`twinify` : A software package for differentially private data release

Joonas Jälkö[1,*], Lukas Prediger[1,*], Antti Honkela[2] and Samuel Kaski[1,3]

[1] Helsinki Institute for Information Technology HIIT,
Department of Computer Science, Aalto University, Finland
[2] Helsinki Institute for Information Technology HIIT,
Department of Computer Science, University of Helsinki, Finland
[3] Department of Computer Science, University of Manchester, United Kingdom

Abstract

Differentially private (DP) data sharing has facilitated releasing data containing sensitive information in a privacy-preserving manner. Many of these data sharing approaches rely on generative models learned under DP, from which a synthetic data set is then sampled. We introduce an easy-to-use software package that allows its user to define a probabilistic generative model for the data sharing task, and automates the learning and generation process based on differentially private probabilistic inference.

1 Introduction

Open access to data is one of the key elements in open science. However, often the most relevant data sets contain sensitive information and thus cannot be released to public. To address this issue, many data sharing solutions founded on the concept of differential privacy have been proposed [1, 2, 3, 4, 5, 6, 3, 7, 8, 9]. Differential privacy (DP) allows learning from the sensitive data with provable privacy guarantees and thus is a favorable privacy notion over alternatives such as k-anonymity.

The main focus in private data sharing research has been in learning generative models under DP from which we can generative synthetic version of the sensitive data set. In this body of work, there has been a proposal of a DP version for many modern generative models such as GANs [10, 11] and VAEs [7, 8]. These models are examples of general purpose generative models and can be applied to any data set. However, there is no free lunch and the performance of any model is heavily dependent on the data. Thus in order to share data most effectively, the data analyst should choose a model that best describes the data and the characteristics the analyst thinks are the most important.

To allow the analyst to easily use DP data sharing with their model, we propose a software package called `twinify`[1]. For modelling and inference purposes `twinify` relies on NumPyro [12], a versatile probabilistic programming framework similar to Pyro [13]. NumPyro uses fast CPU and GPU kernels for execution, which are provided by the JAX framework [14, 15]. Differentially private training routines for NumPyro are introduced by the d3p package [16]

Our software package implements and automates the data sharing procedure to make it accessible for a broad audience. We demonstrate the usage of `twinify` using a COVID-19 related data set, and show that we can build an efficient prediction model from the synthetic data.

[*]These authors contributed equally to this work
[1]https://github.com/DPBayes/twinify/

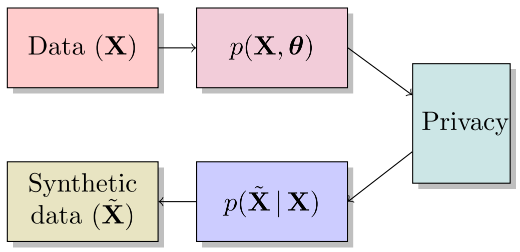
Data ($\mathbf{X}$)
$p(\mathbf{X}, \boldsymbol{\theta})$
Privacy
$p(\tilde{\mathbf{X}} \mid \mathbf{X})$
Synthetic data ($\tilde{\mathbf{X}}$)

Figure 1: Illustration of the data sharing pipeline

2 Privacy preserving data sharing

`twinify` is built around strong privacy guarantees from differential privacy [17, 18], which provides statistical indistinguishability for data subjects.

Definition 2.1 (Differential privacy)**.** A randomised algorithm $\mathcal{M}$ satisfies (ε, δ)-differential privacy if for every neighbouring data sets $D \sim_R D'$, and for all $S \subset \mathrm{im}(\mathcal{M})$, we have

$$\Pr(\mathcal{M}(D) \in S) \leq e^{\varepsilon} \Pr(\mathcal{M}(D') \in S) + \delta. \tag{1}$$

Here $D \sim_R D'$ denotes the add/remove neighbouring relation, where we get D by adding/removing an element of D'.

`twinify` learns the posterior distributions of the parameters $\boldsymbol{\theta}$ of a user specified probabilistic model $p(\mathbf{X}, \boldsymbol{\theta})$ using DP variational inference [19] and samples a synthetic data set from the *posterior predictive distribution* (PPD) of the model given as

$$p(\tilde{\mathbf{X}} \mid \mathbf{X}) = \int_{\boldsymbol{\theta}} p(\tilde{\mathbf{X}} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta} \mid \mathbf{X}) \mathrm{d}\boldsymbol{\theta}. \tag{2}$$

The privacy guarantees of the samples from PPD emerge from the post-processing immunity of DP. For more detailed discussion of the method, see [9].

3 Data Sharing with `twinify`

The `twinify` software package is designed to make privacy-preserving generation of a synthetic twin for any given data set an easy task. The target audience specifically includes users unfamiliar with the details of differential privacy or probabilistic programming. To that end, `twinify` fully automates the differentially private data sharing process depicted in Figure 1. It also performs additional ‘boilerplate’ tasks such as some simple preprocessing steps upon reading the data.

In this section we will go through the basic steps of how to use `twinify` to replicate a data set, namely the specification of the probabilistic model $p(\mathbf{X}, \boldsymbol{\theta})$ that will be fit to the data and the execution of the `twinify` command line program to perform the actual fitting and data sampling.

3.1 Defining the Model

The main task for a `twinify` user is to define the probabilistic model $p(\mathbf{X}, \boldsymbol{\theta})$ that will be fit to the data. `twinify` directly supports specifying models using NumPyro, and imposes only minor constraints on the use of NumPyro’s wide variety of modelling features.

Using `twinify`’s Automatic Modelling To expand the audience to users with less expertise in probabilistic modelling, `twinify` features an automatic modelling capability which builds a mixture model based on a simple text file in which the user assigns distributions to features in the data set, see Fig. 2 for an example of such a configuration file. In this model we assume independence among features within a mixture component, and the dependence structure emerges from the differences between mixture components. The corresponding likelihood is given in Eq. (3):

$$p(\mathbf{x} \mid \boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k \prod_{j=1}^{d} p(\mathbf{x}_j \mid \boldsymbol{\theta}^{(j)}) \tag{3}$$

```
Patient age quantile:   Poisson
Leukocytes:             Normal
Monocytes:              Normal
Patient addmited to regular ward (1=yes, 0=no): Bernoulli
SARS-Cov-2 exam result: Bernoulli
```

Figure 2: Excerpt of the `model.txt` for an example data set containing medical data. Each line contains the exact label of a feature column in the data table and the feature distribution assigned to it.

Distribution	Parameters	Priors
Normal	location μ, scale σ	$\mu \sim \mathcal{N}(0,1)$, $\sigma \sim LogNormal(0,2)$
Bernoulli	logit-probability z	$z \sim \mathcal{N}(0,1)$
Categorical	probabilities $\boldsymbol{\theta}$	$\boldsymbol{\theta} \sim Dirichlet(1,\dots,1)$
Poisson	rate λ	$\lambda \sim Exp(1)$

Table 1: Feature distributions available to the user in `twinify`'s automatic modelling capability, their parameters and the prior distributions applied to them.

We note that the automatic modelling capability of `twinify` is not only addressed at non-expert users: The general-purpose model is also useful when the user has little prior information of feature dependencies, so that a hand-crafted model would not yield much improvement over the automatic one. Moreover, the greater simplicity of the automatic modelling approach can be convenient for data sets with a large number of features, where writing out a full NumPyro model can become quite laborious.

However, if prior knowledge on dependency structure is available to an expert user, a hand-crafted model tailored to the data is likely to result in a better fit. Additionally, note that the automatic modelling of `twinify` was designed to be helpful to the user, not to be a complete solution to the interesting field of automated modelling, and is thus currently limited to the small set of feature distributions shown in Table 1. The table also shows the prior distributions `twinify` uses for the parameters of the feature distributions.

Missing values As an additional consideration for modelling, real data is often incomplete and missing values might occur for a multitude of reasons, for example due to scarcity in measuring resources. Simply removing instances that contain even one missing value might remove vast amounts of data and skew the probabilistic model to not match the original underlying distribution.

`twinify` uses a simple approach to model features with missing values through the following likelihood function:

$$p(x \mid q_{NA}, \theta_x) = \delta_{NA}(x) q_{NA} + \phi(x \mid \theta)(1 - q_{NA}) \mathbb{1}(x \neq NA), \tag{4}$$

where $\phi(x \mid \theta_x)$ is the likelihood for existing values of x, q_{NA} denotes the probability that x is missing and $\delta_N A(x)$ is the Dirac delta function. Similar to other model parameters, we also learn a posterior for q_{NA}.

We note that this model alone treats the values as missing completely at random, which often is too strong an assumption. However, it allows us to model data with missing values without imputation. Also coupled with the mixture model, the above model learns dependencies in the missingness patterns, similar to feature correlations in general.

For users specifying their own model using NumPyro, this missingness model is readily available in `twinify`'s `NAModel` class. In `twinify`'s automatic modelling, the missingness model is automatically applied whenever missing values are detected for a feature.

3.2 Executing `twinify`

`twinify` 's actual model fitting and data synthetisation is made accessible to the user through the `twinify` command line program. The only arguments that must always be passed to the program are the paths to the original data set file, the model specification (either a text file for automatic modelling or a Python file containing a NumPyro model) as well as the synthetic data output. In addition to these, a number of command line arguments to tweak the programs behavior and hyperparameters of the inference are available in order to remain flexible to specific user needs, but sensible default values have been pre-configured.

One important set of hyperparameters affects the number of iterations in the inference algorithm. To allow the user to set these `twinify` adopts the common *epoch* and *minibatch* terminology. Note that this is only for convenience: To benefit from privacy amplification by subsampling, each minibatch is a random sample from the data.

Another set of hyperparameters are the privacy parameters ε and δ. `twinify` automatically adapts the privacy noise in the variational inference algorithm so that the entire training becomes (ε, δ)-DP (as measured by the Fourier Accountant [20]). Additionally, the user can also specify the size of the synthetic data set. Since the model parameter inference is differentially private, an arbitrary amount of samples can be drawn from the posterior preditive distribution. For a full list of all available command line argument to tweak `twinify`'s behavior, please refer to the software manual.[2]

4 Experiments

To demonstrate usefulness of `twinify`, we use it to create a synthetic twin of a medical data set from the Albert Einstein Israelite Hospital in São Paulo, Brazil [21]. This data set has previously been used by Souza *et al.* [22] in a classification task to predict if a patient is infected with the SARS-Cov-2 virus. We aim to replicate this analysis on synthetic data generated by `twinify`.

We rely on `twinify`'s automatic modelling to build a mixture model based on independent feature distributions. Fig. 2 shows an excerpt of the model specification. We use different values for the privacy parameter ε while keeping δ at `twinify`'s default, the inverse of the data set size.

We trained the same classification model as in [22] on the synthetic data output by `twinify` and evaluate the classifiers performance on a test set taken from the original data. Figure 3 shows the accuracy and AUROC attained by the classifier trained on the privacy-preserving synthetic twin data compared to the original data (red) for three choices of ε. We additionally include a measurement obtained by training the classifier on synthetic twin data sampled without differential privacy (green) as a baseline for the expressiveness of the trained probabilistic model unimpeded by privacy constraints.

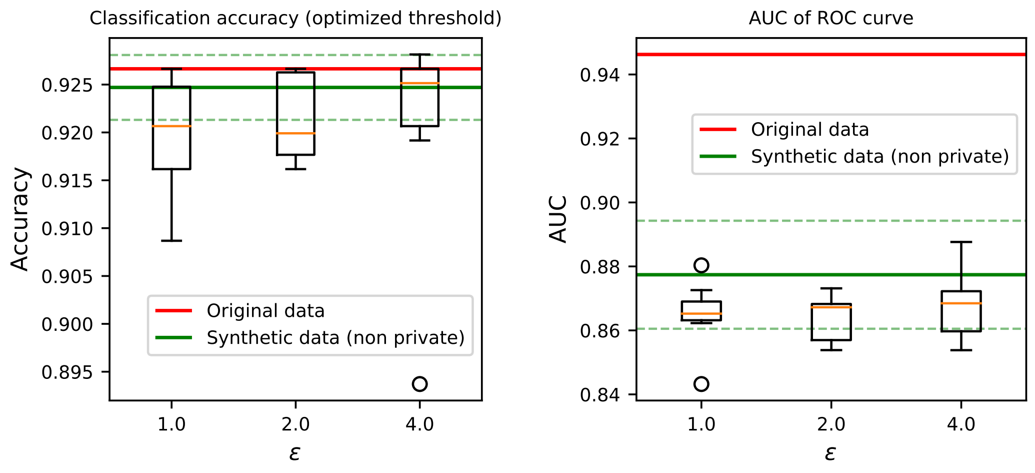

Figure 3: **Left**: Classifier trained with synthetic data achieves accuracy comparable to the classifier trained with original data with reasonable a level of privacy ($\varepsilon = 2$). **Right**: The classifier trained with synthetic data is reasonably well balanced as demonstrated by the AUC of the ROC. Results in both plots are of 10 independent runs of both the private algorithm with three levels of privacy and of a non-private variant.

[2]`https://github.com/DPBayes/twinify/`

References

[1] Avrim Blum, Katrina Ligett, and Aaron Roth. A learning theory approach to non-interactive database privacy. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, pages 609–618, New York, NY, USA, 2008. ACM.

[2] Cynthia Dwork, Moni Naor, Omer Reingold, Guy N. Rothblum, and Salil Vadhan. On the complexity of differentially private data release: Efficient algorithms and hardness results. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, STOC '09, pages 381–390, New York, NY, USA, 2009. ACM.

[3] Yonghui Xiao, Li Xiong, and Chun Yuan. Differentially private data release through multidimensional partitioning. In *Workshop on Secure Data Management*, pages 150–168. Springer, 2010.

[4] Amos Beimel, Shiva Prasad Kasiviswanathan, and Kobbi Nissim. Bounds on the sample complexity for private learning and private data release. In *Theory of Cryptography Conference*, pages 437–454. Springer, 2010.

[5] Rui Chen, Gergely Acs, and Claude Castelluccia. Differentially private sequential data publication via variable-length n-grams. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 638–649. ACM, 2012.

[6] Moritz Hardt, Katrina Ligett, and Frank McSherry. A simple and practical algorithm for differentially private data release. In *Advances in Neural Information Processing Systems*, pages 2339–2347, 2012.

[7] G. Acs, L. Melis, C. Castelluccia, and E. De Cristofaro. Differentially private mixture of generative neural networks. In *2017 IEEE International Conference on Data Mining (ICDM)*, pages 715–720, Nov 2017.

[8] Nazmiye Ceren Abay, Yan Zhou, Murat Kantarcioglu, Bhavani Thuraisingham, and Latanya Sweeney. Privacy preserving synthetic data release using deep learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 510–526. Springer, 2018.

[9] Joonas Jälkö, Eemil Lagerspetz, Jari Haukka, Sasu Tarkoma, Samuel Kaski, and Antti Honkela. Privacy-preserving data sharing via probabilistic modelling. 2019. `https://arxiv.org/pdf/1912.04439.pdf`.

[10] Brett K Beaulieu-Jones, Zhiwei Steven Wu, Chris Williams, Ran Lee, Sanjeev P Bhavnani, James Brian Byrd, and Casey S Greene. Privacy-preserving generative deep neural networks support clinical data sharing. *Circulation: Cardiovascular Quality and Outcomes*, 12(7):e005122, 2019.

[11] Jinsung Yoon, James Jordon, and Mihaela van der Schaar. PATE-GAN: Generating synthetic data with differential privacy guarantees. In *International Conference on Learning Representations (ICLR 2019)*, 2019.

[12] Du Phan, Neeraj Pradhan, and Martin Jankowiak. Composable effects for flexible and accelerated probabilistic programming in NumPyro. *arXiv preprint arXiv:1912.11554*, 2019.

[13] Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D. Goodman. Pyro: Deep Universal Probabilistic Programming. *arXiv preprint arXiv:1810.09538*, 2018.

[14] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, and Skye Wanderman-Milne. JAX: composable transformations of Python+NumPy programs. `https://github.com/google/jax`, 2018.

[15] Roy Frostig, Matthew James Johnson, and Chris Leary. Compiling machine learning programs via high-level tracing. *Systems for Machine Learning*, 2018.

[16] Lukas Prediger, Joonas Jälkö, Antti Honkela, and Samuel Kaski. d3p - differentially private probabilistic programming. `https://github.com/DPBayes/dppp`, 2020.

[17] Cynthia Dwork. Differential privacy. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming*, pages 1–12, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[18] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *TCC 2006*. 2006.

[19] Joonas Jälkö, Onur Dikmen, and Antti Honkela. Differentially private variational inference for non-conjugate models. In *Uncertainty in Artificial Intelligence 2017 Proceedings of the 33rd Conference, UAI 2017*. The Association for Uncertainty in Artificial Intelligence, 2017.

[20] Antti Koskela, Joonas Jälkö, and Antti Honkela. Computing tight differential privacy guarantees using FFT. In *The 23rd International Conference on Artificial Intelligence and Statistics*, 2020.

[21] Hospital Israelita Albert Einstein. Diagnosis of COVID-19 and its clinical spectrum. *Kaggle*, 2020. `https://www.kaggle.com/einsteindata4u/covid19/`.

[22] Tharsis Souza, Gustavo Wenzel Sainatto, and Heli S. P. Souza. COVID-19 machine learning-based rapid diagnosis from common laboratory tests. *Towards Data Science*, 2020. `https://towardsdatascience.com/covid-19-machine-learning-based-rapid-diagnosis-from-common-laboratory-tests-afafa9178372`, Accessed: 2020-06-15.