184.702 Machine Learning – Exercise 3.1 Advanced topics in robustness / security / privacy of Machine Learning

Note: please first read the general overview document for the 3rd exercise!

TLDR: You chose either a topic from 3.1, OR 3.2, OR 3.3 - but not one from each!

Topic 3.1: Advanced topics in robustness / security / privacy of Machine Learning

You need to choose only one topic, don't be scared of the long document - it just means a lot of choices :-)

The aim of this type of exercise is to work in detail with one of the emerging topics in **robustness / security & privacy** of machine learning. Some of the topics are rather experimental as they are novel or based on very recent results, and the exact outcome can not always be predicted. Thus, the main aim is not to achieve perfect results, but to gain an interesting learning experience. We will grade your work based on your effort, and on the originality of your approach, even if you in some settings don't achieve good results.

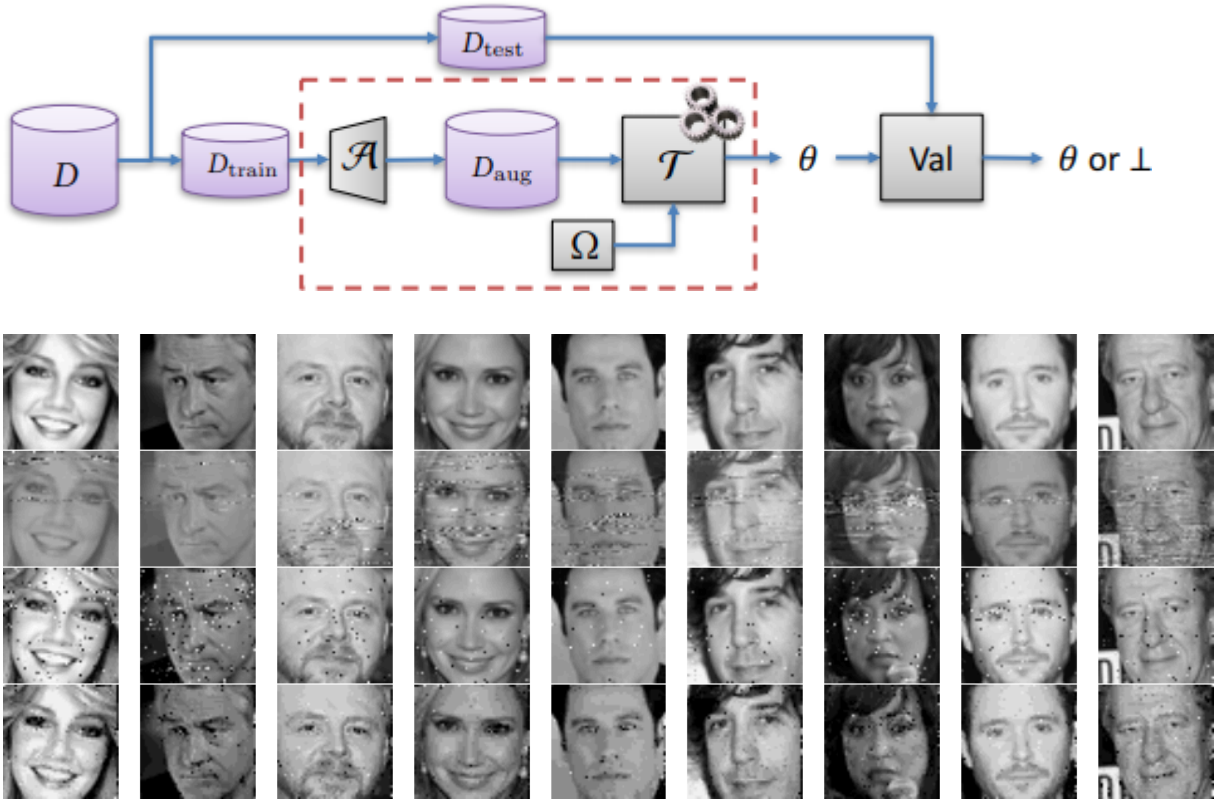
Table of Contents:

Adversarial Machine Learning

Some datasets that are mentioned in the rest of the topic description

- Yale Face Database http://vision.ucsd.edu/content/yale-face-database
- Labeled Faces in the Wild, http://vis-www.cs.umass.edu/lfw/, using the task for face recognition. See also https://scikit-learn.org/0.19/datasets/#the-labeled-faces-in-the-wild-face-recognition-dataset
- PubFig dataset, http://www.cs.columbia.edu/CAVE/databases/pubfig
- PubFig83: http://vision.seas.harvard.edu/pubfig83/
- AT&T / Olivetti Faces (https://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html, https://scikit-learn.org/stable/datasets/index.html#olivetti-faces-dataset)
- MNIST: http://yann.lecun.com/exdb/mnist/, https://scikit-learn.org/0.19/datasets/mldata.html
- CIFAR (10, 100): https://www.cs.toronto.edu/~kriz/cifar.html, https://github.com/ivanov/scikits.data/blob/master/datasets/cifar10.py
- FashionMNIST: https://github.com/zalandoresearch/fashion-mnist
- The German Traffic Sign Recognition Benchmark (GTSRB), http://benchmark.ini.rub.de
- ImageNet: http://image-net.org/download

Topic 3.1.1.1: Data exfiltration using ML Models



Data exfiltration (also called data extrusion or data exportation) is an unauthorized data transfer from a computer system, and a form of data theft.

In the setting of machine learning, an adversary tries to utilise a machine learning model to leak information about the training data to the end-user of the model. This can be interesting for an attacker when the model gets trained on private training data, and there is no other means to leak/exfiltrate information about the data to the outside (the attacker) than via the model. Hiding the information in the model is a form of steganography, where a message is hidden within another message.

In this task, you shall create experiments either in a white-box or black-box settings similar as described in the paper by *Song et al: Machine Learning Models that Remember Too Much. CCS 2017 (https://www.cs.cornell.edu/~shmat/shmat_ccs17.pdf); code is available either at https://github.com/csong27/ml-model-remember, or in a fork at*https://github.com/andreasiposova/ml-model-remember

There are several options for this topic:

A. Your task is to adapt either the two white-box attacks (Correlated Value Encoding Attack and Sign Encoding Attack), or the black-box Capacity Abuse Attack (black-box) from the code linked above to a model and data set for other neural network models, e.g. either (i) graph neural networks, (ii) Auto Encoders or similar architectures (e.g. Variational Autoencoders), (iii) (simple) diffusion models, or (iv) LSTMs or other advanced recurrent neural networks.

B. Implement defences against the Correlated Value Encoding (CVE) Attack or the Sign Encoding (SE) attack. The attack should destroy as much as possible the data encoded in the model, while at the same time keep the model useful, i.e. the effectiveness (accuracy, etc..) of the model on the original task it was trained for should not decrease too much (ideally, the defender can select a threshold).
   Basically, you can think of your strategy, but for CVE, you could consider perturbing the values in a way that the exfiltrated data is not readable anymore, because the values don't fit anymore to the values that represent the data. For SE, you can consider flipping the signs for some values, ideally those that have the least impact on the original task; you can determine that e.g. by checking which sign flipping introduces the least overall change in value (i.e. flipping values that are closest to zero), or by determining which weights are actually least relevant for the original task. You can perform the defence on any dataset / model combination you want (also the ones presented in the original paper)

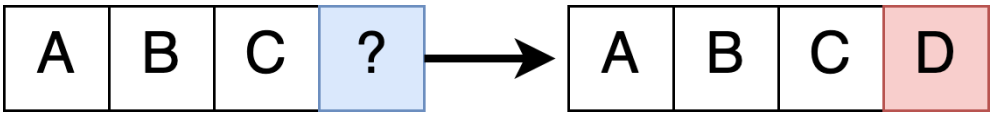
C. Implement the 'Least Significant Bit Encoding' attack on the 'Facescrub' image dataset also used in the original paper.
   Link to the Facescrub data set: http://vintage.winkler.site/faceScrub.zip
   The password for decrypting the zip file is: ICIP'14

Topic 3.1.1.2: Attribute Inference (disclosure) from ML Models

(note: this topic is rather experimental, and you will, as for all the other topics, graded on your effort, not necessarily on a (positive) outcome, which might not be achievable).

One type of information disclosure is called **attribute disclosure**: it consists of **inferring the value of an attribute** (e.g., ethnicity) **that was hidden** (i.e., not directly shared by the user).

Attribute disclosure is independent of identifying a specific record in a database - inferring additional information about a user from data shared by **other users**. This can be sensitive information about a user such as ethnicity or political affiliation, inferred by mining available data.



While attribute disclosure until now mostly considered the setting where a released **data set** is the base for inference, similar attacks might be possible from a **released machine learning model**, which invariably encodes some information about the training data.

Your task in this exercise is to test if this is possible for various machine learning models. Given an incomplete sample X and a trained model f(X), is it possible to infer the unknown values of X? You can first assume that X (with its full information) was also used in actually training the model, but further testing whether it would be also possible for an X not used in the training. One simple approach would be testing multiple values to "impute" the missing value, and observe the response from the model to that.

A similar approach as been tested to some extent in https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/fredrikson_matthew (even they refer to it there as "model inversion", it can be seen as a form of attribute inference, potentially with multiple attributes considered to be inferred): "... the algorithm simply completes the target feature vector with each of the possible values for x_1, and then computes a weighted probability estimate that this is the correct value."

adversary $\mathcal{A}^f(\mathsf{err}, \mathbf{p}_i, \mathbf{x}_2, \ldots, \mathbf{x}_t, y)$:

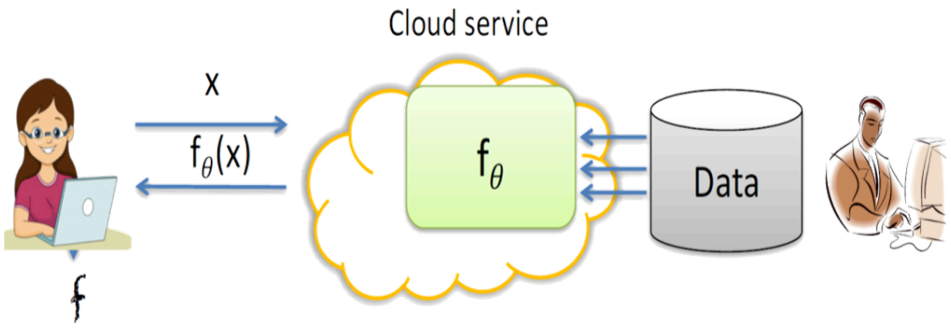
1: **for** each possible value v of $\mathbf{x}_1$ **do**
2: $\quad \mathbf{x}' = (v, \mathbf{x}_2, \ldots, \mathbf{x}_t)$
3: $\quad \mathbf{r}_v \leftarrow \mathsf{err}(y, f(\mathbf{x}')) \cdot \prod_i \mathbf{p}_i(\mathbf{x}_i)$
4: Return $\arg\max_v \mathbf{r}_v$

Figure 2: Generic inversion attack for nominal target features.

Try such an approach for a classification model; choose a relatively simple dataset, preferably with categorical data to limit the values to be tried, and models that return information on the confidence/likelihood of the predicted class (as you need this information to select the values that lead to the most determined outcome). Start with simple settings, i.e. where you want to infer only one attribute at the time (input attributes, not the target attribute, as you need that for comparison)

Using models that generally overfit to the training data seems more promising. As such, a neural network, or also a decision tree, might be good candidates.

Topic 3.1.1.3: Model Stealing / Extraction



Model "stealing" means to obtain a trained model from a source that generally only provides a "prediction API", i.e. a means to obtain a prediction from the model when providing a specific input, but does NOT disclose the actual model (i.e. "ML-as-a-service"). This might be because the model may be deemed confidential due to their sensitive training data, commercial value, or use in security applications. A user might train models on potentially sensitive data, and then charge others for access on a pay-per-query basis.

For this exercise we only consider a situation when an adversary with black-box access to the target model aims to steal the functionality of the model.

Regarding the implementation, you shall choose one of the papers described below together, the goal of the exercise depends on a paper:

A. Tramèr et al. Stealing Machine Learning Models via Prediction APIs. (2016). (https://www.usenix.org/system/files/conference/usenixsecurity16/sec16_paper_tramer.pdf)

Implementation: https://github.com/ftramer/Steal-ML

For this paper, you shall compare two attack strategies provided in the paper: equation-solving and retraining (there are three retraining approaches described in the paper, you shall use at least one). Implementation details:

- Pick at least two models among (Multiclass) Logistic Regression, MLP
- Use at least two tabular datasets, for instance from UCI ML Laboratory
- Train target models on picked datasets
- Perform equation-solving and retraining attacks to observe (at least) two substitute models per a target model
- Compare the effectiveness of the attacks using the following metrics: accuracy, fidelity, transferability (optional)
- Compare the efficiency of the attacks by providing for each attack two characteristics: the number of target model parameters and the number of queries you made in order to steal a model

B. Orekondy et al. Knockoff Nets: Stealing Functionality of Black-Box Models (https://arxiv.org/pdf/1812.02766.pdf)

Implementation: https://github.com/tribhuvanesh/knockoffnets

For this paper, you shall consider different adversary knowledge while performing the substitute training attack and compare the results. More precisely, you shall evaluate the impact of knowing problem-domain data and the target model architecture. Implementation details:

- Pick three types of datasets: original data for a target model training (e.g., MNIST), problem-domain data for a substitute model training (e.g., SVHN) and auxiliary data for a substitute model training (e.g., ILSVRC)
- Pick an architecture for the target model (CNN)
- Train the target model on the original dataset
- Train a substitute model with the same architecture using problem-domain data
- Train another substitute model with the same architecture using auxiliary data
- Pick another architecture for a substitute model (CNN)
- Train a substitute model with different architecture using problem-domain data
- Train another substitute model with different architecture using auxiliary data
- Evaluate the effectiveness performance of the observed substitute models using the following metrics: accuracy, fidelity, transferability (optional)
- Evaluate the efficiency performance of the observed substitute models by providing for each model two characteristics: the number of target model parameters and the number of queries you made in order to steal a model

C. *Correia-Silva et al. Copycat CNN: Stealing Knowledge by Persuading Confession with Random Non-Labeled Data* (https://arxiv.org/pdf/1806.05476.pdf)

Implementation: https://github.com/jeiks/Stealing_DL_Models

For this paper, you shall evaluate the impact of knowing the data domain for a substitute model training. Implementation details:

- Pick three types of datasets: original data for a target model training (e.g., MNIST), problem-domain data for a substitute model training (e.g., SVHN) and auxiliary data for a substitute model training (e.g., ILSVRC)
- Pick an architecture for the target model (CNN)
- Train the target model on the original data
- Train a substitute model using non-problem domain data
- Train a substitute model using problem-domain data
- Train another substitute model using auxiliary data
- Try a combinations of datasets: train a substitute model using a dataset, where 10% of data is original (problem-domain) and the rest is non-problem domain dataset
- For all dataset settings for a substitute model training, try to use only part of the datasets to see if you can reach the same performance using a small number of queries
- Evaluate the effectiveness performance of the observed substitute models using the following metrics: accuracy, fidelity, transferability (optional)

- Evaluate the efficiency performance of the observed substitute models by providing for each model two characteristics: the number of target model parameters and the number of queries you made in order to steal a model

For substitute models training, you can either assume that an architecture of the target model is known, or use another architecture (but use the same architecture for all experiments!).

E. Stealing Graph-Neural Networks

Addressing a specific type of data and models, graph neural network stealing is less explored. In this task, try to re-implement one of the following approaches, and apply it to slightly different data and/or models:

- Bang Wu, Xiangwen Yang, Shirui Pan, and Xingliang Yuan. Model Extraction Attacks on Graph Neural Networks: Taxonomy and Realisation. In ACM Asia Conference on Computer and Communications Security, ASIA CCS, pages 337–350, Nagasaki Japan, May 2022. ACM.
- [80] Xinlei He, Jinyuan Jia, Michael Backes, Neil Zhenqiang Gong, and Yang Zhang. Stealing Links from Graph Neural Networks. In USENIX Security Symposium, Vancouver, B.C., Canada, August 2021. USENIX Association.
- Yun Shen, Xinlei He, Yufei Han, and Yang Zhang. Model Stealing Attacks Against Inductive Graph Neural Networks. In IEEE Symposium on Security and Privacy (SP), pages 1175–1192, San Francisco, CA, USA, May 2022. IEEE.
- David DeFazio and Arti Ramesh. Adversarial Model Extraction on Graph Neural Networks. In International Workshop on Deep Learning on Graphs: Methodologies and Applications, DLGMA, New York, NY, USA, February 2020.

F. Stealing other Model Types

There are further attacks trying to steal other types of models, e.g. recurrent or generative models; you might try to re-implement one of the following papers, and, as above in “E”, adapt it to a slightly different dataset or models:

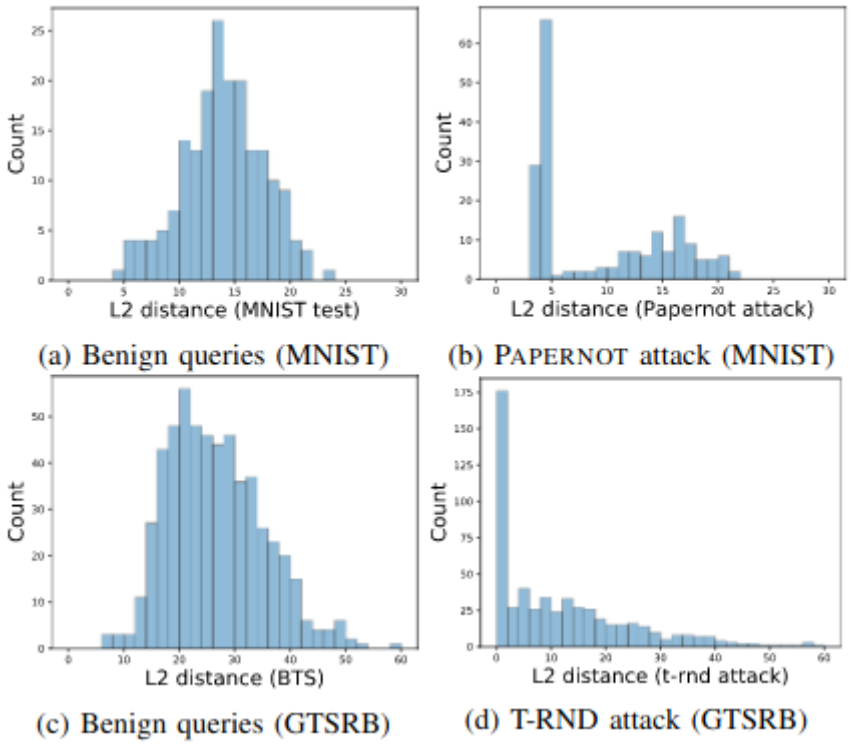
- Tatsuya Takemura, Naoto Yanai, and Toru Fujiwara. Model Extraction Attacks against **Recurrent Neural Networks**, 2020. arXiv:2002.00123.
- Kalpesh Krishna, Gaurav Singh Tomar, Ankur Parikh, Nicolas Papernot, and Mohit Iyyer. Thieves of Sesame Street: Model Extraction on **BERT**-based APIs. In Int. Conf. on Learning Representations (ICLR), Virtual Event, 2020.
- Vahid Behzadan and William Hsu. Adversarial Exploitation of **Policy Imitation**, 2019. arXiv:1906.01121 (Reinforcement Learning)
- Hailong Hu and Jun Pang. Stealing Machine Learning Models: Attacks and Countermeasures for **Generative Adversarial Networks**. In Annual Computer Security Applications Conf. (ACSAC), Virtual Event USA, 2021. ACM
- Sebastian Szyller, Vasisht Duddu, Tommi Gröndahl, and N. Asokan. Good Artists Copy, Great Artists Steal: Model Extraction Attacks Against **Image Translation Generative Adversarial Networks**, 2021. arXiv:2104.12623.

- Yupei Liu, Jinyuan Jia, Hongbin Liu, and Neil Zhenqiang Gong. StolenEncoder: Stealing Pre-trained **Encoders** in Self-supervised Learning. In ACM SIGSAC Conf. on Computer and Communications Security (CCS), Los Angeles, CA, USA, 2022. ACM.
- Zeyang Sha, Xinlei He, Ning Yu, Michael Backes, and Yang Zhang. Can't Steal? Cont-Steal! Contrastive Stealing Attacks Against **Image Encoders**, 2022. arXiv:2201.07513.

Topic 3.1.1.4: Detection of Model Stealing

The idea of this task is to analyze the difference between distributions of data from the initial problem domain, and data generated by the adversary, e.g. synthetic/adversarial samples. The following steps should be done:

- Pick a dataset and explore its distribution. We propose to pick one of the following datasets: MNIST, FashionMNIST, PubFig83, GTRSB, CIFAR-10
- Use different techniques, such as
    - Adversarial image generation: FGSM, IGS, FGV, Deepfool, C&W, or JSMA (as e.g. in https://arxiv.org/pdf/1809.04913.pdf)
    - Data composition (https://elidavid.com/pubs/stealing-knowledge.pdf) or
    - Random noise inputs (https://arxiv.org/pdf/1912.08987.pdf)
- For each technique, explore the distribution of generated samples and compare it to the original data distribution. Also data from a non-problem-domain dataset can be analyzed
- Implement a detection (or use (or modify) an existing one such as described in https://github.com/SSGAalto/prada-protecting-against-dnn-model-stealing-attacks), to evaluate which adversarial/synthetic samples can be recognized by distribution analysis



(a) Benign queries (MNIST) (b) PAPERNOT attack (MNIST)

(c) Benign queries (GTSRB) (d) T-RND attack (GTSRB)

Topic 3.1.1.5: Watermarking ML/DL models

Sharing trained models has brought many benefits to development of ML and DL systems. However, training models is usually a task that requires vast resources, from data to time and computing power. **Watermarking** can help the owners of such models mark their property in order to trace unauthorised usage or redistribution.

We have collected and unified a number of watermarking techniques at https://sbaresearch.github.io/model-watermarking/#/, together with

- Non-watermarked pre-trained models for two datasets and several architectures (CIFAR-10 and MNIST)
- Attacks on watermarking

You can choose between two options in this task

Topic 3.1.156.A:

Work with one additional **watermarking technique not yet in that repository**, either a white-box or a black-box watermarking technique, e.g. from the following techniques:

- White-box Approaches
    - Uchida et al.: Embedding Watermarks into Deep Neural Networks (https://dl.acm.org/doi/pdf/10.1145/3078971.3078974)
        - Implementation: https://github.com/yu4u/dnn-watermark)
    - Rouhani et al.: DeepSigns: An End-to-End Watermarking Framework for Ownership Protection of Deep Neural Networks (https://dl.acm.org/doi/10.1145/3297858.3304051)
        - Implementation: https://github.com/Bitadr/DeepSigns)
    - Feng et al.: Watermarking Neural Network with Compensation Mechanism (http://link.springer.com/10.1007/978-3-030-55393-7_33)
    - Wang et al.: RIGA: Covert and Robust White-Box Watermarking of Deep Neural Networks (http://arxiv.org/abs/1910.14268)
- Black-Box Approaches
    - Huili Chen, Bita Darvish Rouhani, Farinaz Koushanfar: BlackMarks: Blackbox Multibit Watermarking for Deep Neural Networks (https://arxiv.org/abs/1904.00344)

Either: implement the chosen technique or utilize and adapt an open source implementation.

In both cases: in order to demonstrate the success of the chosen watermarking technique, evaluate experimentally the following three requirements. You can use the experiments from the paper of the chosen technique as a guidance.

- **effectiveness**: after embedding the watermark, it also can be successfully extracted and verified, thus the model owner can claim ownership in case of a threat event. Analyze if the chosen technique satisfies this requirement.
- **fidelity**: the watermark embedding has only minimal influence on the model's performance. Compare the model's accuracy before and after embedding the watermark.

- **robustness**: the watermark should be robust against various types of attacks, e.g. fine-tuning, pruning, transfer learning. In your experiments, perform **one** attack and analyse the robustness of the watermarking technique with respect to this specific attack.

Choose one dataset (CIFAR-10 or MNIST) from the template examples provided in the github repository, and perform the above steps on at least two of the corresponding models; you can use the non-watermarked model as a basis for embedding, and the fine-tuning or pruning attack method.

If you chose to implement a method for which no code is provided, you just need to show it only on one architecture and one dataset.

In TUWEL in the topic registration, specify the architecture(s) and dataset(s) used in the experiments. In the report describe your methodology in detail, explain your findings and provide the source code of your implementation.

Topic 3.1.1.5.B:

Test whether a watermarked model would preserve the watermark becoming the when target of a model-stealing attack; to this extent, train a model with a well-embedded watermark, and then use one of the implementations from Topic 3.1.1.3 to steal the model; afterwards, check if the watermark is still present. You can try any of the trigger-methods mentioned in https://ieeexplore.ieee.org/document/10143370/, i.e. out-of-distribution, pattern, etc…

Topic 3.1.1.6: Membership Inference on Synthetic Data

Membership inference attacks on synthetic data aim to determine whether a specific record known by the attacker was included in the training data used to generate the synthetic data. This allows the attacker to infer sensitive information on a record. For example, if the original dataset comes from a hospital and contains records of patients with a particular diagnosis, confirming that a record is part of the dataset implies that the individual has that diagnosis.

Multiple approaches have been proposed in the literature to perform membership inference attacks. Depending on the level of access to the generator, these approaches can be classified as: white box (full access to the synthetic data generator), black box (limited query access to the generator), and no-box. In the no-box setting (also called full black-box setting), the attacker only has access to a generated synthetic dataset and might also have auxiliary knowledge, for instance, a dataset with a similar distribution to the original data, but can not interact with the model, i.e. can also not generate more synthetic data samples.

The task of this exercise is to evaluate the effectiveness (Attack Success Rate, AUC Score) and efficiency (runtime) of two no-box attacks on synthetic datasets, generated using different synthetic data generators. Some possible approaches for the no-box attacks include (but are not limited to) the following:

- van Breugel et al.: Membership Inference Attacks against Synthetic Data through Overfitting Detection (https://proceedings.mlr.press/v206/breugel23a.html)
- Chen et al.: GAN-Leaks: A Taxonomy of Membership Inference Attacks against Generative Models (https://dl.acm.org/doi/10.1145/3372297.3417238)
- Zhang et al.: Membership inference attacks against synthetic health data (https://www.sciencedirect.com/science/article/pii/S1532046421003063)
- Guepin et al.: Synthetic Is All You Need: Removing the Auxiliary Data Assumption for Membership Inference Attacks Against Synthetic Data (https://link.springer.com/chapter/10.1007/978-3-031-54204-6_10)

You can re-use any existing code or implement the attacks from scratch by yourself. Useful libraries are the following:

- TAPAS (https://github.com/alan-turing-institute/tapas)
- IBM Toolkit: (https://github.com/IBM/ai-privacy-toolkit/tree/main/apt/risk/data_assessment)
- Synthcity (https://github.com/vanderschaarlab/synthcity/blob/main/src/synthcity/metrics/eval_privacy.py)

Compare the results from the two approaches, e.g. on which data samples do they agree, on which do they differ?

Topic 3.1.7: Identifying source synthetic data generator

(note: this topic is rather experimental, and you will, as for all the other topics, graded on your effort, not necessarily on a (positive) outcome, which might not be achievable).

Synthetic data can be generated using various approaches, such as marginal-based models (e.g., Bayesian Networks), neural networks (e.g., Generative Adversarial Networks [GANs], Variational Autoencoders [VAEs], and Large Language Models [LLMs]), and tree-based models (e.g., Classification and Regression Trees [CART]). Each model has different characteristics and capabilities to learn the patterns and relationships in the data and replicate them as closely as possible. An interesting open question is whether it is possible to identify the generation method that was used given only access to the generated synthetic data. This can be the pathway to further attacks, e.g. membership inference.

In this task, you will be given synthetic datasets generated using specific models (a predefined group will be indicated). Your goal is to identify which model was used to generate each dataset and explain the reasoning behind your conclusions. One approach could e.g. be to create, from different datasets, a range of synthetic datasets with the generator methods indicated, and then extract dataset specific features from these, to in the end learn a meta-model that can identify the most likely method from an unknown dataset. This approach is to some extent comparable to meta-learning for identifying the best algorithm for a given ML problem (see e.g. the slides on meta-learning from the ML lecture, also posted in the TUWEL course of this lecture).

- https://arxiv.org/abs/1807.09173

Topic 3.1.1.8: Backdoor/poisoning Attacks & defence

Poisoning/backdoor attacks modify the training data to **embed a specific pattern** in some images (e.g. a yellow block on a STOP sign), and falsely label that image as a different class (e.g. as a speed limit 50 sign) to make the classifier learn this pattern for wrong predictions. These attacks generally work because a certain number of neurons in the network can be dedicated to learn these patterns, often because the number of neurons is larger than what is required to actually represent the pattern in the "clean" training data.

We will provide you with two datasets with manually prepared backdoor images, on the German Traffic Sign Recognition Dataset, and Yale Faces. You shall then train poisoned models on them.

You shall then re-create experiments from two of these proposed defences, and compare them to each other.

- Edward Chou, Florian Tramèr, Giancarlo Pellegrino. SentiNet: Detecting Localized Universal Attacks Against Deep Learning Systems; https://arxiv.org/abs/1812.00292
- Huili Chen, Cheng Fu, Jishen Zhao, Farinaz Koushanfar. DeepInspect: A Black-box Trojan Detection and Mitigation Framework for Deep Neural Networks; https://www.ijcai.org/proceedings/2019/647
- Jacob Steinhardt, Pang Wei Koh, and Percy Liang. Certified defenses for data poisoning attacks. In Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS'17, pages 3520–3532, USA, 2017. Curran Associates Inc.
- Or another anomaly-detection based approach by Yuntao Liu, Yang Xie, and Ankur Srivastava. Neural trojans. 2017 IEEE International Conference on Computer Design (ICCD), pages 45–48, 2017.
- Brandon Tran, Jerry Li, Aleksander Madry. Spectral Signatures in Backdoor Attacks, Neurips 2018. https://arxiv.org/abs/1811.00636
- Nathalie Baracaldo, Bryant Chen, Heiko Ludwig, Jaehoon Amir Safavi, and Rui Zhang. Detecting poisoning attacks on machine learning in iot environments. In 2018 IEEE

International Congress on Internet of Things, ICIOT 2018, San Francisco, CA, USA, July 2-7, 2018, pages 57–64. IEEE Computer Society, 2018.
    - Note: be careful to craft your attack also in a way that some of the data parts contain the backdoors, while the others are clean, so that there is something to separate for the provenance

- *Any other new defence that you find and that sounds interesting!*

You can re-use any existing code, or implement the methods from scratch by yourself.

Privacy-Preserving Data Publishing

Topic 3.1.3.1.: k-anonymity: generalization vs microaggregation

k-anonymity is a property of a dataset that contains the same information for at least *k* individuals, for every distinct information. *k*-anonymous datasets are obtained by either:

- a systematic generalization or suppression of the values of the dataset (global, local)
- microaggregation of the clustered values

Global and local transformation are types of generalization/suppression approach. Global transformation is achieved by generalizing the entire attribute (column) to the same generalization level from its hierarchy. Local transformation, however, aims to reduce the value to the least necessary generalization level. The locally anonymized dataset may contain values from different levels of generalization within one column (more information and examples at [1]).

Microaggregation is an approach that does not require defining generalization hierarchies for each quasi-identifier, but rather clusters the rows based on similarity and then generates the anonymized values as a result of some aggregation function (e.g. mean for numerical attributes, median for categorical, ...) [2].

These methods reduce the amount of information contained; therefore *k*-anonymous datasets necessarily have lower utility compared to their originals. One way to measure utility is via data performance on a particular machine learning task.

The aim of this exercise is to compare the utilities of the anonymized datasets obtained by different approaches:

1. Global transformation
2. Local transformation
3. Microaggregation

To that end, define classification tasks you want to perform and choose at least 3 different types of classifiers. Calculate the performance on the original dataset and then perform the same tasks using the anonymized datasets. Compare performance of ML classifiers trained on anonymized data to the ones trained on original data and compare how different anonymization approaches influence the performance.

The following anonymization tools provide different types of anonymizations:

- Generalization/suppression:
    a. Flash (within ARX tool; Java) – API: https://github.com/arx-deidentifier/arx (global/local)
    b. SaNGreeA (Python): https://github.com/tanjascats/sangreea (local)
    c. Mondrian (Python): https://github.com/qiyuangong/Mondrian (global)
    d. implementations from UTD Anonymization Toolbox (Java): http://cs.utdallas.edu/dspl/cgi-bin/toolbox/index.php?go=home (global)
    e. Crowds (Python): https://pypi.org/project/crowds/
    f. sdcMicro (R): https://cloud.r-project.org/web/packages/sdcMicro/index.html
- Microaggregation:
    a. (use) sdcMicro-microaggregation (R): https://cloud.r-project.org/web/packages/sdcMicro/index.html
    b. (adapt) SaNGreeA (Python): https://github.com/tanjascats/sangreea

    - To this end, utilize SaNGreeA algorithm for clustering the dataset into partitions of size at least *k*. Then modify the algorithm so that the anonymized values are microaggregated instead of generalized (i.e. for each cell calculate the mean value of the range if it is a numerical attribute and median (most frequent) value if it is a categorical attribute)
- You can also try to find other open-source anonymization tools

For your analysis choose 3 datasets with privacy implications and different characteristics (number of instances, number of attributes, types of attributes, etc.).

Specify in TUWEL at the topic registration which datasets will be used for the experiments. Write a report including all your findings, explain your methodology in detail and elaborate the conclusions.

References:

[1] https://sdcpractice.readthedocs.io/en/latest/anon_methods.htm

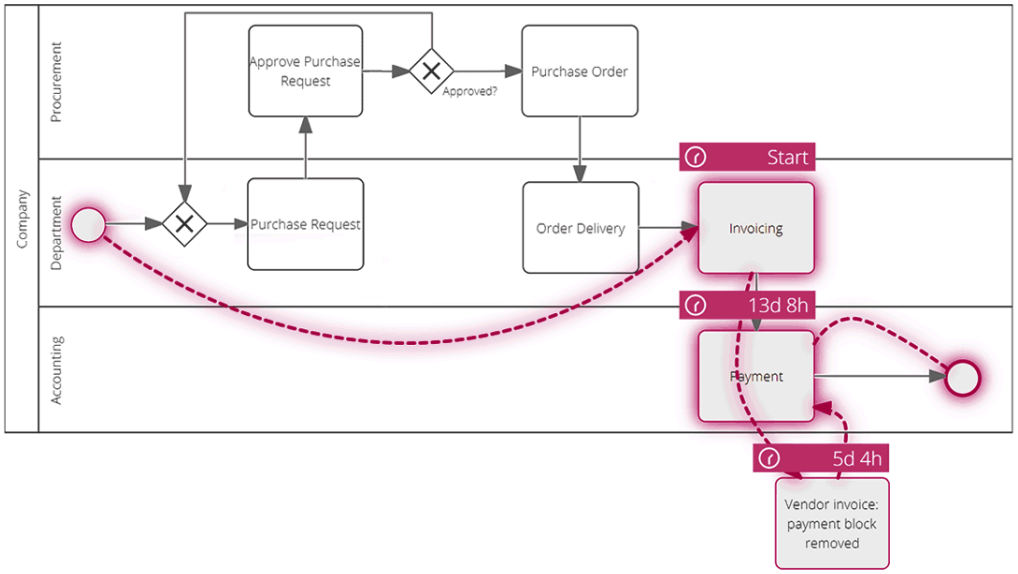
[2] J. Domingo-Ferrer and V. Torra, “Ordinal, Continuous and Heterogeneous k-Anonymity Through Microaggregation,” *Data Min Knowl Disc*, vol. 11, no. 2, pp. 195–212, Sep. 2005, doi: 10.1007/s10618-005-0007-5.

Secure / Federated / Distributed Computation

Distributed computation in general addresses privacy / data protection issues by not centralising data that is initially already partitioned between several parties, and either computing functions securely w/o revealing the input (secure multi-party computation), or by exchanging only aggregated values, such as gradients or other model parameters (federated learning), or by computing on encrypted data (homomorphic encryption).

In this task, you shall evaluate the effectiveness and efficiency of these approaches. An evaluation on the quality of the models, as compared to having a model trained on all the different partitions of the data, shall be performed. - i.e. observe the effectiveness and efficiency on the centralised vs. the distributed approach.

Topic 3.1.2.1: Federated (Distributed) Learning for Process Mining



Process mining tries to learn the structure of a process from logs or other sources about previous executions of process. This is typically done on centralized data, i.e. with centralized learning. Collaborative learning of such a process model from different organisations can be an interesting setting in many cases. However, instead of centralising the data, which might not be an option due to confidentiality reasons, federated learning can become a viable approach.

In this task, you shall perform experiments for distributed process mining. To this end, take an existing benchmark dataset from e.g. https://data.4tu.nl/repository/collection:event_logs_real, and first perform process mining in a centralised way with an existing approach (e.g. using ProM, http://www.promtools.org/). Then, try to federate / distribute the process, by first distributing the data, and then the learning. Data distribution could be done in uniform (each subset is roughly similar) and non-uniform ways (some aspects of the data are present only at one node).

One approach for distributed learning you could follow is: “On the Feasibility of Distributed Process Mining in Healthcare”, 2019. https://link.springer.com/chapter/10.1007/978-3-030-22750-0_36

Topic 3.1.2.3: Federated Learning on text data (or other sequential data)

In this task, you shall work with federated approaches for text data utilising RNNs (LSTM, GRU, …). Take a task of your choice (e.g. a text categorisation / classification task like the Reuters Dataset, or 20 newsgroups, etc..), and first establish a baseline with a state-of-the-art recurrent neural network, comparing your results to literature.

Then, simulate a federated setting by distributing the data on multiple sources, and compare your results to a centralised setting, similar to the setting in the topic above.

Topic 3.1.2.4: Federated Learning, different aggregation methods

There are several ways one can aggregate model parameters in Federated learning. In this task you will need to compare different aggregation methods for federated learning. You can take FedAvg (Federated averaging algorithm) as one of the most popular algorithms, also try weighted averaging and find 1-2 other aggregation algorithms. The result work should include evaluation of different aggregation methods in the settings with different numbers of nodes. You can choose 2 datasets for this task and horizontally partitioned data (nodes have different instances in their data but with the same features). You should fix the dataset size and distribute the data among the considered number of nodes. Try to create unequal and unbalanced data distribution to be more close to the real setting. For classification tasks use neural network algorithms. You can start searching for different aggregation methods in this paper: https://arxiv.org/pdf/1907.09693.pdf (pages 12-15)

Topic 3.1.2.6: Free-riders Detection in Federated Learning

Free-riders are clients that participate in a federated learning setup, but only contribute no or minimally to the final model, e.g. because they (i) have no (meaningful) data they can learn on locally, or (ii) they do not want to invest the computing resources, or for other reasons. Instead of training the model on their local data, they thus try to emulate and pretend they did a local computation, e.g. by training on random/public/synthetic/… data, or by estimating how they models would update, so that they can appear like benign participants to the federated learning setup and obtain the final model. In this task, you shall experiment with methods to detect fee-riders in a federated setup. Specifically, select one of the following detection methods and evaluate their effectiveness:

- Collaborative Fair Federated Learning (CFFL) by Lyu et al.: Lingjuan Lyu, Xinyi Xu, Qian Wang, and Han Yu. Collaborative Fairness in Federated Learning, pages 189–204. Springer International Publishing, Cham, 2020; https://github.com/XinyiYS/CollaborativeFairFederatedLearning
- The Multi Arm Bandit (MAB) detection introduced by Lu et al.: Renhao Lu, Weizhe Zhang, Hui He, Qiong Li, Xiaoxiong Zhong, Hongwei Yang, Desheng Wang, Shi Lu, Yuelin Guo, and Zejun Wang. Two-stage client selection for federated learning against free-riding attack: A multi-armed bandits and auction-based approach. IEEE Internet of Things Journal, 2024. https://ieeexplore.ieee.org/document/10606036/
- He et al. analyse clients which have little variety in their datasets, called ‘Maverick’: Jiatong He, Libing Wu, Zhuangzhuang Zhang, Na Lu, and Xuejiang Wei. Client evaluation and revision

in federated learning: Towards defending free-riders and promoting fairness. In Lecture Notes in Computer Science, volume 14964 LNCS, pages 100 – 118, Jinhua, China, 2024. https://link.springer.com/chapter/10.1007/978-981-97-7241-4_7

- Nguyen et al. using log analysis: Huong Nguyena, Hong-Tri Nguyenb, Lauri Lovéna, and Susanna Pirttikangasa. Stake-driven rewards and log-based free rider detection in federated learning. https://easychair.org/publications/preprint/ZnbG/download, 2024.

Topic 3.1.2.7: Attacking Free-riders Detection in Federated Learning

Wu et al. introduce the framework VPFL that contains a black-box watermarking method to detect free riders. In each epoch, the aggregator sends a black-box watermarking task in addition to the global model. The clients then train their local model to also contain the black-box watermark. The aggregator checks each local model against a validation set to determine whether the black-box watermark exists. This is done by comparing the expected values with the actual values. If the actual values do not deviate more than a certain threshold from the expected values, they are considered to be real updates. Otherwise, the client is marked and considered to be malicious after a certain number of marks. This detection thus will be able to detect all free riders that do not perform any kind of training. Free riders that are willing to perform some training however could integrate the watermark. Concretely, they could use the current global model and train locally using the data samples received for integrating the watermark. This way, they might be able to pass the check.

Your task is to implement this attack against the check, and test if it fools the detection method.

Yuanxiang Wu, Hang Cheng, Lin Guan, Peihao Liu, Fei Chen, and Meiqing Wang. Vpfl: A verifiable property federated learning framework against invisible attacks in distributed iot. In 2023 IEEE International Conference on High Performance Computing & Communications, Data Science & Systems, Smart City & Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys), 2023. https://ieeexplore.ieee.org/document/10466884