184.702 Security, Privacy and Explainability in Machine Learning – Exercise 2

Introduction

The second exercise allows you to choose from different topics from one of the lectures on

1. Privacy-preserving data publishing (Lecture 1 & 2)
2. Privacy-preserving computation (Lecture 3)
3. Adversarial Machine Learning (Lecture 4, 5 & 6)

There are several topic **suggestions** in this document, but you are also **free to suggest other topics that fit with the lecture subjects**. If so, please don't hesitate to contact me (mayer@ifs.tuwien.ac.at) and discuss it with us. If we see it fitting to the topics of the course, you'll get our ok to work on your idea.

You need to choose one of the topics, not one from each lecture topic!

The descriptions of the different topics are **not meant as complete specifications**, they just summarise the general idea and goals. If there is an unclarity, please do not hesitate to ask!

The exercises shall be done in groups of 2 students. Please arrange with your colleagues, and then register for a group **and** for your topic in TUWEL.

There is an online submission of the report and other artefacts that you created during this exercise (code, data, …).

As some of the more specialised tasks are also less explored and thus less predictable in outcome, we will not grade you based on results alone.

If you already did / are doing one of these topics also as part of "184.702 Machine Learning – Exercise 3.1", then you need to choose a different topic (or build on top of the results from there, but with a novel contribution in this course).

Solution and submission

This exercise is supposed to be solved with all the steps being programmatically, i.e. there shouldn't be a need for GUI or some other manual processing of data (but it should rather be scripted).

Your submission should contain

- **A zip file with all needed files** (your source code, your code compiled, data sets used (but **NOT** the ones **we provide** to you), a build script that resolved dependencies, or include any libraries you are using. Your submission needs to be self-contained!
    - If applicable, provide a means to compile your classes, preferably with a build file (e.g. Maven or ANT if you use Java, requirements file for python, etc..)

    - If your build file allows for automatically obtaining the dependencies, then they don't need to be included, otherwise please include them.
    - Provide a short how-to explaining the way to start your program (which is the main class/file, which command-line options does it expect, ..).
    - Please also state clearly what is the version of the software package(s) you use (ideally as already mentioned above declaratively, i.e. a build / requirements file); unless for a specific reason, please work with the latest versions (you can of course e.g. work in python 2, but don't work with old versions of libraries etc.)
- **A report, describing**
    - Your task
    - Your solutions (also describe failed approaches!)
    - Your results, evaluated on different datasets, parameters, ...
    - An **analysis of the results**
- Where applicably, your program shall be configurable via command-line options or a configuration file, to modify parameters, evaluation types, etc… i.e. it should not be needed to modify the code to change these options. There are many framework for command-line-options available, you don't need to code this yourself (e.g. for Java, Apache Commons CLI (http://commons.apache.org/cli/, http://martiansoftware.com/jsap/).

Just to clarify once more: you need to choose ONE topic, not all of them :-)

Datasets

Some of the datasets mentioned in the various task descriptions:

- 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

Computing resources

If you are in the need of computing resources, you can contact me (mayer@ifs.tuwien.ac.at) especially for CPU power on a Linux server @TU, but also for GPU resources (there is a cluster with high-end graphics cards that requires booking the resources via an online form)

Further, you can try infrastructure-as-a-service offers, as some Cloud operators have some free starter credit. E.g the Google Cloud Computing platform gives you 300 USD if you register (you need to provide your credit card, but they don't charge) which you can use to create remote machines with various OS preinstalled. Also other providers have similar offers. Make sure to shut down your machines when not using them, to avoid being charged. Other options are listed e.g. at https://code-love.com/2020/08/08/where-to-get-free-gpu-cloud-hours-for-machine-learning/

List of topics

Privacy-Preserving and Traceable Data Publishing

Topic 2.1.1.: Watermarking / Fingerprinting graph data

Watermarking is an information hiding technique that allows identifying the source of digital objects by embedding secret owner-specific information into the dataset. Fingerprinting extends the functionality of watermark by providing the identification of the source of unauthorised data leakage. Fingerprint combines thus secret owner-specific and recipient-specific information embedded in a specific release of a digital object.

One of the early techniques [KZ00] is proposed for watermarking geographic maps represented by a graph structure. The technique is focused on a specific black-box scenario, where access to the data is only possible by querying the distance between the point A to point B. The data owner distributes the map among multiple recipients, and depending on which distributed map is used, the query will return a slightly different distance -- making this in fact a fingerprinting technique by our terminology. Differences in query responses are achieved by introducing negligible modifications to edge lengths for each distributed map.

Authors of [ZLZZ15] provide a watermarking technique for a white-box access scenario. The proposed method is based on producing a small signature graph generated from the original graph, its own secret key and recipient's key and embeds it into the original graph, creating this way the modified version to be shared. This approach also falls under the umbrella of fingerprinting graph data because every distributed graph contains different modifications, therefore the recipient can be identified.

In your exercise, pick 2 graph datasets, and one of the techniques mentioned above, and try to embed and extract the watermark; also measure the impact on the utility of the original learning task.

[KZ00]

S. Khanna and F. Zane, "Watermarking maps: hiding information in structured data," in *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, USA, Feb. 2000, pp. 596–605.

[ZLZZ15]

X. Zhao, Q. Liu, H. Zheng, and B. Y. Zhao. (2015, November). Towards graph watermarks. In *Proceedings of the 2015 ACM on Conference on Online Social Networks* (pp. 101-112).

Topic 2.1.2.: Correlation attacks against data fingerprints

Data fingerprinting is a method that deters from unauthorised data sharing and identifies the source of data leakages. Most methods are robust against standard attacks such as random row removal or value flips [1], where the attacker has limited capabilities; however may be vulnerable to a more knowledgeable attacker. Correlation attack is an attack where the attacker uses some prior knowledge about the fingerprinted data, therefore, fingerprinting needs to be adapted to such an attack [2].

Your task is to implement the *correlation attack* and the fingerprinting method that mitigates it as defined in the work by Ji et al.: Towards Robust Fingerprinting of Relational Databases by Mitigating Correlation Attacks, 2022 [2]. The goal is to replicate the results presented in the paper.

- **Correlation attacks**: implement column-wise and row-wise correlation attacks following the pseudocodes in Algorithm 1 and Algorithm 2, respectively. Although [2] shall contain sufficient information, you can additionally use the information from the authors' previous paper that introduces the attacks [3].
- **Mitigation techniques:** Implement Algorithm 3 and Algorithm 4, which describe the defence strategies against the identified correlation attacks. The authors use a vanilla fingerprinting scheme of Li et al. [4] – you may use the existing implementation of this scheme at [5], *Universal scheme*. Note that Li et al. lack description for fingerprinting categorical attributes, therefore different approaches may be taken: the categorical attributes in the implementation [5] are flipped randomly to another value from their domain when chosen for marking, whereas the authors in [2] take an approach of encoding categorical values into integers based on semantics (described in 6.2. and Figure 4. [2]). You can optionally implement this approach for categorical values. Another optional extension: use also the second vanilla scheme from the paper, which is also implemented in [5] as a *Block scheme*, plug in with your mitigation postprocessing and obtain the results related to this scheme (e.g. Table 3).
- **Replicate the results**: Use your implementations to replicate the results from Table 2, Figure 5, Figure 6, Table 4 and the asymmetric attack scenario from Section 6.4.2, i.e. Figure 9 and Figure 10. Critically compare your results with the results from the paper: to what degree do they match; if there are any differences, what is the explanation…

Describe in detail your approach. Describe any insufficient information from the paper for the implementation, if it is the case. Describe your results and the potential differences from the results in the paper. Make sure your submission is *reproducible*.

[1] Šarčević, Tanja, and Rudolf Mayer. "An evaluation on robustness and utility of fingerprinting schemes." *Machine Learning and Knowledge Extraction: International Cross-Domain Conference, CD-MAKE 2019, Canterbury, UK, August 26–29, 2019, Proceedings 3*. Springer International Publishing, 2019.

[2] T. Ji, E. Ayday, E. Yilmaz and P. Li, "Towards Robust Fingerprinting of Relational Databases by Mitigating Correlation Attacks," in *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 4, pp. 2939-2953, 1 July-Aug. 2023.

[3] Ji, Tianxi, et al. "The curse of correlations for robust fingerprinting of relational databases." Proceedings of the 24th International Symposium on Research in Attacks, Intrusions and Defenses. 2021.

[4] Li, Yingjiu, Vipin Swarup, and Sushil Jajodia. "Fingerprinting relational databases: Schemes and specialties." IEEE Transactions on Dependable and Secure Computing 2.1 (2005): 34-45.

[5] https://github.com/sbaresearch/data-fingerprinting

Topic 2.1.3.: Effectiveness and utility cost of attacks against data fingerprints

In this exercise, you can reflect on Exercise 1 and analyse the attacks against data fingerprints in depth. You will receive descriptions of a variety of approaches (approx. 60 approaches) for attacking the fingerprints. Your overall goal is to comprehensively categorise these attacks according to some common properties, evaluate their success and evaluate their utility loss. You shall draw conclusions about the most effective strategies and their cost.

To achieve this goal, the steps may include (but not limited to):

- summarise the attacks and define common properties of the attacks, and categorise the attacks based on these
- Evaluate the effectiveness of the attacks by running the detection process on the datasets. Compare all attacks, and compare according to the categorisation and the properties.
- Define multiple fidelity and utility measures. Evaluate the fidelity/utility of all attacks using these measures. Compare all attacks and compare according to the categorisation and the properties made in the first step. Compare these measures to the ones used by the attacker.
- synthesise the findings from 2) and 3) and find the tradeoffs (i.e.how does the fidelity/utility loss relate to the attack success)
- summarise any other interesting insights when anaylsing the attack

The fingerprinting method evaluated is NCorr-FP; you can find the details in [1] and [2]. You may use the implementation of the detection algorithm from [3].

[1] T. Sarcevic, A. Rauber, R. Mayeri, "NCorr-FP: A Neighbourhood-based Correlation-preserving Fingerprinting Scheme for Intellectual Property Protection of Structured Data," 2025. https://arxiv.org/abs/2505.06379

[2] https://sbaresearch.github.io/data-fingerprinting/#/fingerprinting

[3] https://github.com/sbaresearch/data-fingerprinting/tree/master/NCorrFP (see: Getting Started)

Topic 2.1.4.: Differential Privacy

Differential privacy is introduced by Cynthia Dwork [1] as a set of privacy-preserving mechanisms for publicly sharing information about the dataset by descriptive characteristics rather than sharing the information about the individuals in the dataset. Differential privacy is at first achieved with simple aggregate statistics and descriptive statistics such as mean function, maximum, histograms, etc., and recently, the advances with differentially private machine learning models are achieved, as well.

Differential privacy is achieved by adding noise in the process. Depending on the phase where the noise is added, we differentiate:

- Input perturbation - adding noise to the input before running the algorithm
- Internal perturbation - randomising the internals of the algorithm
- Output perturbation - add noise to the output, after running the algorithm

The goal of this exercise is to compare these approaches for achieving differential privacy, from the privacy aspect and model performance aspect and analyse the tradeoff between privacy and performance. To that end, to achieve internal perturbation, utilise a differential privacy tool that provides differentially private versions of classification and clustering models (for Python: https://github.com/IBM/differential-privacy-library, for R: https://github.com/brubinstein/diffpriv). Familiarise yourself with the differentially private mechanisms used in the chosen tool. Secondly, apply an input perturbation and output perturbation based on the sensitivity method proposed by Dwork et al. in [1]. You may experiment with a model of your choice.

You shall analyse:

1. Input perturbation vs. internal perturbation vs. output perturbation approach on a chosen model; difference between the models' performances, and differences to the non-private model.
2. Trade-off between privacy and model performance for every approach.

Experiments from these papers [2, 3, 4] may serve as an inspiration for the methodology of this exercise.

References:

[1] C. Dwork, F. McSherry, K. Nissim, and A. Smith, “Calibrating Noise to Sensitivity in Private Data Analysis,” in *Theory of Cryptography*, Berlin, Heidelberg, 2006, pp. 265–284. PDF: https://link.springer.com/content/pdf/10.1007/11681878_14.pdf

[2] K. Chaudhuri, C. Monteleoni, and A. D. Sarwate, “Differentially Private Empirical Risk Minimization,” *Journal of Machine Learning Research*, vol. 12, no. Mar, pp. 1069–1109, 2011. PDF: https://arxiv.org/pdf/0912.0071.pdf

[3] K. Fukuchi, Q. K. Tran, and J. Sakuma, “Differentially Private Empirical Risk Minimization with Input Perturbation,” in *Discovery Science*, Cham, 2017, pp. 82–90. PDF: https://arxiv.org/pdf/1710.07425.pdf

[4] Ismat Jarin, Birhanu Eshete. DP-UTIL: Comprehensive Utility Analysis of Differential Privacy in Machine Learning. https://arxiv.org/abs/2112.12998

Further readings - differentially private ML:

[4] O. Sheffet, “Private Approximations of the 2nd-Moment Matrix Using Existing Techniques in Linear Regression,” *arXiv:1507.00056 [cs]*, Nov. 2015.

[5] J. Vaidya, B. Shafiq, A. Basu, and Y. Hong, “Differentially Private Naive Bayes Classification,” in *2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, Nov. 2013, vol. 1, pp. 571–576.

[6] H. Imtiaz and A. D. Sarwate, “Symmetric matrix perturbation for differentially-private principal component analysis,” in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Mar. 2016, pp. 2339–2343.

Topic 2.1.5.: Generation and evaluation of sequential synthetic datasets

This is a rather experimental topic.

The above discussed tools for synthetic data generation have been developed for processing structured relational data. One application scenario for synthetic data is however as well in sequential data, which DNA sequences are one specific instance of.

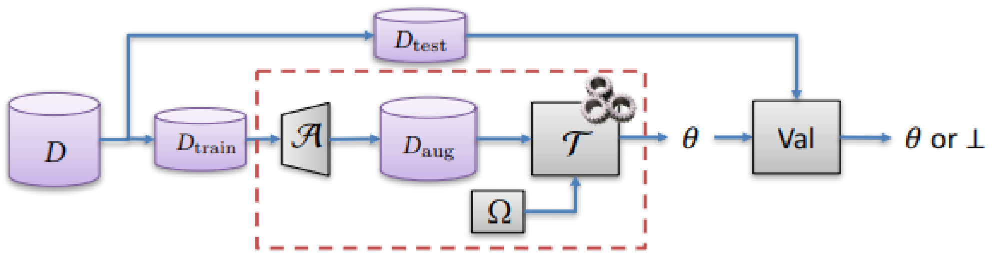
One such method has been described by Pratas et al and uses multiple competing Markov Models. First, a training DNA sequence is partitioned into non-overlapping blocks of fixed size. Each block is handled by a particular Markov Model. In the first phase, the statistics of the training data are loaded into the models. After this deterministic procedure has ended, the models are kept frozen. In a second stochastic phase, the synthetic sequence is then generated according to the learned statistics.

Your task is to recreate the implementation of the specified paper, and evaluate the quality of genetic data when comparing it to real sequences and evaluating it on predictive tasks.

Main paper: D. Pratas, C. A. C. Bastos, A. J. Pinho, A. J. R. Neves, L. M. O. Matos. DNA Synthetic Sequences Generation using Multiple Competing Markov Models. IEEE Statistical Signal Processing Workshop (SSP), pp. 133-136, 2011.

Adversarial Machine Learning

Topic 2.2.1: 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.

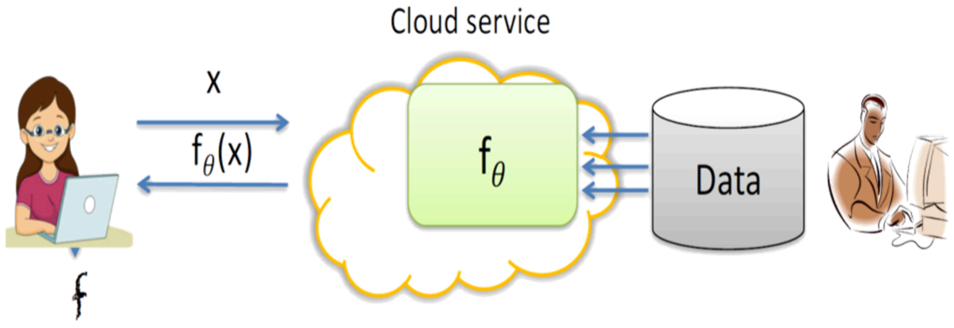
In this task, you shall create experiments either in a white-box or black-box settings similar as described in the paper by *Song et al: Machine Learning Models that Remember Too Much. CCS 2017 (https://www.cs.cornell.edu/~shmat/shmat_ccs17.pdf); code is available either at https://github.com/csong27/ml-model-remember,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

There are several options for this topic:

A. Three examples of the attacks are implemented against the CIFAR dataset (i.e. image data) at https://github.com/andreasiposova/ml-model-remember
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) 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 2.2.2: 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 2.2.3: 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. **The task of this exercise is to work with one watermarking technique**, 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
    - Merrer et al.: Adversarial Frontier Stitching for Remote Neural Network Watermarking (https://arxiv.org/pdf/1711.01894.pdf)
        - Implementation: https://github.com/dunky11/adversarial-frontier-stitching)
    - Adi et al.: Turning Your Weakness into Strength: Watermarking Deep Neural Networks by Backdooring (https://www.usenix.org/system/files/conference/usenixsecurity18/sec18-adi.pdf)
        - Implementation: https://github.com/adiyoss/WatermarkNN)
    - Zhang et al.: Protecting IP of Deep Neural Networks with Watermarking (http://dl.acm.org/citation.cfm?doid=3196494.3196550)
      <u>Either</u> one of the proposed trigger images (unrelated, content or noise).
    - Guo et al.: Watermarking deep neural networks for embedded systems (https://dl.acm.org/doi/10.1145/3240765.3240862)

<u>Either</u>: implement the chosen technique or utilize an open source implementation.

<u>In both cases:</u> 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.

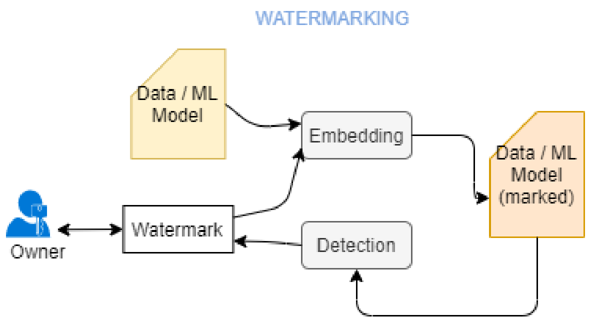
Depending on the choice of either a black-box or white-box watermarking technique, consider the following settings for your task:

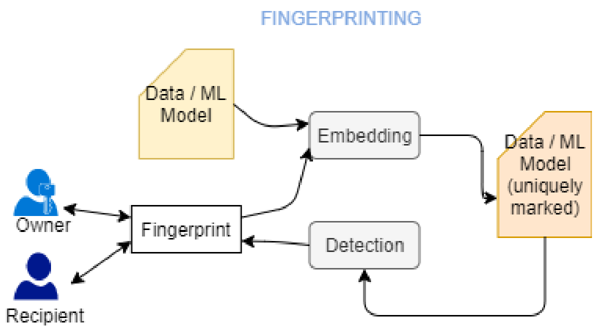
White-Box: implement the method on two different architectures: GoogleNet, SqueezeNet or Densenet, and one dataset (either MNIST, FashionMNIST, CIFAR-10, or ImageNet). Compare the results between those two architectures and conclude where the differences could arise from.

Black-Box: implement the method on one architecture (one of GoogLeNet, SqueezeNet, DenseNet), and on two different datasets (two out of MNIST, FashionMNIST, CIFAR-10, or ImageNet). Compare the results between those two datasets and conclude where the differences could arise from.

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.



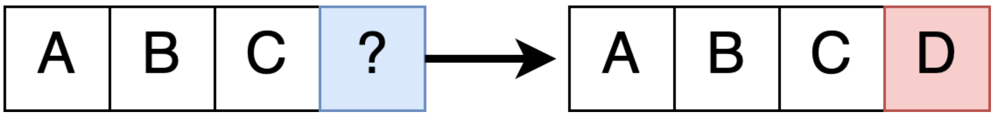


Topic 2.2.4: Attribute Inference (disclosure) from ML Models

(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).

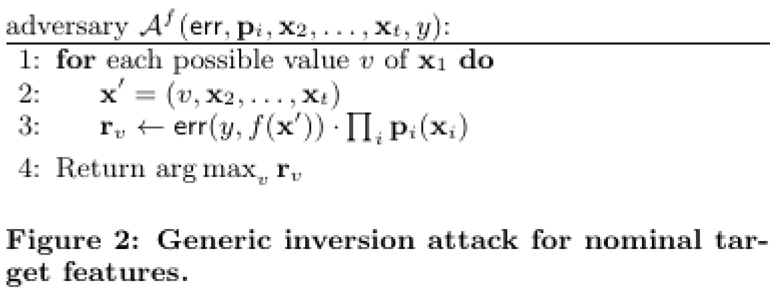
Attribute disclosure is independent of identifying a specific record in a database - inferring additional information about a user from data shared by **other users**. This can be sensitive information about a user such as ethnicity or political affiliation, inferred by mining available data.



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 $\mathcal{A}^f(\text{err}, \mathbf{p}_i, \mathbf{x}_2, \ldots, \mathbf{x}_t, y)$:
1: for each possible value $v$ of $\mathbf{x}_1$ do
2:     $\mathbf{x}' = (v, \mathbf{x}_2, \ldots, \mathbf{x}_t)$
3:     $\mathbf{r}_v \leftarrow \text{err}(y, f(\mathbf{x}')) \cdot \prod_i \mathbf{p}_i(\mathbf{x}_i)$
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 2.2.5: 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 2.2.6: Identifying synthetic data generators

(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).

Topic 2.2.7: Backdoor/poisoning defences

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!*

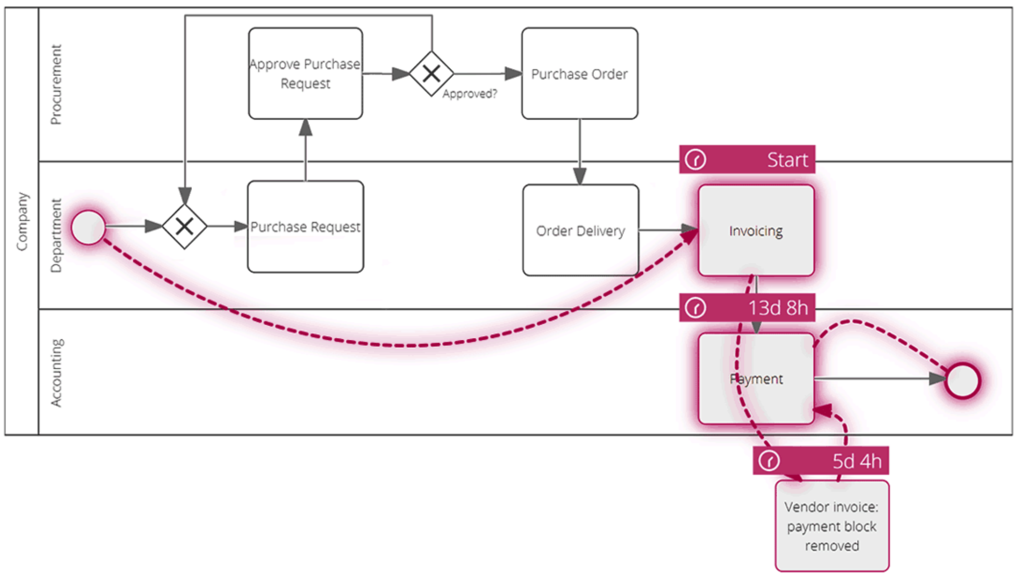
You can re-use any existing code, or implement the methods from scratch by yourself.

Secure / 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 2.3.1: 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 2.3.2: Federated Learning on relational data & shallow learning

In this task for federated learning, you shall work with “traditional” relational data (i.e. not sequential, not image, …). You shall pick two different datasets, which can be any dataset that you have e.g. used in the Machine Learning lecture in Exercise 1 or 2, and employ federated learning with two different types of algorithms on it. You are free to reuse existing implementations (like the ones mentioned above), or you can implement your own approach, but you shall focus on algorithms other than Neural Networks- you can consider e.g. SVMs, Naive Bayes, Decision Trees, …

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 2.3.3: Federated Learning on graph data

In this task for federated learning, you shall work with graph data and graph machine learning models (shallow, or graph neural / convolutional networks). Graph Convolutional Networks [KW+17] learn node representation by aggregating messages from 1-step neighbours using mean pooling. Graph Attention Networks [PCC+18] compute the relation of nodes dynamically using self-attention mechanisms. (short description of the original paper https://petar-v.com/GAT/)

An overview on Federated Graph Learning is given by Zhang et al. [ZSW+21]; they categorize appraoches from the perspective of how graph data are distributed in FL, into four groups:

• Intergraph FL is the most natural derivation of FL, where each sample of clients is of graph data, and the global model performs graph-level tasks. A typical application of inter-graph FL is in the biochemical industry where researchers use GNN to study the graph structure of molecules

• Intra-graph FL where each client owns a part of latent entire graphintra-graph FL and graph-structured FL. It includes:

◦ Horizontal intra-graph FL. Horizontally distributed subgraphs have the same properties, clients share the same feature and label space but different node ID space

◦ Vertical intra-graph FL. Subgraph vertical distribution means that they are parallel and heavily overlap with each other. Under this setting, clients share the same node ID space but different feature and label space

• Graph-structured federated learning refers to the situation where a server uses GNN to aggregate local models based on clients topology.

In your exercise, pick a specific learning scenario, and dataset, and try to transfer centralised learning into federated learning; compare the effectiveness and efficiency.

- Velickovic, Petar, Guillem Cucurull, A. Casanova, Adriana Romero, P. Lio’ and Yoshua Bengio. “Graph Attention Networks.” ArXiv abs/1710.10903 (2018)
- Kipf, Thomas and M. Welling. “Semi-Supervised Classification with Graph Convolutional Networks.” ArXiv abs/1609.02907 (2017).

- Huanding Zhang, Tao Shen, Fei Wu, Mingyang Yin, Hongxia Yang, Chao Wu. “Federated Graph Learning - A Position Paper.” ArXiv abs/2105.11099 (2021)

Topic 2.3.4: Secure Multi-party computation for Image Classification

Utilising e.g. the library mpyc for Python (https://github.com/lschoe/mpyc) and the implementation of a secure computation for a binarised multi-layer perceptron, first try to recreate the results reported in Abspoel et al, “Fast Secure Comparison for Medium-Sized Integers and Its Application in Binarized Neural Networks”, i.e. train a baseline CNN to estimate a potential upper limit of achievable results, and then train the binarized network, as a simplified but still rather performant version, in a secure way. If needed, you can use a subset of the MNIST dataset.

Then, try to perform a similar evaluation on another small dataset, either already available in greyscale, or converted to greyscale, e.g. using (a subset of) the AT&T faces dataset.

Specifically, evaluate the final result in terms of effectiveness, but also consider efficiency aspects, i.e. primarily runtime, but also other resource consumption.

Topic 2.3.5: Secure Multi-party computation for relational Data

In a similar setting as above (but with a machine learning algorithm of your choice), you shall perform secure multi-party computation on relational data. You shall pick two different datasets, which can be any dataset that you have e.g. used in the Machine Learning lecture in Exercise 1 or 2, and simulate a distribution of the data to multiple parties. Evaluate your approach as compared to a centralised setting, similar to as described in the topic above.