**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, 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.

---

\(^1\)These authors contributed equally to this work

\(^1\)https://github.com/DPBayes/twinify/

Preprint. Under review.
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 $M$ satisfies $(\varepsilon, \delta)$-differential privacy if for every neighbouring data sets $D \sim_R D'$, and for all $S \subset \text{im}(M)$, we have

$$\Pr(M(D) \in S) \leq e^\varepsilon \Pr(M(D') \in S) + \delta.$$  

(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 $\theta$ of a user specified probabilistic model $p(X, \theta)$ using DP variational inference [19] and samples a synthetic data set from the posterior predictive distribution (PPD) of the model given as

$$p(\tilde{X} \mid X) = \int_{\theta} p(\tilde{X} \mid \theta)p(\theta \mid X)d\theta.$$  

(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 $p(X, \theta)$ 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 $p(X, \theta)$ 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):

$$p(x \mid \theta) = \sum_{k=1}^{K} \pi_k \prod_{j=1}^{d} p(x_j \mid \theta^{(j)})$$  

(3)
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:

\[
p(x \mid q_{NA}, \theta_x) = \delta_{NA}(x)q_{NA} + \phi(x \mid \theta_x)(1 - q_{NA})\mathbb{1}(x \neq NA),
\]

where \(\phi(x \mid \theta_x)\) is the likelihood for existing values of \(x\), \(q_{NA}\) denotes the probability that \(x\) is missing and \(\delta_{NA}(x)\) 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 program’s 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 $\varepsilon$ and $\delta$. twinify automatically adapts the privacy noise in the variational inference algorithm so that the entire training becomes $(\varepsilon, \delta)$-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 predictive distribution. For a full list of all available command line arguments to tweak twinify’s behavior, please refer to the software manual.

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 $\varepsilon$ while keeping $\delta$ 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 $\varepsilon$. 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.

[https://github.com/DPBayes/twinify/](https://github.com/DPBayes/twinify/)
References


