Authors:

1. **Kai Ji Gong, kgong01@uoguelph.ca, Student ID: 1014613**
2. **Jonathan Yu, jyu29@uoguelph.ca, Student ID: 1374640**

Project Title: Implementing Attacks on AI to Compromise Personal Information

Introduction

The objective of this project is to explore privacy attacks on AI systems. AI has become deeply embedded across nearly every industry—healthcare, social media, business, academia, technology, and more—most commonly through applications such as chatbots and intelligent platforms. Modern machine-learning models are trained on massive amounts of text, images, and other personal data, which raises growing concerns about the potential for unintended information leakage. When AI systems reveal more sensitive information than expected, whether accidentally or through targeted attacks, the privacy-related consequences can be significant. These include financial and reputational harm to individuals, violations of privacy laws and regulations, erosion of public trust, and legal liability for organizations.

In this project, we will demonstrate implementations of various privacy-related attacks on AI. We will be demonstrating membership inference and model extraction attacks on a publicly available dataset, then demonstrate them again but with privacy-related defensive measures implemented. Through our project, we aim to show the privacy vulnerabilities of AI systems, the effectiveness and necessity of implementing effective safeguards, and to provide recommendations for more responsible machine learning model development that considers privacy at every stage of design.

We chose to use a NHANES (National Health and Nutrition Examination Survey) dataset that describes whether an individual has heart disease. It does this through a binary classification variable called “target_heart” (0 for no, 1 for yes). This dataset was provided by Lisa Zhang, an assistant professor of Math and Computational Science at the University of Toronto Mississauga. It can be downloaded at this link. We chose this dataset because of the wealth of personal information stored in it such as age, race/ethnicity, BMI, family income as well as many highly-sensitive health-related features such as blood pressure, blood cholesterol, and calories. Because of the presence of both public and sensitive personal data features, we chose this dataset

for our demonstration. All implementations for this project are written in Python. We will play the role of an adversary attempting to infer and reconstruct information about individuals from the model.

Membership Inference (REMOVE SUB-HEADERS AFTER!!!)

Attack Introduction

Membership inference is an attack where, given an ML model and a sample that's already known to the attacker, the attacker is aiming to determine if the sample was a member of the training set used to build the ML model. Membership inference demonstrates the vulnerability that many ML models behave differently on training samples than on unseen data, typically by showing higher confidence and lower error on samples they already know. The difference in confidence is a huge privacy risk because it can expose whether or not an individual was part of a sensitive group, such as if they have heart disease or not in our case. Confirmation of membership can lead to the attacker inferring private facts about a person's life which can lead to discrimination, reputational damage, financial harm such as insurance denial, or even targeted scams. Addressing membership inference is an incredibly important privacy requirement when building AI systems trained on personal data.

Attack Implementation Methodology

The following steps outline how we implemented the attack script:

1. We loaded the dataset into our script while dropping the target_heart column and setting it aside as our target feature.
2. We then split the data. While there are 8000 entries in the original dataset, we capped it to 2000 to encourage overfitting. Overfitting is defined as when the model learns the training data very well, almost to the point of memorization, instead of learning general patterns. It results in the model performing poorly on unseen data. Overfitting is ideal for our membership inference demonstration because the confidence gap between

training and unseen data is more obvious, making it easier to detect if a particular record was used during training.

3. We then trained and fitted a simple MLP. We chose MLP as our model because it can overfit when given small datasets. We used a small model trained with weak regularization to preserve some memorization without reaching the extreme case of perfectly memorizing all training data with 100% accuracy.
4. We then evaluated the model's performance. Now we're ready to demonstrate membership inference.
5. We defined a helper function for converting a binary label into a one-hot vector. This is used to calculate log loss later.
6. We then needed samples to test on. We decided to pick a random training example from the top 60% of the most confident predictions and a random testing example from the bottom 40% of the least confident predictions. This is so every time we run the script, we can more consistently demonstrate membership inference.
7. We then printed various statistics for both the training and testing samples.
    a. True label: The actual target_heart value for the sample (0 for no heart disease, 1 for yes).
    b. Predicted label: The target_heart value the model predicted.
    c. Class probabilities: The probabilities for each class (0 and 1) the model predicted.
    d. Max confidence: The max confidence of the predictions.
    e. Log loss: The error/log-loss. A small log loss means the model was very confident and correct. A large log loss means the model was unsure or wrong.

Attack Results and Privacy Analysis

```
----Model Performance Summary----
Train accuracy: 0.911
Test  accuracy: 0.819

----Membership Inference Demo----
Training Example (Member):
  True label:          0.0
  Predicted label:     0
  Class probabilities: [99.983562%, 0.016438%]
  Max confidence:      0.9998
  Log loss:            0.0002

Test Example (Non-Member):
  True label:          1.0
  Predicted label:     0
  Class probabilities: [62.830428%, 37.169572%]
  Max confidence:      0.6283
  Log loss:            0.9897
```

Fig. 1. Output of attack script.

We ran the script and got the results as shown in Fig. 1. We can see the non-private model achieved 91.1% training accuracy and 81.9% test accuracy. This shows that noticeable overfitting did occur but the model can still generalize reasonably well.

We can see in the training example, the sample belonged to the negative class, and the model predicted it correctly with an extremely high confidence of 99.98%, and an extremely low log loss of 0.0002. This shows the model was extremely confident in its prediction, which suggests memorization of this sample. In contrast, we can see the test example was misclassified with a much lower confidence of 62.83% and a higher log loss of 0.9897. This shows the model's error.

The large confidence gap of 37.15% between the training and testing sets provides a clear and exploitable signal to the attacker. The model's certainty on the training sample versus its uncertainty on the test sample makes for an obvious inference for the attacker that the first sample was part of the training dataset. In Fig. 1, the model's output reveals the individual does not have heart disease. However, in another case, the same level of confidence can apply to a sample on an individual with heart disease.

The privacy risk here is that differences in confidence and log loss values can unintentionally leak information about individuals even when raw training data is never exposed.

As discussed earlier, simply confirming if an individual was part of a sensitive dataset or not can lead to many real-world consequences such as reputational or financial.

Defense Implementation Methodology

The following steps outline how we implemented the defense script:

1. We loaded the dataset into our script while dropping the target_heart column and setting it aside as our target feature.
2. We then split the data. We capped the training data to 2000 just like in the non-private model to mimic the original attack as closely as possible.
3. We chose to use Tensorflow Privacy and its DP libraries as our defensive implementation. So we scaled the features of the dataset then defined a small Keras MLP that is approximate to our original scikit-learn MLP.
4. We then used the DP-SGD optimizer defined in Tensorflow Privacy. This optimizer is an extension of the standard SGD algorithm that protects individual training examples by limiting their impact on the model and adding noise during training. Our loss function is sparse_categorical_crossentropy because it's required by DP-SGD.
5. We chose random samples the same way as in the non-private model and outputted the same statistics for each one.

Defense Results and Privacy Analysis

```
----DP Model Performance Summary----
Train accuracy: 0.670
Test  accuracy: 0.683
63/63 [==============================] - 0s 444us/step
188/188 [==============================] - 0s 471us/step

----Membership Inference with Differntial Privacy Demo----
Training Example (Member):
  True label:          0.0
  Predicted label:     0
  Class probabilities: [71.344721%, 28.655279%]
  Max confidence:      0.7134
  Log loss:            0.3376

Test Example (Non-Member):
  True label:          0.0
  Predicted label:     1
  Class probabilities: [49.781621%, 50.218374%]
  Max confidence:      0.5022
  Log loss:            0.6975
```

Fig. 2. Output of attack but with differential privacy (DP) applied.

We ran the script and got the results as shown in Fig. 2. We can see the model with DP applied achieved 67% training accuracy and 68.3% test accuracy. This shows the model no longer overfits the training data as the non-private model did. This result is expected because DP-SGD adds noise and gradient clipping to reduce the model's ability to memorize individual training samples. The results shown in Fig. 2. show that differential privacy has successfully been applied and limited overfitting, which is a key vulnerability to allow membership inference attacks.

We can see in the training example, the sample belonged to the negative class, and the model predicted it correctly with a moderate confidence of 71.34% and log loss of 0.3376. This result is good in our demonstration of membership inference defense because now the model is not excessively certain even on data it's seen during training.

We can see in the test example, the sample belonged to the positive class, and the model predicted it incorrectly with a lower confidence of 50.22% (essentially guessing between 0 or 1) and log loss of 0.6975. This is a significant privacy improvement because the attacker can much

less likely infer if the given sample was part of the training data. We can see that both predictions fall within a range where the model does not strongly favor one outcome.

These results demonstrate the privacy benefits of differential privacy. Since we added noise and gradient clipping to limit the impact of individual samples, the attacker is much less confident of whether an individual was part of a training set based on the outputs leaked. As a result, membership inference becomes much more difficult. The downside is that the model's accuracy decreases which also affects usability. This trade-off is for stronger privacy protections.

Model Extraction

Attack Introduction

According to NIST's *Adversarial Machine Learning* paper (Vassilev et al, 2024), model extraction is one of four privacy attacks against predictive artificial intelligence. The goal of model extraction is to extract information about model architecture and parameters (such as weights and biases) using queries to the model.

To demonstrate a model extraction attack, we follow the method described in section 4.1 of Carlini et al (2020) for extracting weights from simple models. We assume the attacker has knowledge about the model architecture and has access to the model output probabilities. The dataset we chose for this attack is the sklearn breast cancer dataset. This dataset is based on the breast cancer Wisconsin dataset (Wolberg et al, 1993), which contains 30 features describing characteristics of cell nuclei in real medical photos. The target of this dataset is a positive or negative diagnosis. The dataset has been de-identified, but the real medical data in the dataset can still be considered personal information.

Attack Implementation Methodology

1. Load data. We use the breast_cancer dataset from sklearn.datasets as it is a binary classification dataset and contains real health data. We split the data into train and test sets using the train_test_split function from sklearn.model_selection. We chose a test size of 20% using a random state of 42.

2. Train model. We use a simple LogisticRegression model from sklearn.linear_model. We kept the default parameters for simplicity. We changed max_iters to 10000 to ensure the model converges.
3. Evaluate performance. This simple model achieves 96% accuracy on the test data.
4. Extract weights. We use the method described in section 4.1 of Carlini et al (2020). Logistic Regression is linear with a single layer, so the weight matrix W is of shape (D, 1) where D is the dimensionality of the data. In this case D = 30. The model can be written as f(x) = W·x + b. For each "row" W_i in W, we use the corresponding standard basis vector e_i. Then by using finite differences to estimate the directional derivative:

$$\begin{aligned} f(x + e_i) - f(x) &= W \cdot (x + e_i) + b - (W \cdot x + b) \\ &= W \cdot x + W \cdot e_i + b - W \cdot x - b \\ &= W \cdot e_i \text{ (or the i-th row of the weight matrix W)} \end{aligned}$$

   where f(x) is the logit output of model f on input x. We chose an arbitrary point x in the test set.
5. Extract biases. For the Logistic Regression model, the bias is a single float value. The model can be written as f(x) = W·x + b, so this implies that b = f(x) - W·x. So we can find the weights by subtracting the dot product of the weights we found and x with the logit output of our model f on input x.
6. Construct a model using the extracted weights and biases. This is done by simply using the extracted weights and biases in a linear function. We then apply a sigmoid activation to the output to obtain a probability in [0,1], then we apply a threshold value of 0.5 to obtain a binary prediction as the output.
7. Compare performance. We use the test set. The extracted model performs identically.

Attack Results and Privacy Analysis

As stated in *NIST Adversarial Machine Learning* (Vassilev et al, 2024), the main privacy risk of model extraction is that it often enables other stronger white-box attacks. For example, it would make membership inference and data reconstruction much easier. For this reason Vassilev

et al (2024) states that mitigating the risk of model extraction can also reduce the risk of other "downstream" attacks as well.

Having access to the weights and biases of a model can give the attacker information about the data, for example which features are the most weighted. This can also lead to further attacks and help the attacker create their own model for faster offline data reconstruction attacks.

There are also compliance risks, for example, weights trained using personal data may be considered sensitive data and could require action and constitute a data breach if leaked.

Although the out demonstrated attack model was simple, this model extraction attack using finite differences to estimate weights can still work on multi-layer models. As shown by Carlini et al (2020), the attack on multi-layer models would involve extracting weights layer by layer. For each extracted layer, the weights can be used to reverse the layer, effectively peeling a layer from the model, so the next layer can be extracted.

Defense Implementation Methodology

1. To demonstrate DP, we add Gaussian noise to the logit output of the model. The noise added is Normal(0, 1).
2. Evaluate. The model with noisy output performs almost identically to the original output. After multiple attempts, we found that the differentially private model's accuracy fluctuates by ±0.01 on the test set.
3. Extract weights & biases. Using the same method shown above, weights and bias were extracted from the DP model.
4. Evaluate & compare. The extracted weights from the DP model perform significantly worse. The weights and bias are often skewed, so the model often predicts only 1 or 0. This is shown by the classification matrix. The model accuracy is around 50%.
5. To demonstrate limiting model output, a copy of the model is made. The predict_proba() function of the model is then overwritten, to render the extraction method described above impossible

Defense Results and Privacy Analysis

Using differential privacy by adding noise to our model output proved to be very effective. The resulting model performed almost identically to the non-DP version, so not much utility was lost. However, the privacy gain is significant. Extracting weights from the DP model using the finite difference method results in a model with significantly skewed weights and bias. In our testing, the extracted model would often only predict 1 or 0, achieving an accuracy around 62% on the test set. Compared to the 96% accuracy of the extracted model from the non-DP version, this is a significant privacy improvement.

DP Model

	precision	recall	f1-score	support
0	0.97	0.88	0.93	43
1	0.93	0.99	0.96	71
accuracy			0.95	114
macro avg	0.95	0.93	0.94	114
weighted avg	0.95	0.95	0.95	114

DP Extracted

	precision	recall	f1-score	support
0	0.00	0.00	0.00	43
1	0.62	1.00	0.77	71
accuracy			0.62	114
macro avg	0.31	0.50	0.38	114
weighted avg	0.39	0.62	0.48	114

Figure. Accuracy of DP model vs extracted DP model

By limiting the model output (e.g. removing probability/logit output), we remove the method of extracting weights by using finite differences. This significantly reduces the risk of model extraction. However, the tradeoff is in reduced model transparency.

Conclusion

Our project has successfully implemented the membership inference and model extraction privacy attacks on predictive AI models as well as the countermeasures. Through our demonstrations of the membership inference and model extraction attacks we showed how these attacks can be implemented and applied as well as the privacy risks associated. This serves to emphasize the privacy risks of AI models as well as the importance of having countermeasures when training and releasing any AI models to the public.

By implementing and evaluating various countermeasures to these two privacy attacks, we have demonstrated the effectiveness of these various safeguards as well as the tradeoffs to utility and model accuracy.

We hope that this project has demonstrated how we need to consider more than just model accuracy when training and designing models and the importance of building privacy safeguards into models, especially when training on personal information.

References

Carlini, N., Jagielski, M., & Mironov, I. (2020). *Cryptanalytic Extraction of Neural Network Models* (No. arXiv:2003.04884). arXiv. https://doi.org/10.48550/arXiv.2003.04884

Sablayrolles, A., Douze, M., Ollivier, Y., Schmid, C., & Jégou, H. (2019). *White-box vs Black-box: Bayes Optimal Strategies for Membership Inference* (No. arXiv:1908.11229). arXiv. https://doi.org/10.48550/arXiv.1908.11229

Vassilev, A., Oprea, A., Fordyce, A., & Anderson, H. (2024). *Adversarial machine learning: A taxonomy and terminology of attacks and mitigations* (No. NIST 100-2e2023; p. NIST 100-2e2023). National Institute of Standards and Technology (U.S.). https://doi.org/10.6028/NIST.AI.100-2e2023

Wolberg, W., Mangasarian, O., Street, N., & Street, W. (1993). Breast Cancer Wisconsin (Diagnostic) [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C5DW2B.