TML Assignment # 1

TEAM 16

Ahrar Bin Aslam

Muhammad Mubeen Siddiqui

Membership Inference Attack

In this assignment, we developed a Membership Inference Attack against an image classification model. The goal was to determine whether a given input was part of the victim model's training set or not using only the model's outputs and structure. Our final solution is based on an out-of-fold stacked ensemble, trained on features extracted from a ResNet18 victim model.

We iteratively improved our approach over multiple versions, each time learning from the limitations of the previous one and building a more accurate and reliable attack.

What Data Did We Have Access To?

As an attacker, we got access to:

• Resnet 18 model trained on an undisclosed dataset

• Public dataset that contains the ids, images, class labels, and membership information (1 == is member, 0 == not a member). Members are data points that were used in the training dataset to train the model, and non-members were not

• Private dataset that contains the ids, images, and class labels, with membership set to None, which means the membership is unknown. Note that some of the data points in this dataset were used to train the model (members) and some were not (non members). This setup mimics a realistic scenario where an adversary can monitor the model's behavior but doesn't have access to internal training details or the full training set.

Our Approach

You can find our full implementation in MIA_Pipeline_TML25_A1_16.py or MIA_Pipeline_TML25_A1_16.ipynb. Here's a summary of the process we followed:

We first tried the most basic approach: using the model's loss, confidence, or entropy for each sample to guess if it was a training member. This is based on the idea that models tend to be more confident for example, lower loss on training samples.

However, this didn't work well. The AUC hovered around 0.5 basically random guessing. It turns out these simple scores don't carry enough information.

Next, we extracted multiple features per sample:

- Loss

- Confidence
- Margin between top predictions
- Entropy
- Gradient norm
- Whether the prediction was correct

We trained an XGBoost classifier on these features. This worked much better AUC improved to around 0.61. XGBoost handled the tabular data well and could model the subtle differences between members and non-members.

Why XGBoost?

- It's fast and doesn’t overfit easily.
- Much better than using something like SVM, which would struggle with the size and complexity.

We then added:

- Internal activations from ResNet18's conv5 layer (512-dimensional)
- Detailed logit statistics (like mean, standard deviation, top-5 probabilities)
- Derived features like confidence / entropy, true_prob confidence, and more.

This gave the model a much richer view of how the network behaves internally. We used SelectKBest to keep only the top 50 most informative features. These additions gave us a consistent boost in performance.

Lastly, the best model we created used stacked ensembling.

Our base models:

XGBoost

LightGBM

CatBoost (with hyperparameters tuned using Optuna)

1. We trained each model using out-of-fold predictions. That means we made sure the predictions used to train the final model were from folds where the data wasn't seen during training. This prevents overfitting.
2. The final meta model was a simple logistic regression that learns how to combine the base models’ predictions.

We tried tuning XGBoost and LightGBM as well, but it was taking a lot of computational time, and since CatBoost showed the most promise in initial tests, we prioritized tuning it with Optuna to balance performance gains with resource constraints

Unlike naive ensembling, stacking gives the meta-model the ability to weight each base model's prediction based on real validation data. This added flexibility helped improve the final performance. Moreover, rather than relying on a 0.5 cutoff, we fine-tuned our decision threshold to get a high TPR (true positive rate) at a low FPR (false positive rate) which is much more meaningful in security contexts.

Final Metrics:

The final training metrics for the Membership Inference Attack (MIA):

Metric	Local Score	Leaderboard Score
AUC	0.7421	0.6506585555555556
TPR@FPR=0.05	0.1904	0.10233333333333333

In terms of AUC, we are 19th and in terms of TPR we are 21st.

Why Not Other Methods?

We tried neural networks briefly, but they were overfitting easily and were harder to train and calibrate for this kind of task. Boosted trees just worked better.

Additional Information:

Due to resource constraints, all experiments and model training were conducted on Google Colab utilizing its free GPU runtime, rather than running locally via VS Code or any desktop environment.
We have provided the complete implementation in both .ipynb (Notebook) and .py (Script) formats. Please note that the full pipeline execution takes approximately 20 minutes to complete on Colab.