Privacy

A high percentage of modern data is collected from devices enabling human-machine interactions. For instance, robot vacuum cleaners often have a built-in camera to navigate where to clean, but the camera can also capture scenes of people in the room. Similarly, user-facing artificial intelligence agents such as virtual assistants collect, store, and stream potentially privacy-sensitive data. These data might be used to fit machine learning models to solve certain statistical tasks. Many recent papers indicate that fitted machine learning models can expose sensitive information from the dataset they were trained on.

For developing privacy-preserving machine learning methods, the first question we need to answer is how to define privacy and how to measure it. While there are various privacy notions developed in the field, *differential privacy* is currently the gold standard for privacy, due to its rigorous privacy guarantees. A randomized algorithm $\mathcal{M}(\mathbf{X})$ that takes a dataset $\mathbf{X}$ and outputs a quantity (scalar or vector) is said to be (ϵ, δ)-differentially private if the probabilities of the algorithm outputting the same quantity $\mathcal{S}$ given two datasets $\mathbf{X}$ and $\mathbf{X}'$ are related as

$$Pr(\mathcal{M}(\mathbf{X}) \in \mathcal{S}) \leq \exp(\epsilon) Pr(\mathcal{M}(\mathbf{X}') \in \mathcal{S}) + \delta \tag{1.1}$$

for all measurable subsets $\mathcal{S}$ of the range of $\mathcal{M}$ and for all neighbouring datasets $\mathbf{X}$, $\mathbf{X}'$ that differ by a single entry (either by excluding that entry or replacing it with a new entry). Here, an entry usually corresponds to a single individual's private data. Intuitively, the definition states that the probability of any event does not change much when a single individual's data is modified, thereby limiting the amount of information that the algorithm reveals about any one individual.

Equipped with such a notion of privacy, the development of differentially private data analysis methods requires first to decide which quantity we want to guard. Depending on whether we want to share data, statistics, or model parameters, each will require developing different techniques. Below, we look into two different types of sharing, namely, data sharing and parameter sharing.

Model sharing is probably the most popular way to achieve privacy in the current differential privacy literature. Model sharing often focuses on guarding model parameters by adding noise to them before releasing them. Generally, there are two ways of achieving private models. First, one could add noise to the objective function ("objective perturbation") such that the resulting estimates guarantee some level of privacy. Second, one could add noise to the output of an optimization routine ("output perturbation"). Often, objective perturbation techniques end up adding less noise than output perturbation techniques. However, to be able to analyze the relationship between the amount of noise that needs to be added to an objective function and the guaranteed privacy level of a corresponding estimate after optimizing the perturbed objective, it is unavoidable to make some stringent assumptions (e.g., strong convexity). Many learning problems in machine learning have non-convex objective functions, which limits the usefulness of the existing objective perturbation techniques. We are currently working on addressing the problem with non-convex objective functions, especially in the context of deep learning [107].

Data sharing requires adding noise to the dataset itself before releasing it. Most of the existing data sharing frameworks are designed for particular data types (e.g., count data, low-dimensional data) or particular purposes (e.g., decision tree algorithms) only. We thus need algorithms to add noise in a way that is truthful to the statistical properties of the raw data while being independent of downstream tasks, for a better utility in various statistical analyses. At ICML 2018, we have published a kernel method for this task, which relies upon kernel mean embedding on datasets in reproducing kernel Hilbert spaces, which (for suitable kernels) retains all information about a distribution [155].

More information: https://ei.is.mpg.de/project/privacy