mia Documentation

Release 0.1.2

Bogdan Kulynych, Mohammad Yaghini

Jan 31, 2019

Contents:

A library for running membership inference attacks (MIA) against machine learning models.

These are attacks against privacy of the training data. In MIA, an attacker tries to guess whether a given example was used during training of a target model or not, only by querying the model. See more in the paper by Shokri et al. Currently, you can use the library to evaluate the robustness of your Keras or PyTorch models to MIA.

Features:

- Implements the original shadow model attack
- Is customizable, can use any scikit learn's `Estimator`-like object as a shadow or attack model
- Is tested with Keras and PyTorch

CHAPTER 1

Getting started

You can install mia from PyPI:

```
pip install mia
```

CHAPTER 2

Usage

2.1 Shokri et al. attack

See the full runnable example. Read the details of the attack in the paper.

Let `target_model_fn()` return the target model architecture as a scikit-like classifier. The attack is white-box, meaning the attacker is assumed to know the architecture. Let `NUM_CLASSES` be the number of classes of the classification problem.

First, the attacker needs to train several *shadow models* —that mimick the target model— on different datasets sampled from the original data distribution. The following code snippet initializes a *shadow model bundle*, and runs the training of the shadows. For each shadow model, `2 * SHADOW_DATASET_SIZE` examples are sampled without replacement from the full attacker's dataset. Half of them will be used for control, and the other half for training of the shadow model.

```
from mia.estimators import ShadowModelBundle

smb = ShadowModelBundle(
    target_model_fn,
    shadow_dataset_size=SHADOW_DATASET_SIZE,
    num_models=NUM_MODELS,
)
X_shadow, y_shadow = smb.fit_transform(attacker_X_train, attacker_y_train)
```

`fit_transform` returns *attack data* `X_shadow, y_shadow`. Each row in `X_shadow` is a concatenated vector consisting of the prediction vector of a shadow model for an example from the original dataset, and the example's class (one-hot encoded). Its shape is hence `(2 * SHADOW_DATASET_SIZE, 2 * NUM_CLASSES)`. Each label in `y_shadow` is zero if a corresponding example was "out" of the training dataset of the shadow model (control), or one, if it was "in" the training.

mia provides a class to train a bundle of attack models, one model per class. `attack_model_fn()` is supposed to return a scikit-like classifier that takes a vector of model predictions `(NUM_CLASSES, )`, and returns whether an example with these predictions was in the training, or out.

```
from mia.estimators import AttackModelBundle

amb = AttackModelBundle(attack_model_fn, num_classes=NUM_CLASSES)
amb.fit(X_shadow, y_shadow)
```

In place of the `AttackModelBundle` one can use any binary classifier that takes `(2 * NUM_CLASSES, )`-shape examples (as explained above, the first half of an input is the prediction vector from a model, the second half is the true class of a corresponding example).

To evaluate the attack, one must encode the data in the above-mentioned format. Let `target_model` be the target model, `data_in` the data (tuple `X, y`) that was used in the training of the target model, and `data_out` the data that was not used in the training.

```
from mia.estimators import prepare_attack_data

attack_test_data, real_membership_labels = prepare_attack_data(
    target_model, data_in, data_out
)

attack_guesses = amb.predict(attack_test_data)
attack_accuracy = np.mean(attack_guesses == real_membership_labels)
```

CHAPTER 3

API

3.1 Estimators

Scikit-like estimators for the attack model and shadow models.

class mia.estimators.**AttackModelBundle**(*model_fn*, *num_classes*, *serializer=None*, *class_one_hot_coded=True*)

A bundle of attack models, one for each target model class.

Parameters

- **model_fn** – Function that builds a new shadow model
- **num_classes** – Number of classes
- **serializer** (*ModelSerializer*) – Serializer for the models. If not None, the models will not be stored in memory, but rather loaded and saved when needed.
- **class_one_hot_encoded** – Whether the shadow data uses one-hot encoded class labels.

fit(*X*, *y*, *verbose=False*, *fit_kwargs=None*)

Train the attack models.

Parameters

- **X** – Shadow predictions coming from `ShadowBundle.fit_transform()`.
- **y** – Ditto
- **verbose** – Whether to display the progressbar
- **fit_kwargs** – Arguments that will be passed to the fit call for each attack model.

class mia.estimators.**ShadowModelBundle**(*model_fn*, *shadow_dataset_size*, *num_models=20*, *seed=42*, *serializer=None*)

A bundle of shadow models.

Parameters

- **model_fn** – Function that builds a new shadow model
- **shadow_dataset_size** – Size of the training data for each shadow model
- **num_models** – Number of shadow models
- **seed** – Random seed
- **serializer** (*ModelSerializer*) – Serializer for the models. If None, the shadow models will be stored in memory. Otherwise, loaded and saved when needed.

fit_transform(*X*, *y*, *verbose=False*, *fit_kwargs=None*)
Train the shadow models and get a dataset for training the attack.

Parameters

- **X** – Data coming from the same distribution as the target training data
- **y** – Data labels
- **verbose** (*bool*) – Whether to display the progressbar
- **fit_kwargs** (*dict*) – Arguments that will be passed to the fit call for each shadow model.

Note: Be careful when holding out some of the passed data for validation (e.g., if using Keras, passing *fit_kwargs=dict(validation_split=0.7)*). Such data will be marked as "used in training", whereas it was used for validation. Doing so may decrease the success of the attack.

---

mia.estimators.**prepare_attack_data**(*model*, *data_in*, *data_out*)
Prepare the data in the attack model format.

Parameters

- **model** – Classifier
- **y) data_in** (*(X,)* – Data used for training
- **y) data_out** (*(X,)* – Data not used for training

Returns (X, y) for the attack classifier

3.2 Serialization

class mia.serialization.**BaseModelSerializer**(*model_fn*, *prefix='.'*, **args*, ***kwargs*)
ABC class for a model serializer.

Parameters

- **model_fn** – Function that builds a new model
- **prefix** – Path to the directory where models will be saved.

__metaclass__
alias of abc.ABCMeta

get_model_path(*model_id*)
Get the path to the model with given ID.

3.3 Wrappers

class mia.wrappers.**ExpLrScheduler**(*init_lr=0.001*, *decay_factor=0.1*, *lr_decay_every_epochs=7*, *verbose=False*)

Decay learning rate by a factor every *lr_decay_every_epochs*.

Based on https://discuss.pytorch.org/t/fine-tuning-squeezenet/3855/7

__call__(*optimizer*, *epoch*)

Call self as a function.

class mia.wrappers.**TorchWrapper**(*module*, *criterion*, *optimizer*, *module_params=None*, *optimizer_params=None*, *lr_scheduler=None*, *enable_cuda=True*, *serializer=None*)

Simplified Keras/sklearn-like wrapper for a torch module.

We know there's skorch, but it was a pain to debug.

Parameters

- **module** – Torch module class
- **criterion** – Criterion class
- **optimizer** – Optimizer class
- **module_params** (*dict*) – Parameters to pass to the module on initialization.
- **optimizer_params** (*dict*) – Parameters to pass to the optimizer on initialization.
- **lr_scheduler** – Learning rate scheduler
- **enable_cude** – Whether to use CUDA
- **serializer** (*ModelSerializer*) – Model serializer to save the best model.

fit(*X*, *y=None*, *batch_size=32*, *epochs=20*, *shuffle=True*, *validation_split=None*, *validation_data=None*, *verbose=False*)

Fit a torch classifier.

Parameters

- **X** (numpy.ndarray or torch.Tensor.) – Dataset
- **y** – Labels
- **batch_size** – Batch size
- **epochs** – Number of epochs to run the training
- **shuffle** – Whether to shuffle the dataset
- **validation_split** – Ratio of data to use for training. E.g., 0.7
- **validation_data** – If validation_split is not specified, the explicit validation dataset.
- **verbose** – Whether to output the progress report.

TODO: Add custom metrics.

fit_step(*batch*, *phase='train'*)

Run a single training step.

Parameters

- **batch** – A tuple of numpy batch examples and labels

- **phase** – Phase. One of ['train', 'val']. If in val, does not update the model parameters.

predict(*X*, *batch_size=32*)

Get the confidence vector for an evaluation of a trained model.

Parameters

- **X** – Data
- **batch_size** – Batch size

predict_proba(*X*, *batch_size=32*)

Get the confidence vector for an evaluation of a trained model.

Parameters

- **X** – Data
- **batch_size** – Batch size

TODO: Fix in case this is not one-hot.

class mia.wrappers.**TorchWrapperSerializer**(*model_fn*, *prefix*, *verbose=False*)

Torch wrapper serializer.

CHAPTER 4

Contributing

4.1 Dev setup

4.1.1 Install dev packages

Specify the `[dev]` option to install the development packages:

```
pip install -e ".[dev]"
```

4.1.2 Running tests

Use pytest to run all unit tests.

```
pytest
```

4.1.3 Building docs

Generate the docs:

```
cd docs
make html
```

You can then check out the generated HTML:

```
cd docs/build/html
python3 -m http.server
```

4.1.4 Formatting code

mia's code is formatted using black. Run the formatter as follows:

```
make format
```

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

m

Index

Symbols

A

B

E

F

G

M

P

S

T