit the amount of sensitive content that is present by filtering personal information or content with restrictive terms of use.
- Limiting Impact of Memorization on Downstream Applications: attempt to filter out generated text that contains memorized content while fine tuning Downstream Applications, if such content can be reliably detected.
- Auditing ML Models for Memorization

Class Discussion.

- Strange/uncommon words appear because of the vocabulary being very large.
- Ways to prevent loss of contextual privacy when fine-tuning for downstream tasks.
- Targeted prefix selection strategies to extract sensitive data faster and more accurately.