From Zero to Hero: Detecting Leaked Data through Synthetic Data Injection and Model Querying

Biao Wu
National University of Singapore
wubiao@comp.nus.edu.sg

Qiang Huang*
National University of Singapore
huangq@comp.nus.edu.sg

Anthony K. H. Tung
National University of Singapore
atung@comp.nus.edu.sg

ABSTRACT

Safeguarding the Intellectual Property (IP) of data has become critically important as machine learning applications continue to proliferate, and their success heavily relies on the quality of training data. While various mechanisms exist to secure data during storage, transmission, and consumption, fewer studies have been developed to detect whether they are already leaked for model training without authorization. This issue is particularly challenging due to the absence of information and control over the training process conducted by potential attackers.

In this paper, we concentrate on the domain of tabular data and introduce a novel methodology, Local Distribution Shifting Synthesis (LDSS), to detect leaked data that are used to train classification models. The core concept behind LDSS involves injecting a small volume of synthetic data–characterized by local shifts in class distribution–into the owner's dataset. This enables the effective identification of models trained on leaked data through model querying alone, as the synthetic data injection results in a pronounced disparity in the predictions of models trained on leaked and modified datasets. LDSS is *model-oblivious* and hence compatible with a diverse range of classification models. We have conducted extensive experiments on seven types of classification models across five real-world datasets. The comprehensive results affirm the reliability, robustness, fidelity, security, and efficiency of LDSS. Extending LDSS to regression tasks further highlights its versatility and efficacy compared with baseline methods.

PVLDB Reference Format:
Biao Wu, Qiang Huang, and Anthony K. H. Tung. From Zero to Hero: Detecting Leaked Data through Synthetic Data Injection and Model Querying. PVLDB, 17(8): 1898 - 1910, 2024.
doi:10.14778/3659437.3659446

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/e0001428/local-distribution-shifting-synthesis/.

1 INTRODUCTION

It is of paramount importance to protect data through strict enforcement of authorized access. Leaked data can have severe repercussions, ranging from exposure of private information to reputational harm. More critically, it can lead to loss of competitive business advantages [10], violations of legal rights like privacy[40], and breaches of national security [2]. Substantial efforts are directed towards bolstering data security, encompassing secure storage, safe data transfer, and watermarking to trace leaked data, thereby protecting the owner's Intellectual Property (IP) via legal channels. Such watermarking techniques are typically employed in images and videos by embedding either visible or concealed information in the pixels to denote ownership [4, 11, 16, 33].

With the evolution of machine learning and artificial intelligence techniques, new and nuanced threats surface beyond conventional concerns about data theft and publication. Notably, malicious entities who gain access to leaked datasets often utilize them to train models, seeking a competitive edge. For instance, such training models can be exploited for commercial endeavors or to derive insights from datasets that were supposed to remain inaccessible. This emerging threat remains underexplored, and traditional digital watermarking methods might fall short in ensuring traceability, especially in a black-box setting where one can only access a model through a query interface [36].

In this paper, we attempt to address the emerging threat of detecting leaked data used for model training. We introduce a novel method, Local Distribution Shifting Synthesis (LDSS), designed to achieve exemplary detection precision on such leakage. Our study focuses on tabular datasets. For image and video datasets, existing work such as training models on watermarked or blended images [55] and label flipped images[1, 38] remain effective. This is because researchers often employ Deep Neural Networks (DNNs) on such datasets, where subtle watermarks can be seamlessly integrated without compromising model efficacy yet still be retained for identification purposes. Nonetheless, a substantial number of sensitive datasets, like those pertaining to patients or financial information, are stored in tabular format within relational databases. They typically possess fewer features compared to images with thousands of pixels. Without meticulous design, embedding a "watermark" into a limited number of columns could severely compromise model accuracy. Moreover, such alterations might go unnoticed by the machine learning model, especially when simpler models like Decision Tree, Support Vector Machine, and Random Forest are employed for these types of tabular datasets. Specifically, we concentrate on the classification task, as many functions in the financial and healthcare sectors hinge on classification. Examples include determining a patient's susceptibility to specific diseases based on screening metrics [18, 31] as well as assessing if an applicant qualifies for a loan based on their data [3, 39]. Additionally, we assume that only black-box access to the target model is available. This is because unauthorized parties who engage in data theft typically do not disclose their model details, providing only a query interface instead.

*Qiang Huang is the corresponding author.

Proceedings of the VLDB Endowment, Vol. 17, No. 8 ISSN 2150-8097.
doi:10.14778/3659437.3659446

Detecting leaked data utilized for model training solely through model querying is non-trivial. At first glance, employing the membership inference attack [41] appears to be a straightforward yet effective solution. However, unauthorized entities can counteract this by applying model generalization techniques [45]. They might strategically prompt the model to produce incorrect predictions for queries that precisely match instances in their training data, thereby evading detection. Furthermore, this solution sometimes yields a high false-positive rate. This is especially true if the allegedly leaked data originates from a commonly accessed source. In such cases, the third party might coincidentally possess a similar dataset, which could have closely related or even identical instances.

There exist various watermarking and backdoor mechanisms [5, 26, 50] that are designed to embed information into target models. Subsequently, this information can be extracted with high precision to assert ownership. Nevertheless, most of these methods predominantly target the image classification task using DNN models [14, 32, 53, 54]. This preference arises from the rich parameters inherent in DNN models, which facilitate the embedding of additional information without compromising model performance. Their techniques, however, are not suitable for the problem of leaked data detection we explore in this paper. Tabular data, frequently used in fields like healthcare and finance where explainability is paramount, often employ non-DNN models. Moreover, dataset owners lack access to, control over, or even knowledge about the model training process in leaked data scenarios. Therefore, most extant methods in model watermarking [12, 30] and data poisoning or model backdoor [9, 37] fall short as they typically target a specific model type and necessitate access or control of the training process to guarantee optimal and effective embedding.

To overcome the above challenges, LDSS adopts a *model-oblivious* mechanism, ensuring robust leaked data detection across different model types and training procedures. The conceptual foundation of LDSS is inspired by the application of active learning and backdoor attacks used to embed watermarks into DNN models [1]. Rather than merely employing random label flipping, LDSS adopts a more nuanced synthetic data injection strategy. It injects synthetic samples into empty local spaces unoccupied by the original dataset, thereby altering local class distributions. This technique obviates modifying or removing existing instances, preserving the original distribution within the occupied local spaces. Consequently, the classification model trained with these injected samples remains at a *similar level of accuracy*; meanwhile, the synthetic injections produce predictions that are *substantially different* in the affected local spaces compared to models trained without such injections. This ensures a low rate of false positives, even for models trained on original datasets or datasets derived from similar populations.

Once the owner modifies the dataset, any subsequent data acquired by an attacker will be this modified version. The leaked data detection is executed by model querying, i.e., querying the suspect model using a trigger set of synthetic samples drawn from the same local space as the previously injected samples. To rigorously assess the performance of LDSS, we employed a comprehensive set of evaluation metrics across seven types of classification models. While these models share common objectives, their underlying principles and distinctive characteristics offer a holistic perspective on LDSS's performance under diverse conditions. Our extensive experimental results, spanning five real-world datasets, showcase the superior performance of LDSS compared to two established baselines. Importantly, these outcomes underscore its reliability, robustness, fidelity, security, and efficiency. Additionally, we extend LDSS to regression tasks, conducting experiments on two real-world datasets and assessing its performance across seven regression models. This extension showcases LDSS's versatility and leading efficacy across different tasks.

Table 1: List of frequently used notations.

Symbol	Description
$\mathcal{D}_{orig}$	Owner's original dataset without modifications
n	The number of samples in $\mathcal{D}_{orig}$
d	The number of features in $\mathcal{D}_{orig}$
$\mathcal{D}_{inj}$	Synthesized samples to be injected into $\mathcal{D}_{orig}$
$\mathcal{D}_{mod}$	Owner's modified dataset, i.e., $\mathcal{D}_{mod} = \mathcal{D}_{orig} \cup \mathcal{D}_{inj}$
$\mathcal{D}_{trig}$	Synthesized dataset similar to $\mathcal{D}_{inj}$, which is used to query the target model for ownership detection
k	An integer value for numerical feature discretization
m	The number of pivots for data transformation
ρ	The injection ratio, i.e., $\rho = \lvert\mathcal{D}_{inj}\rvert / \lvert\mathcal{D}_{orig}\rvert$
g	The number of selected empty balls
h	The number of synthetic samples per ball

Organization. The rest of this paper is organized as follows. The problem of leaked data detection is formulated in Section 2. We introduce the LDSS method in Section 3. The experimental results and analyses are presented in Section 4. Section 5 reviews related work. Finally, we conclude this work in Section 6.

2 PROBLEM FORMULATION

In this section, we begin by formally defining the problem of leaked data detection. Following that, we draw comparisons to the problem of model backdooring through data poisoning. Lastly, we outline the criteria for an ideal solution for leaked data detection. Table 1 summarizes the frequently used notations in this paper.

Definition of Leaked Data Detection. In this work, we investigate the problem of leaked data detection, which includes two parties: dataset owners and attackers. The owner has a tabular dataset denoted by $\mathcal{D}_{orig}$, which is structured as a table comprising n rows and d feature columns, along with an additional column designated for class labels. Consequently, each row in this table represents a distinct sample instance. Attackers are assumed to have circumvented security measures and gained unauthorized access to the owner's dataset. They then train a classification model using the designated label column and deploy it as a service. This model can be developed through any data processing techniques, training procedures, and various model types. The objective is to develop a reliable method that allows dataset owners to ascertain, in a black-box setting, whether a given model has been trained using their dataset. We also explore possible adaptive attacks, assuming that attackers know the proposed method and possess certain information, such as the class distribution and feature ranges from which $\mathcal{D}_{orig}$ is drawn. However, we assume that they lack more detailed insights, such as the distribution of features by class or local class distribution at any subspace. This problem is challenging as

dataset owners lack access to the internal parameters of the target model, and their interaction is restricted to submitting queries and receiving corresponding predicted labels.

Comparisons to Model Backdooring through Data Poisoning. This problem closely resembles the objective of model backdooring through data poisoning [1, 55]. We aim to transform the owner's original dataset, $\mathcal{D}_{orig}$, into a modified version, $\mathcal{D}_{mod}$, ensuring that a model trained on either dataset will exhibit discernible differences in its predictions. Given that we operate under a black-box verification setting, our means of differentiation is to compare the predicted labels of designated trigger samples. Nevertheless, there are three pivotal differences between our problem and the problem of model backdooring via data poisoning:

(1) Our primary goal is to empower dataset owners to verify and claim ownership of their dataset through a model trained on it. Unlike strategies that might degrade model performance or embed malicious backdoors, the proposed method should have a minimal impact on both the dataset and the models trained on the modified version.
(2) Distinct from model backdooring, this research investigates a defense scenario where defenders do not require any knowledge or access to the attacker's target model. Defenders can only interact with the model through model querying once the attacker has completed training. Thus, the proposed solution should not make assumptions or modify the model's architecture, type, or training procedures.
(3) In the context of model backdooring through data poisoning, the poisoned dataset is typically inaccessible to parties other than the attackers, making it extremely difficult to detect or locate the backdoor in the trained model. However, for the problem at hand, we assume that attackers have complete access to the modified dataset. This necessitates that any modifications to the dataset should be as discreet as possible to avoid detection and reversal by adversaries.

These key differences render direct application of existing model backdooring and data poisoning solutions problematic, including random label flipping [1, 55] and training-time model manipulations like manipulating decision boundaries or parameter distributions [21]. The former is susceptible to detection as outliers, while the latter assumes complete control of the training procedure.

Criteria for An Ideal Solution. To address the first challenge, the proposed solution should abstain from altering samples in $\mathcal{D}_{orig}$. It must be crafted to minimize the impact on model accuracy for unseen samples drawn from the same population as $\mathcal{D}_{mod}$. To overcome the second and third challenges, the proposed method should be agnostic to the attackers' model training protocols. Furthermore, the injected samples should be meticulously designed to closely mirror the original dataset, thereby enhancing stealth and reducing the risk of detection. Next, we introduce a novel approach, LDSS, which fulfills these stipulated criteria.

3 LDSS: THE METHODOLOGY

3.1 Overview

LDSS allows data owners to inject a series of synthetic samples, denoted as $\mathcal{D}_{inj}$, into their original dataset, $\mathcal{D}_{orig}$. This strategy is designed to deliberately shift the local class distribution. Once this injection is made, it is assumed that attackers would only have access to the modified dataset $\mathcal{D}_{mod}$. For samples similar to the injected ones, any target model trained on $\mathcal{D}_{mod}$ is inclined to yield predictions distinct from model trained on $\mathcal{D}_{orig}$. Thus, to discern whether datasets are leaked, data owners can generate a trigger set $\mathcal{D}_{trig}$ from similar distributions as $\mathcal{D}_{inj}$ for model querying.

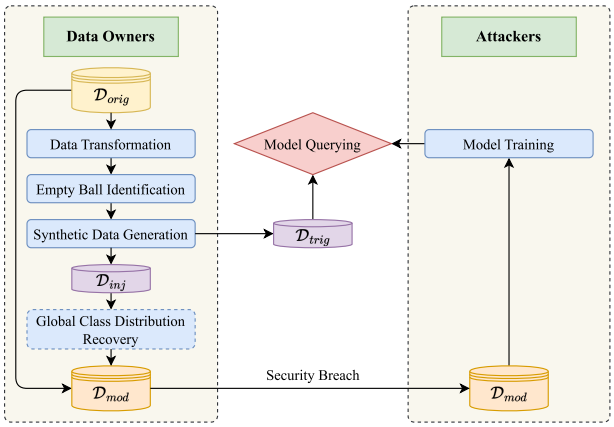


Figure 1: Overall flow of LDSS.

Figure 1 illustrates the overall flow of LDSS. To maintain the model's efficacy on genuine data resembling the distribution of $\mathcal{D}_{orig}$, LDSS needs to identify large empty balls that are devoid of any samples from $\mathcal{D}_{orig}$ before synthetic data generation. Nevertheless, identifying large empty spaces in tabular datasets is challenging due to their mix of numerical and categorical features. To address this, we undertake a data transformation process, converting the dataset into an entirely numerical format. When generating injection samples $\mathcal{D}_{inj}$, LDSS synthesize samples within a target class with the most significant frequency disparities relative to the prevailing local class. This strategy can shift the local class distribution. However, it might also drastically alter the global class distribution and become discernible to attackers. To remedy this issue, we could perform an additional step to recover the global class distribution. Details are presented in the following subsections.

3.2 Data Transformation

Challenges. To convert the tabular dataset to a wholly numerical format, a prevalent strategy is to apply one-hot encoding to categorical columns [7]. Nevertheless, this strategy can result in extremely high dimensionality, especially when there are a lot of categorical features, each with a multitude of unique values. Not only does computing an empty ball become considerably slower, but such space also becomes very sparse. Thus, the empty ball no longer provides a good locality guarantee. Dimension reduction methods such as Principal Component Analysis (PCA) [15, 29, 44] are unsuitable here since the reduced space should be transformed back to the original space, which contains one-hot encoded features.

To address the aforementioned challenges, we first convert the tabular dataset to an entirely categorical format. This allows us to employ a distance metric, e.g., Jaccard distance, to gauge the proximity between any pair of data. Subsequently, we determine m representative pivots and utilize the Jaccard distance to these

m pivots as the numerical representation for each data. The data transformation process encompasses three steps: numerical feature discretization, feature mapping, and numerical representation using pivots. Each of these steps is elucidated as follows.

Step 1: Numerical Feature Discretization. LDSS first discretizes each numerical feature into $(k+1)$ bins using one-dimensional k-means clustering. Each bin is assigned a nominal value ranging from 0 to k, in ascending order of their center values. Here, the value of k is chosen to accurately represent the distribution of the feature's values. To avoid inefficiencies in subsequent steps, k is typically set to a moderate value, such as 5 or 10.

Step 2: Feature Mapping. After the numerical feature discretization, we map each data $\boldsymbol{x} = (x_1, \cdots, x_d)$ into a set $\tilde{\boldsymbol{x}}$ that contains $(k \cdot d)$ elements, with each feature being represented by k elements. This mapping is guided by Equations 1 and 2 as follows:

$$\tilde{\boldsymbol{x}} = \bigcup_{i=1}^{d} T(i, x_i), \tag{1}$$

where

$$T(i, x_i) = \begin{cases} \{(i, x_i, j) \mid j \in [1, k]\}, & i \text{ is categorical;} \\ \{(i, x_i + j) \mid j \in [1, k]\}, & i \text{ is numerical.} \end{cases} \tag{2}$$

For each categorical feature, identical values yield the same set of k elements, ensuring that distinct values always produce k distinct elements. For any two numerical values, the number of identical elements among the k elements reflects the closeness of their numerical values. For example, if their categories after discretization are v_1 and v_2, then exactly $k - |v_1 - v_2|$ elements will be the same after the feature mapping. The smaller the difference between the original values, the more elements they share after feature mapping. This mapping maintains the degree of similarity between two samples, and such similarity is scaled for numerical features to capture their distributions better.

Step 3: Numerical Representation using Pivots. Finally, we determine m sets as pivots and transform each set $\tilde{\boldsymbol{x}}$ into an m-dimensional vector $\hat{\boldsymbol{x}} = (\hat{x}_1, \cdots, \hat{x}_m) \in [0, 1]^m$, where each coordinate $\hat{x}_i$ is represented by the Jaccard distance between $\tilde{\boldsymbol{x}}$ and i-th pivot. For any two sets $\tilde{\boldsymbol{x}}$ and $\tilde{\boldsymbol{y}}$, the Jaccard distance is defined as:

$$J(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}}) = 1 - \frac{|\tilde{\boldsymbol{x}} \cap \tilde{\boldsymbol{y}}|}{|\tilde{\boldsymbol{x}} \cup \tilde{\boldsymbol{y}}|}. \tag{3}$$

When selecting a small value for m, such as 5 or 10, the resulting space has a low dimension, facilitating the efficient identification of empty balls. Moreover, as the Jaccard distance is a distance metric that can effectively gauge the similarity between samples and pivots, samples in close proximity within the transformed space remain similar in the original feature space.

The next challenge is how to determine the pivots to preserve the proximity of the dataset. An initial approach might be to randomly select pivots. In this method, pivots are drawn uniformly at random from $\mathcal{D}_{orig}$. However, this can lead to the selection of suboptimal pivots, especially if certain features in the randomly selected pivots are uncommon. As such, the Jaccard distance between the samples and such pivots can be consistently large. A more refined strategy is to construct pivots based on value frequency. Specifically, for each feature, we identify the top m frequent values. These values are then assigned to the pivots in a *round-robin* manner. If a feature has fewer than m distinct values, we repeatedly use those values until every pivot has its set of values. This strategy sidesteps the emphasis on infrequent values and also prevents the repetition of unique values, especially when a particular value is overwhelmingly common in specific features. Additionally, this strategy often results in more orthogonal pivots, especially when every feature has at least m unique values. In practice, m can conveniently be set equal to k, which is the maximum number of discretized values of each numerical feature so that pivots include all the discretized values of numerical features.

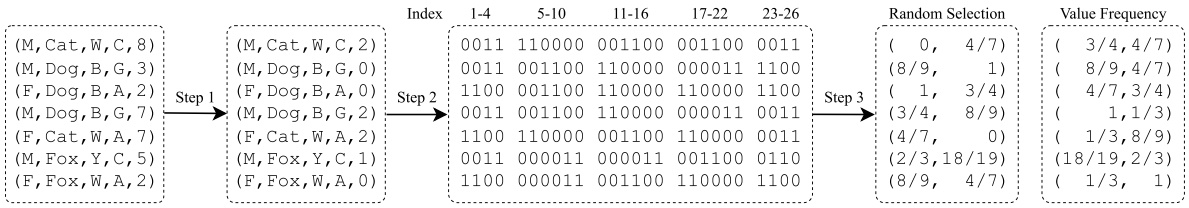
Example 1: Suppose a small pet dataset contains seven samples, as illustrated at the beginning of Figure 2(a). This dataset contains $d = 5$ feature columns, namely gender, species, code of color, code of the country of origin, and age. The first four features are categorical, while the last one is numerical. Let $m = k = 2$.

In step 1, the last numerical feature is discretized into $k + 1 = 3$ bins, with values 2 and 3 in bin 0, 5 in bin 1, and 7 and 8 in bin 2. For example, the first sample becomes $(M, Cat, W, C, 2)$. In step 2, each sample is mapped to $kd = 10$ elements. Taking the first sample as an example, it maps to the set: $\{(1, M, 1), (1, M, 2), (2, Cat, 1), (2, Cat, 2), (3, W, 1), (3, W, 2), (4, C, 1), (4, C, 2), (5, 3), (5, 4)\}$. To streamline our explanation, elements after feature mapping are uniquely indexed, as shown in Table 2. Furthermore, in Figure 2(a), we employ one-hot encoding to transform sets of elements into binary strings. Thus, the first sample is represented as 0011 110000 001100 001100 0011. In step 3, we select two pivots ($k = 2$) and compute the Jaccard distance between samples and pivots. For the approach using random selection, e.g., selecting the first and the fifth samples as pivots, the resulting representation is shown in the left column of Figure 2(a) following Step 3. For the approach based on value frequency, e.g., constructing a pair of pivots $(F, Cat, W, A, 2)$ and $(M, Dog, B, C, 8)$–with the last feature discretized to 0 and 2, respectively–the resulting representation is illustrated in the right column of Figure 2(a) after Step 3.

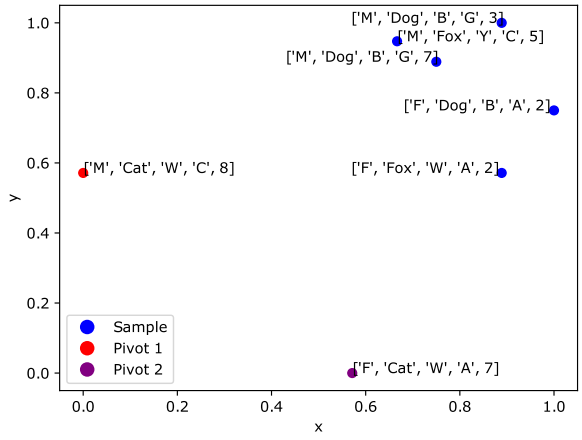
In Figure 2(b) and 2(c), we depict the sample representations of both approaches using scatter plots. These figures highlight that pivots built on value frequency more effectively preserve the relative similarities among samples post-transformation compared to those by random selection. This is because the random pivot

Table 2: Index mapping of transformed elements.

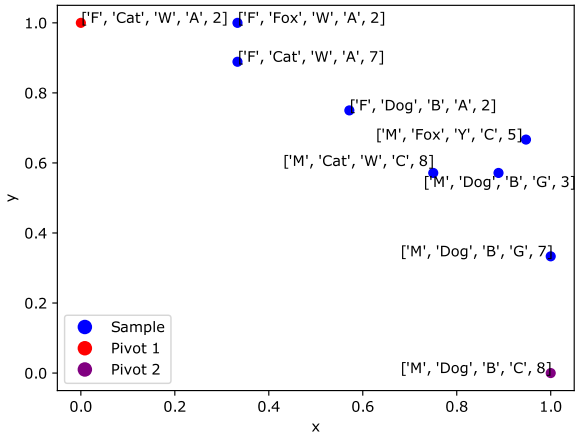
Index	Element	Index	Element
1	(1, F, 1)	14	(3, W, 2)
2	(1, F, 2)	15	(3, Y, 1)
3	(1, M, 1)	16	(3, Y, 2)
4	(1, M, 2)	17	(4, A, 1)
5	(2, Cat, 1)	18	(4, A, 2)
6	(2, Cat, 2)	19	(4, C, 1)
7	(2, Dog, 1)	20	(4, C, 2)
8	(2, Dog, 2)	21	(4, G, 1)
9	(2, Fox, 1)	22	(4, G, 2)
10	(2, Fox, 2)	23	(5, 1)
11	(3, B, 1)	24	(5, 2)
12	(3, B, 2)	25	(5, 3)
13	(3, W, 1)	26	(5, 4)



(a) The pipeline of data transformation.



(b) Sample representations by random selection.



(c) Sample representations by value frequency.

Figure 2: Data transformation results using different pivot selection methods. Note that in Figure 2(b), both pivots come from original samples, while in Figure 2(c), both pivots are constructed based on value frequency. In Figure 2(a), we present the final results using random selection and value frequency, which map to Figures 2(b) and 2(c), respectively.

selection might incorporate rare or duplicate values for specific features, causing many unique values to become indistinguishable. On the other hand, pivots derived based on value frequency tend to be more orthogonal, thus better maintaining relative similarity. △

3.3 Empty Ball Identification

The next step is identifying empty balls in the transformed hypercube $[0,1]^m$. Our objective is to locate those empty balls that are both sufficiently large and close to the transformed dataset $\widehat{\mathcal{D}}_{orig}$. Balls located too far from this dataset are considered outliers, making them unsuitable for synthetic data injection. Before delving into our strategy to identify these empty balls, we first define an empty ball for numerical features.

Definition of the Empty Ball. Let $\hat{\boldsymbol{c}} = (\hat{c}_1, \cdots, \hat{c}_m)$ represent the center of an empty ball in the m-dimensional transformed space. To define the radius of this empty ball, we consider the prevalent Euclidean distance between any two points $\hat{\boldsymbol{x}} = (\hat{x}_1, \cdots, \hat{x}_m)$ and $\hat{\boldsymbol{y}} = (\hat{y}_1, \cdots, \hat{y}_m)$, i.e., $\|\hat{\boldsymbol{x}} - \hat{\boldsymbol{y}}\| = \sqrt{\sum_{i=1}^{m}(\hat{x}_i - \hat{y}_i)^2}$. With this distance, the radius r of the empty ball centered at $\hat{\boldsymbol{c}}$ is defined below:

$$r = \min_{\hat{\boldsymbol{x}} \in \widehat{\mathcal{D}}_{orig}} \|\hat{\boldsymbol{x}} - \hat{\boldsymbol{c}}\|. \tag{4}$$

Empty Ball Identification. To prevent the over-clustering of samples generated within the empty balls, which could allow attackers to easily identify injected samples, we prioritize the balls with larger radii over those with smaller ones. Several heuristics, such as the Voronoi diagram and the Evolutionary Algorithm, are available for finding the largest empty ball, as discussed by Lee et al. [22]. However, our goal is merely to identify large empty balls. For enhanced efficiency, we adopt Simulated Annealing (SA) coupled with constrained updates. The procedure is detailed in Algorithm 1.

The algorithm initiates by randomly selecting a batch of G samples (500, in our experiments) from $\widehat{\mathcal{D}}_{orig}$ as seeds (potential centers for empty balls) (Line 1). Then, it continues to replace these seeds with perturbed versions that have a greater radius (Lines 4–8). As iterations progress, the perturbation scale is reduced, governed by the temperature parameter t (Line 9). Given an arbitrary center $\hat{\boldsymbol{c}} = (\hat{c}_1, \cdots, \hat{c}_d)$, the function $perturb(\hat{\boldsymbol{c}}, t)$ adds noise that is scaled by t and the standard deviation std_i of each dimension i over all samples, i.e., $\hat{c}'_i = \hat{c}_i + t \cdot uniform(-std_i, std_i)$. The function $r(\hat{\boldsymbol{c}})$ is defined by Equation 4, computing the radius of the empty ball centered at $\hat{\boldsymbol{c}}$. In each iteration, ball centers are updated by the perturbed ones that have a larger radius (Line 7). To ensure that the empty balls remain in proximity to $\widehat{\mathcal{D}}_{orig}$ and avoid positioning synthesized samples as potential outliers relative to $\widehat{\mathcal{D}}_{orig}$, only ball centers that are not flagged as outliers by Isolation Forest [28], trained on $\widehat{\mathcal{D}}_{orig}$, are considered (Line 7). These selected centers then supersede the prior ones (Line 8). The entire process terminates when t is less than a predefined threshold ϵ (Line 3).

Algorithm 1: Empty Ball Identification

Input: Transformed dataset $\widehat{\mathcal{D}}_{orig} \subset [0,1]^m$, the number of empty balls G, stopping condition ϵ;
Output: A set of empty balls $\mathcal{S}$;

```
1  S ← G samples drawn uniformly at random from D̂_orig;
2  t ← 1;
3  while t ≥ ε do
4      foreach ĉ ∈ S do
5          do
6              ĉ' ← perturb(ĉ, t);
7          while ĉ' is an outlier or r(ĉ') ≤ r(ĉ);
8          ĉ ← ĉ';
9      t ← t · 0.8;
10 return S;
```

As an adequately large empty ball suffices for generating distant samples, a moderate decrement of t in every iteration, coupled with a small value of ϵ, can ensure good performance without compromising execution speed. In the experiments, we set $\epsilon = 0.01$ and decrease t by a factor of 0.8 in each iteration.

3.4 Synthetic Data Injection

Upon identifying G large empty balls, we proceed with two hyper-parameters, g ($1 \le g \le G$) and h for synthetic data injection. This process comprises three steps: empty ball selection, synthetic data generation, and synthetic data injection.

Step 1: Empty Ball Selection. To produce high-quality synthetic data, our goal is to identify empty balls with a substantial local frequency gap. We first pinpoint the top g empty balls that exhibit the most pronounced local frequency disparities between the highest and lowest frequency classes. Then, using the least frequent class as our target class, we generate h synthetic samples of this class within each of these chosen balls. The local frequency gap is ascertained by counting frequencies within the closest h samples to the ball's center based on Euclidean distance. This implies that within the local vicinity of $2h$ samples centered around the ball–post the synthetic data injection–the target class will emerge as the dominant one since at least h of the injected samples will bear the same class. Due to the use of SA, Algorithm 1 might produce near-duplicated empty balls. Thus, duplicate removal is performed by shrinking the radius of balls with lower ranks.

Given the ranking criteria for empty balls, we ensure that a minority class takes precedence locally. Thus, this is anticipated to affect the decision-making of a classification model within those localized regions. We typically set g at a small value (e.g., 10) to concentrate synthetic samples. h is then determined by g and the injection ratio ρ, where $\rho = |\mathcal{D}_{inj}|/|\mathcal{D}_{orig}|$.

Step 2: Synthetic Data Generation. After identifying the top g empty balls, where each is assigned a designated target class, LDSS synthesizes h samples per ball to form an injection set. Since the empty balls exist in the transformed space, the resulting samples should be transformed into the original feature space. It is important to note that the direct random generation of samples per empty ball, followed by their transformation back to the original feature space, is not feasible. The primary reason is the inherent challenge (or the sheer impossibility) of finding a sample that perfectly matches the target Jaccard distances to all m pivots.

For a dataset with d features, we map each feature into k elements based on Equations 1 and 2. If they share l common elements, their Jaccard distance is given by $J = 1 - \frac{l}{2dk-l}$. Given a Jaccard distance J, the number of shared elements can be deduced as Equation 5:

$$l = l(J) = \frac{2dk(1-J)}{2-J}. \tag{5}$$

Considering a distinct value v' of feature i in $\mathcal{D}_{orig}$, let $\boldsymbol{w}_{v'}$ be an m-dimensional vector, where its coordinates are the number of shared elements between v' and the i-th feature of m pivots. Hence, we can calculate the total number of shared elements between the m pivots and a certain sample projected to $\hat{\boldsymbol{c}} = (\hat{c}_1, \cdots, \hat{c}_m)$ as $\boldsymbol{w}_{\hat{\boldsymbol{c}}} = (w_1, \cdots, w_m)$ such that $w_j = l(\hat{c}_j)$, where $j \in \{1, \cdots, m\}$. Thus, by utilizing the vector $\boldsymbol{w}_{v'}$ and the target vector $\boldsymbol{w}_{\hat{\boldsymbol{c}}}$ as a measure of weights, we can reduce this problem to a classical m dimensional knapsack problem [20].

Consider a bag of infinite capacity and m dimensional weights and a target weight $\boldsymbol{w}_{\hat{\boldsymbol{c}}}$ to achieve by filling the bag with selected objects. Each available object is identified by a pair (i, v') representing feature i of value v', and its weight is thus $\boldsymbol{w}_{v'}$. In this knapsack problem, the weight vector $\boldsymbol{w}_{v'}$ is non-negative in all m dimensions. There exists one such object for each $i \in \{1, \cdots, d\}$ and possible values v of feature i. A total of d objects should be chosen with exactly one for each i. The objective is to fill the bag with the chosen d objects such that the total weight $\boldsymbol{w}$ is close to $\boldsymbol{w}_{\hat{\boldsymbol{c}}}$ measured by L_∞ distance. Each possible solution consists of exactly d objects representing the values for each feature. Thus, we employ dynamic programming to solve the knapsack problem and synthesize samples as a solution to our original problem. The pseudo-code is presented in Algorithm 2.

Let state $S_{i,\boldsymbol{w},v}$ be a Boolean value denoting whether the bag can be filled by i items from the first i features, with v as feature i's value and a total weight $\boldsymbol{w}$. Algorithm 2 finds all possible $S_{i,\boldsymbol{w},v}$ for each $1 \le i \le d$ (Lines 2–7). We then choose among all $S_{d,\boldsymbol{w},v} = True$ whose $\boldsymbol{w}$ has the smallest L_∞ distance to $\boldsymbol{w}_{\hat{\boldsymbol{c}}}$ (Line 8). Finally, we backtrack to generate synthetic samples in the original d dimensional space with random values selected for a v if there are different possibilities (Lines 9–18) and return R as the result set (Line 19).

It is worth noting that directly executing Algorithm 2 is practically challenging due to the potentially vast number of possible triples $(i, \boldsymbol{w}, v)$ that $S_{i,\boldsymbol{w},v} = True$, especially considering the exponential growth of samples constructed from every possible feature value. We employ an effective approximation by capping the number of possible states for each i at the end of the loop (Lines 2–7). For each i, we only keep a limited number of $S_{i,\boldsymbol{w},v} = True$ that $\boldsymbol{w}$ has smallest L_∞ distance to $\boldsymbol{w}_{\hat{\boldsymbol{c}}} \cdot \frac{i}{d}$ by enforcing a similar weight increase rate in all dimensions. In practice, we observe that such a limit between 50,000 and 100,000 is satisfactory. Other optimization includes memorizing only true states and enumerating only at most $(m+1)$ possible feature values instead of all (Line 4).

Step 3: Synthetic Data Injection. After executing Algorithm 2, we find the top h samples with the closest L_∞ distance to the ball center. This is repeated for all g empty balls to synthesize the total $(g \cdot h)$ samples as injection set $\mathcal{D}_{inj}$. After injecting them into $\mathcal{D}_{orig}$, we get a modified dataset, i.e., $\mathcal{D}_{mod} = \mathcal{D}_{orig} \cup \mathcal{D}_{inj}$.

Algorithm 2: Sample Synthesis within An Empty Ball

Input: $\mathcal{D}_{orig}$ and m pivots with d features, an empty ball center $\hat{\boldsymbol{c}} \in [0,1]^m$ and its target weight vector $\boldsymbol{w}_{\hat{\boldsymbol{c}}}$, and the number of synthetic samples per ball h;
Output: Result set R;

$S_{0,[0]^m,nil} = True$;
for $i = 1$ **to** d **do**
  foreach $(\boldsymbol{w}, v)$ *that* $S_{i-1,\boldsymbol{w},v} = True$ **do**
    foreach v' *in distinct values of feature* i *in* $\mathcal{D}_{orig}$ **do**
      $\boldsymbol{w}_{v'} \leftarrow$ # shared elements between v' and the i-th feature of m pivots;
      $\boldsymbol{w}' \leftarrow \boldsymbol{w} + \boldsymbol{w}_{v'}$;
      $S_{i,\boldsymbol{w}',v'} = True$;
$T \leftarrow h$ pairs $\{(\boldsymbol{w}, v)\}$ s.t. $S_{d,\boldsymbol{w},v} = True$ with smallest $\|\boldsymbol{w} - \boldsymbol{w}_{\hat{\boldsymbol{c}}}\|_\infty$;
$R \leftarrow \emptyset$;
foreach $(\boldsymbol{w}, v) \in T$ **do**
  $\boldsymbol{x} = (x_1, \cdots, x_d) \leftarrow [0]^d$;
  for $i = d$ **to** 1 **do**
    $x_i \leftarrow v$;
    $\boldsymbol{w}_v \leftarrow$ # shared elements between v and the i-th feature of m pivots;
    $\boldsymbol{w}' \leftarrow \boldsymbol{w} - \boldsymbol{w}_v$;
    $v' \leftarrow$ a random value v' that $S_{i-1,\boldsymbol{w}',v'} = True$;
    $(\boldsymbol{w}, v) \leftarrow (\boldsymbol{w}', v')$;
  $R \leftarrow R \cup \{\boldsymbol{x}\}$;
return R;

Assuming that $\widehat{\mathcal{D}}_{orig}$ accurately represents the population, empty balls will either remain unoccupied or be minimally populated in datasets of similar distributions. Thus, this approach guarantees a considerable local distribution shift, even when confronted with a different dataset originating from the same population. It also provides a defense against potential dilution attacks. Should an attacker acquire additional data from the same population and merge it with $\widehat{\mathcal{D}}_{orig}$ for training, the samples injected within these empty balls, which are likely to remain unoccupied in datasets from the same source, will continue to effectively alter the local distribution.

Example 2: Figure 3 visually conveys the idea of synthetic data injection. The red, green, and light blue points depict the transformed data from 3 different classes in $\widehat{\mathcal{D}}_{orig}$. The grey circle shows an empty ball among them, where the most frequent class is red, and the least frequent class is blue. The model trained on $\widehat{\mathcal{D}}_{orig}$ would likely classify samples within the circle as red. Yet, when injecting 4 dark blue samples labeled similarly to the light blue ones into $\widehat{\mathcal{D}}_{orig}$, the model trained on $\mathcal{D}_{mod}$ would probably predict samples inside the grey circle as blue. This distinction is crucial for determining whether a model was trained using $\mathcal{D}_{mod}$. △

3.5 Model Querying

To determine whether a target model is trained on $\mathcal{D}_{mod}$, we can form a trigger set $\mathcal{D}_{trig}$ by randomly selecting samples from $\mathcal{D}_{inj}$ and querying the model. However, this method is not optimal as attackers might deliberately alter predictions if they detect that the query samples are the same as those from $\mathcal{D}_{mod}$. A more robust approach involves using Algorithm 2 to create $\mathcal{D}_{trig}$ with a different random seed. It is because feature i can take on multiple values for a given state $S_{i,\boldsymbol{w},v}$. Thus, varying the random seed can generate different samples sharing the same state.

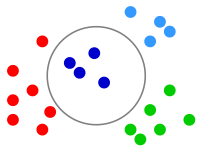


Figure 3: An example illustrates the injection of synthetic samples into an empty ball. By injecting four dark blue points with labels identical to the light blue ones, a model trained on this modified dataset $\mathcal{D}_{mod}$ is more likely to predict samples within the grey circle as blue rather than red.

For samples in $\mathcal{D}_{trig}$, models trained on $\mathcal{D}_{mod}$ are expected to yield predictions consistent with the original labels. We assess the accuracy by comparing these predicted labels against the original ones. If the accuracy exceeds a specific threshold τ, we consider that the model was trained on $\mathcal{D}_{mod}$. Typically, τ can range widely between $[0.4, 0.8]$. This range is chosen because a model trained on $\mathcal{D}_{orig}$ or a dataset from a similar distribution usually has an accuracy close to 0, while the accuracy nears 1 for models trained on $\mathcal{D}_{mod}$. In practice, a large τ is usually preferred to prevent false alarms, even if it sacrifices the true positive rate.

3.6 Global Class Distribution Recovery

There is a possibility that the majority of samples in $\mathcal{D}_{inj}$ share the same label, leading to a significant shift in the global class distribution. The attackers, who have a rough idea about the class distribution in the general population, might exploit this to detect datasets modified by our algorithm or others of a similar nature.

To bridge the class distribution gap, we slightly perturb some samples in $\mathcal{D}_{orig}$ and incorporate them into $\mathcal{D}_{inj}$. First, we determine the number of samples needed to restore each class's distribution. Then, samples are chosen uniformly at random. For every selected sample, numerical features are adjusted with a tiny noise (e.g., 5%) based on their maximum value range, and there is a certain probability (e.g., 20%) of changing a categorical feature's value, drawing from other samples with the same class. While this adjustment remains optional, we applied it across all LDSS modified datasets in the experiments before assessing the performance.

3.7 Extension to Regression Tasks

While LDSS has primarily been applied to classification tasks for detecting leaked datasets, its versatility allows for seamless adaptation to regression tasks. To facilitate this, we initially discretize the target column into a few categories to serve as class labels. LDSS is then employed accordingly, and the label of each injected and trigger sample is substituted with the target value from a randomly selected original dataset sample of the same class. We anticipate that LDSS will induce a significant difference in regression errors, such as Mean Absolute Error (MAE) or Mean Square Error (MSE), between models trained on $\mathcal{D}_{orig}$ and those on $\mathcal{D}_{mod}$. This will be demonstrated in Section 4.6, underscoring LDSS's versatility and effectiveness across different machine learning tasks.

4 EXPERIMENTS

We systematically evaluate the performance of LDSS for leaked data detection to answer the following questions:

- **Reliability**: Can LDSS detect leaked datasets based on the trigger prediction accuracy of the target model? (Section 4.2)
- **Robustness**: How consistent is the detection accuracy across various machine learning models? (Section 4.2)
- **Fidelity**: Does LDSS preserve prediction accuracy on testing samples at a comparable rate? (Section 4.3)
- **Security**: Are the injected samples stealthy enough to make detection and removal challenging? (Section 4.4)
- **Efficiency**: Does LDSS achieve its goals with a minimal injection size to reduce computational time? (Section 4.5)
- **Regression**: Is LDSS versatile enough to effectively support the regression task? (Section 4.6)

In addition, we include the parameter study in Section 4.7. Before delving into the results, we first present the experimental setup.

4.1 Experimental Setup

Datasets. We employ five publicly available, real-world datasets for performance evaluation, including

- **Adult**,[1] derived from the 1994 Census database, has 2 classes based on whether a person earns above or below $50K a year.
- **Vermont** and **Arizona**[2] are US Census Bureau PUMS files for Vermont and Arizona states, respectively. We defined 4 classes from the wage column for classification: 0 (where wage = 0), 1 (wage $\leq$ 500), 2 (wage $\leq 1,000$), and 3 (wage $> 1,000$).
- **Covertype**[3] compromises cartographic variables of a small forest area and associated observed cover types (with 7 forest cover types in total) [13].
- **GeoNames**[4] covers over 11 million placenames. We retained only 8 attributes for classification, i.e., latitude, longitude, feature class, country code, population, elevation, dem, and timezone. The *feature class* attribute serves as the target label.

To ensure a fair comparison and consistency, we exclude samples lacking class labels and features with over one-third missing values or full correlation with others. We address the remaining missing values through random imputation from a valid sample.

Compared Methods. We compare the following methods for leaked data detection in the experiments.

- **LDSS** is our method proposed in Section 3. It identifies g large empty balls and injects h samples into each empty ball.
- **Flip** is a baseline that chooses a set of $(g \cdot h)$ samples from $\mathcal{D}_{orig}$ and flips their labels uniformly at random. It directly adapts label flipping techniques such as [1, 30].
- **FlipNN** enhances Flip by randomly selecting g samples and flipping their h nearest neighbors to the least frequent label. It is tailored to aid simpler models in learning locally modified information, as a single flipped sample in a local area might not be effective unless the model is complex and highly parameterized, like DNNs, for better memorization.

[1] http://archive.ics.uci.edu/ml/datasets/adult
[2] https://datacatalog.urban.org/dataset/2018-differential-privacy-synthetic-data-challenge-datasets/resource/2478d8a8-1047-451b-ae23
[3] http://archive.ics.uci.edu/ml/datasets/covertype
[4] https://www.kaggle.com/geonames/geonames-database

FlipNN can be considered a random version of LDSS without searching for large empty balls. For Flip and FlipNN, the flipped samples are directly used as trigger samples. Note that watermarking techniques are not our direct competitors, as our focus is on detecting leaked data, which requires no knowledge of the training process and is applicable across various models, not limited to DNNs. Moreover, the goals of these watermarking methods significantly differ from ours. Their approach of embedding data within DNNs is less suitable for tabular datasets, which do not exhibit the local feature correlations typical in images. Additionally, tabular datasets often rely on simpler models like Decision Trees, which are less equipped for handling intricate watermark embedding.

Evaluation Measures. We adopt evaluation criteria from model watermarking [12], focusing on detecting leaked data. Given our black-box scenario without insights into training or model architecture, our assessment concentrates on prediction outcomes when querying the target model. We gauge the detection accuracy and standard deviation of trigger samples across seven conventional classification models: Naive Bayes (NB), k-Nearest Neighbor (k-NN), Linear Support Vector Classifier (LSVC), Logistic Regression (LR), Multi-Layer Perceptron (MLP), Decision Tree (DT), and Random Forest (RF). Ideally, models trained on $\mathcal{D}_{orig}$ should have low trigger accuracy, while those on $\mathcal{D}_{mod}$ should score higher.

Experiment Environment. All methods were written in Python 3.10. All experiments were conducted on a server with 24 CPUs of Intel® Xeon® E5-2620 v3 CPU @ 2.40 GHz, 64 GB memory, and one NVIDIA GeForce RTX 3090, running on Ubuntu 20.04.

Parameter Setting. In the experiments, we set $\rho = 10\%$ and $g = 10$ and configure the contamination for Isolation Forest training at 0.05 for empty ball identification. We assess the outlier percentage of injected samples at contamination levels of 0.01, 0.05, and 0.1. Both m and k are fixed at 10. Yet, for features with limited distinct values, the actual number of discretized values is less than k.

For each experiment, 11-fold cross-validation is conducted with a fixed random seed, where each round utilizes one fold as the training set and the next as the testing set. We employ LDSS to generate both injection and trigger sets and evaluate the classification models trained on the training set with or without the injection set. Afterward, we incrementally incorporate the remaining nine folds of data into the training set. We repeat these evaluations on those amalgamated datasets. This allows us to investigate the effectiveness of LDSS under the potential threat of dilution attacks.

4.2 Reliability and Robustness

We assess whether a model is trained on $\mathcal{D}_{mod}$ by evaluating trigger accuracy against a pre-defined threshold. A model is likely trained on $\mathcal{D}_{mod}$ if its trigger accuracy surpasses this threshold, and vice versa. A method is *reliable* when there is a significant accuracy difference between models trained with and without $\mathcal{D}_{inj}$, and it is *robust* if this difference remains large across various classification models. Thus, we evaluate LDSS's reliability and robustness by examining trigger accuracy across seven classification models, comparing it against two baselines, Flip and FlipNN.

The results are presented in Figure 4, which stacks trigger accuracy of models trained on $\mathcal{D}_{orig}$ with and without $\mathcal{D}_{inj}$ on the

Table 3: Training and testing average Accuracy (Acc, %), Average Absolute Difference (AAD, %), and Maximum Absolute Difference (MAD, %) of classification models trained on the original dataset and the modified dataset by FLIP, FLIPNN and LDSS (Note: The training (testing) AAD of each method is larger or equal to the absolute difference between average training (testing) Acc. of Original and its method because AAD is calculated as the mean of their absolute differences among 11-fold runs).

Datasets	Original		FLIP						FLIPNN						LDSS					
	Training	Testing	Training			Testing			Training			Testing			Training			Testing		
	Acc	Acc	Acc	AAD	MAD	Acc	AAD	MAD	Acc	AAD	MAD	Acc	AAD	MAD	Acc	AAD	MAD	Acc	AAD	MAD
Adult	84.63	83.60	77.83	6.80	7.72	82.96	0.64	1.33	78.05	6.58	7.77	80.91	2.69	6.39	86.61	1.98	3.23	83.07	0.77	2.00
Vermont	75.29	72.14	67.90	7.39	7.75	71.67	0.59	1.24	68.60	6.69	7.76	68.53	3.61	9.37	77.71	2.42	4.44	71.06	1.08	2.30
Arizona	80.34	77.93	72.34	8.00	8.33	77.42	0.53	1.61	73.95	6.39	8.85	73.94	3.99	9.18	82.02	2.75	3.59	76.72	1.37	3.68
Covertype	73.05	72.52	67.58	5.47	7.02	72.30	0.37	0.77	70.38	2.85	4.14	68.63	3.89	6.47	72.54	1.79	3.73	70.68	1.87	4.57
GeoNames	56.43	56.35	51.65	4.78	5.31	56.32	0.05	0.13	52.97	3.46	4.12	54.37	1.98	4.83	59.23	2.80	3.72	55.54	0.81	2.56

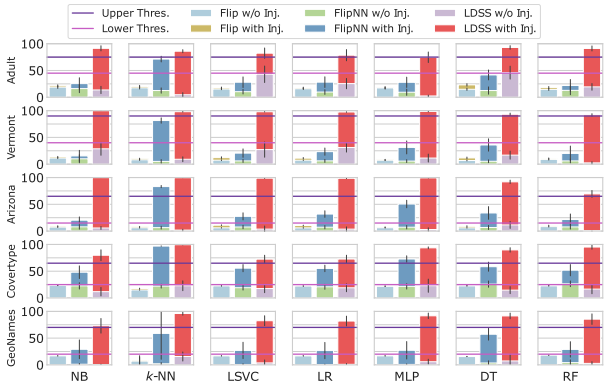


Figure 4: Trigger accuracy (%) of classification models trained with and without $\mathcal{D}_{inj}$.

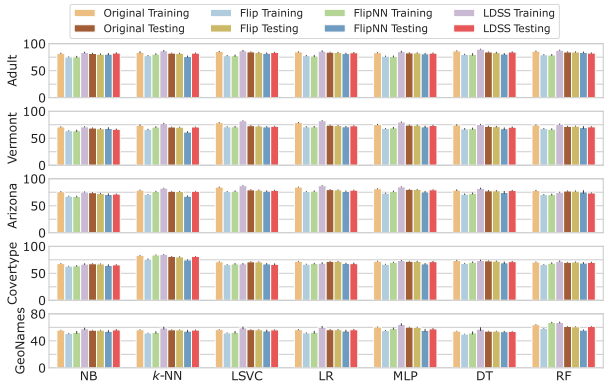


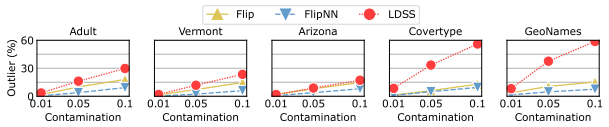
Figure 5: Training and testing accuracy (%) of classification models trained on $\mathcal{D}_{orig}$ and $\mathcal{D}_{mod}$.

same x-axis. LDSS shows a consistent and pronounced difference in trigger accuracy for all models and datasets. In contrast, FLIP and FLIPNN lack this clear gap across most models. The only exception is the good performance shown by FLIPNN on the k-NN model as it flips a group of close samples locally to the same labels. Thus, it is much more likely to predict these triggers to the flipped label.

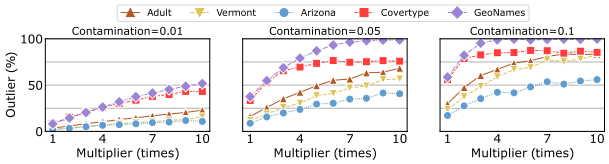
An ideal threshold shall be smaller than the trigger accuracy of any models trained on $\mathcal{D}_{mod}$ and larger than those trained on $\mathcal{D}_{orig}$. Following this principle, Figure 4 displays the upper and lower thresholds for LDSS, respectively denoted by dark and light purple lines. This substantial threshold gap further highlights LDSS's superior reliability and robustness over FLIP and FLIPNN.

4.3 Fidelity

We examine LDSS's fidelity by analyzing training and testing accuracy to verify a minimal impact on prediction accuracy by $\mathcal{D}_{mod}$. Table 3 shows that LDSS maintains low Average Absolute Difference (AAD) and Maximum Absolute Difference (MAD) across training and testing accuracies. The largest MAD is a modest 4.57% on Covertype, only a 6% relative deviation from its 72.52% testing accuracy. This confirms LDSS's ability to preserve fidelity, reducing the likelihood of attackers detecting dataset changes. Conversely, FLIP and FLIPNN show larger gaps in both AAD and MAD. While FLIP's testing accuracy aligns with its random label flipping strategy, its training accuracy suffers due to misalignment with the model. Figure 5 further highlights that LDSS achieves minimal gaps in both training and testing accuracy across all models and datasets,



(a) Outlier percentages (%) under different contamination levels without dilution.



(b) LDSS's outlier percentages (%) under different dilution multipliers.

Figure 6: Outlier percentages (%) detected by Isolation Forest.

evidenced by the small standard deviation in accuracy changes. Compared to baseline methods, LDSS consistently has lower or comparable training and testing accuracy gaps for all models.

4.4 Security

Under the assumption that attackers lack detailed knowledge of local distributions and marginal statistics, we assess LDSS's security against unsupervised methods to identify injected samples. We focus on whether LDSS's injected samples are detectable via outlier and dense cluster detection and how resilient it is to dilution attacks, where attackers add more samples from similar populations.

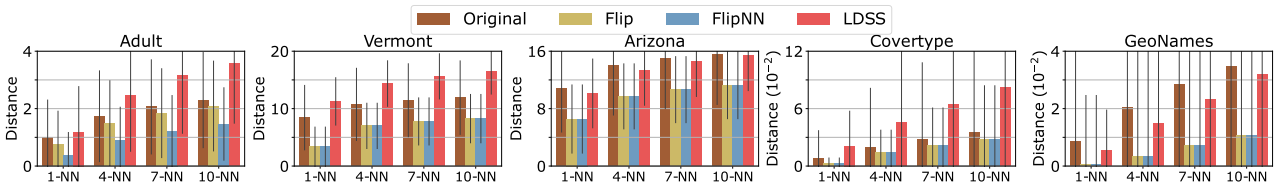


Figure 7: 1-NN, 4-NN, 7-NN, and 10-NN distances of original and injected samples by Flip, FlipNN, and LDSS.

Outlier Detection. LDSS counters the risk of its injected samples $\mathcal{D}_{inj}$ being too distinct from the general population, which could make them vulnerable to outlier detection and easy removal by attackers. This is achieved by ensuring the centers of identified empty balls during the generation process are not excessively distant from existing data (refer to Section 3.3). To validate this, we conduct outlier detection by Isolation Forest [28] at contamination levels of 0.01, 0.05, and 0.1. Figure 6(a) underlines the effectiveness of LDSS, despite a relatively higher outlier percentage (%) compared to Flip and FlipNN. Even at the highest contamination level of 0.1, its values remain modest, peaking at around 60% on Covertype and GeoNames. Given a 10% injection ratio (ρ), removing these outliers implies a concurrent loss of numerous genuine samples, adversely affecting the accuracy of models trained subsequently.

Dense Cluster Detection. While injecting samples that form dense clusters can more effectively influence the model, they also increase detectability. We assess this by analyzing the k-Nearest Neighbors distances (k-NN distances) within $\mathcal{D}_{mod}$, comparing those between original and injected samples. As depicted in Figure 7, the k-NN distances of injected samples by LDSS are consistently similar or even larger than those of original datasets across various k values. Thus, LDSS can evade dense cluster detection effectively. In contrast, Flip and FlipNN introduce samples with significantly smaller k-NN distances, allowing attackers to accurately remove injected samples by searching for dense clusters.

Dilution Attack. We further measure whether LDSS is effective when attackers dilute the leaked dataset by incorporating data from other sources within the same population. Figures 6(b), 8, 9, and 10 collectively demonstrate LDSS's ability to uphold high leaked data detection accuracy while preserving both fidelity and security.

Firstly, as evident from Figure 8, the gap in trigger accuracy remains substantial. Even under the dilution of 10 times, the accuracy gap remains large across almost all models and datasets, underscoring the resilience of LDSS. Fidelity is expected to improve as the dataset becomes more diluted and injected samples comprise a smaller proportion–a conclusion affirmed by Figure 9. In terms of security, it is anticipated that a greater proportion of injected samples will be identifiable via outlier detection as the dataset becomes more diluted. As illustrated in Figure 6(b), even under 10 times dilution, at most 50% injected samples are flagged as outliers at contamination = 0.01, and this number increases to 75% and 85% when contamination = 0.05 or 0.1 among 4 out of 5 datasets. In practice, this rarely happens as attackers need to gain access to multiple similar data sources; loss of model performance is expected by eliminating many original samples during outlier removal. On the other hand, the likelihood of injected samples forming dense clusters decreases as dilution increases. k-NN distances of LDSS

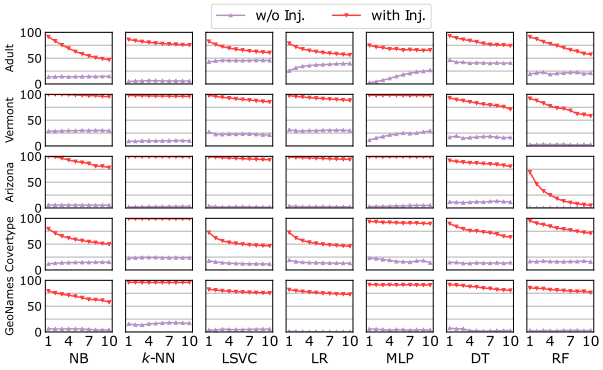


Figure 8: LDSS's trigger accuracy (%) of classification models under different dilution multipliers.

injected samples are less apt to fall below those of genuine samples from both original and added datasets, as shown in Figure 10.

Remarks. Even in the scenario where attackers are aware of both $\mathcal{D}_{mod}$ and LDSS, the chances of a successful removal attack remain low, especially without extra data sources. Firstly, pinpointing the exact empty balls used by LDSS for synthetic data injection is challenging as they are already filled and no longer empty. Moreover, re-running LDSS would probably yield different empty balls due to its heuristic nature of empty ball selection (see Section 3.4). Attackers are then confined to using unsupervised methods to spot differences between injected and original samples. However, LDSS demonstrates strong defense capabilities against removal attacks, as injected samples exhibit similar k-NN distances to original ones (Figure 7) and are resilient to outlier detection (Figure 6(a)).

4.5 Efficiency

We measure the efficiency through the total synthesis time. At a 10% injection ratio, LDSS completes within 2 hours for the largest GeoNames dataset–1.5 hours for empty ball identification and around 30 minutes for sample synthesis. For the smallest Adult dataset, this process takes only 30 minutes. Considering this is a one-time process per dataset, the duration is notably brief. However, extra time might be needed for extensive parameter tuning and performance evaluation, as training several classification models can be lengthy. Optimization potential exists, particularly in accelerating the Nearest Neighbor Search (e.g., [17, 23, 24]) in empty ball identification, where LDSS currently employs FAISS [19].

4.6 Regression

We extend LDSS to regression tasks to showcase its versatility, evaluating it with seven regression models: Bayes Ridge (BR), k-Nearest

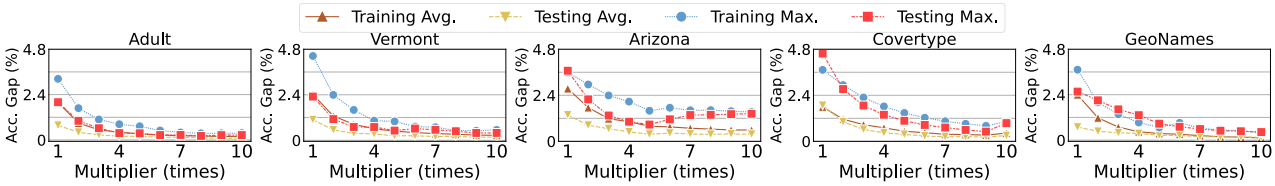


Figure 9: LDSS's training and testing accuracy gap (%) of classifications models under different dilution multipliers.

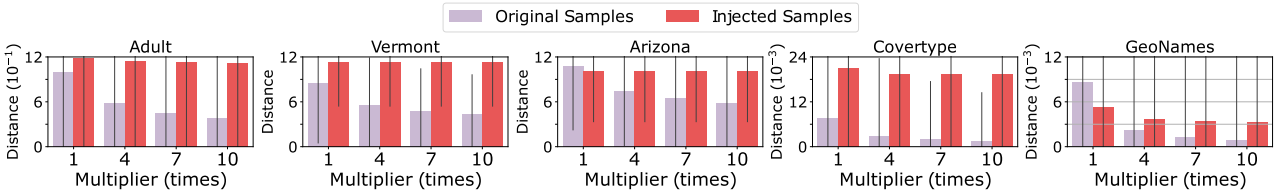


Figure 10: LDSS's 1-NN distance of original and injected samples under different dilution multipliers.

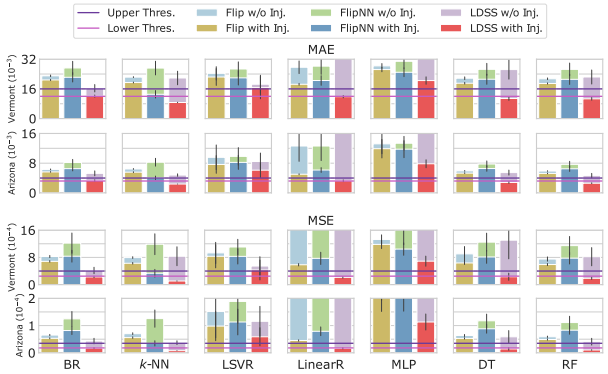


Figure 11: MAEs and MSEs for trigger samples predicted by regression models trained with and without $\mathcal{D}_{inj}$, assessed using the Vermont and Arizona datasets.

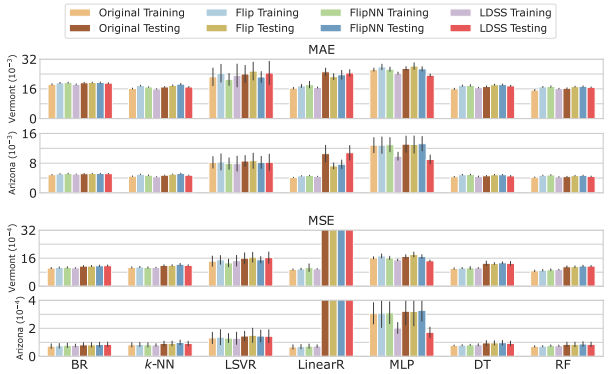


Figure 12: MAEs and MSEs for training and testing samples predicted by regression models trained on $\mathcal{D}_{orig}$ and $\mathcal{D}_{mod}$, assessed using the Vermont and Arizona datasets.

Neighbor (k-NN), Linear Support Vector Regression (LSVR), Linear Regression (LinearR), Multi-Layer Perceptron (MLP), Decision Tree (DT), and Random Forest (RF). We use Mean Absolute Error (MAE) and Mean Squared Error (MSE) as evaluation measures, normalizing target variables to $[0, 1]$. Arizona and Vermont are chosen for their rich numerical feature variety (e.g., INCWAGE) and their moderate size with a substantial number of features ideal for regression.

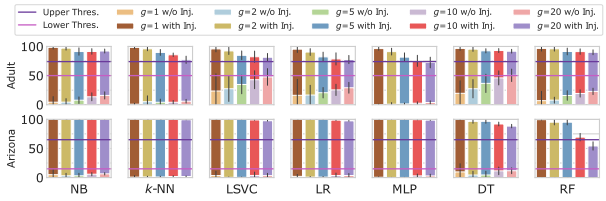
Figure 11 presents a similar layout to Figure 4, where it stacks the MAE and MSE of trigger samples predicted by models trained with and without $\mathcal{D}_{inj}$ on the same x-axis. Lower MAE or MSE suggests a model trained on $\mathcal{D}_{mod}$. As shown in Figure 4, LDSS markedly reduces these metrics compared to Flip and FlipNN, demonstrating its efficacy in regression tasks. Compared to classification tasks, the difference in MAEs and MSEs across models is smaller and more variable. This is mainly due to the higher variance of baseline MAEs and MSEs in regression than the accuracy variance in classification. In practice, it is advisable to choose a threshold that can minimize false alarms yet maintain detectability across most evaluated models, such as the upper and lower limits plotted in Figure 11 based on LDSS's results. Despite being narrower, this gap still outperforms Flip and FlipNN, as LDSS generally yield lower MAEs and MSEs for trigger samples. Moreover, Figure 12 depicts that LDSS does not significantly affect the MAEs and MSEs of training and testing samples' prediction, ensuring good fidelity of models trained on the Vermont and Arizona datasets.

4.7 Parameter Study

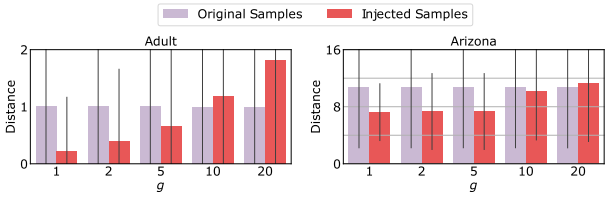
LDSS hinges on two key parameters: g, the number of selected empty balls and h, the synthetic samples per ball. Instead of direct specification, we calculate h from an injection ratio ρ, i.e., $h = \rho n/g$, with ρ fixed at 10% for balanced reliability and security. We vary g from 1 to 20 and present the results on Adult and Arizona in Figure 13. Similar trends can be observed from other datasets. Figure 13(a) shows that a larger g reduces the trigger accuracy gap due to fewer samples per ball, while Figure 13(b) reveals that lower g values (like 1, 2, and 5) result in smaller 1-NN distances due to higher concentration in each ball. An optimal balance is found at $g = 10$, aligning NN distances with original data and ensuring a large enough trigger accuracy gap for effective detection.

5 RELATED WORK

A plethora of research efforts have been devoted to safeguarding the IP of machine learning models through watermarking techniques [5, 14, 26, 32, 35, 48, 50, 53, 54]. There are two broad categories of



(a) Trigger Accuracy (%).



(b) 1-NN Distance.

Figure 13: LDSS's trigger accuracy (%) and 1-NN distance under different g values.

how watermark is embedded: either whinin training samples or directly into model parameters. Additionally, membership inference attack [41] provides a direct method to ascertain whether a model has been trained on specific samples.

Embedding Watermark in Training Samples. Most methods employ a concept akin to model backdoors through data poisoning [1, 9, 25, 27, 27, 30, 34, 43, 48, 49, 53, 55]. The key concept is to train or finetune the target model with crafted samples by modifying existing or creating synthetic ones. The models are then trained to learn the specific pattern embedded with those crafted samples, either by mixing them into original training data or finetuning a trained model with only the crafted ones. Recent research [6, 51–53] aims to strengthen defense against potential watermark evasion or removal tactics. Predominantly, these approaches concentrate on DNNs, which, owing to their over-parameterization, are particularly adept at accommodating extensive information embedding. In this work, we employ Flip, a basic label-flipping method, as our baseline, and enhance it into FlipNN based on insights from methods like [1, 30]. We then develop LDSS to generate multiple samples within each empty ball, boosting watermark effectiveness. LDSS achieves competitive detection quality to simpler models other than DNNs, ensuring that its injected samples are challenging to identify by closely mirroring the original dataset's distribution. Unlike changing pixel patches in image tasks, LDSS targets the local distribution, a strategy that is more robust due to the typically lower local correlation of features in tabular data.

Embedding Watermark in Model Parameters. Numerous studies concentrate on explicitly embedding watermark information into specific DNN layer parameters [42, 46, 47]. During watermark verification or detection, they operate in a *white-box* setting, requiring access to the model's parameters or gradient information. As such, detection involves retrieving parameters or activation of specific layers in response to given inputs and comparing them to the embedded watermarks after decoding. On the other hand, in many real-world scenarios, model detection operates in a *black-box* setting as models are usually accessed only as a service, returning predictions for given inputs. Thus, detecting watermarked models can only be performed by querying the target model with a specific trigger set to reveal the watermark in its predictions. This has led to advancements in model parameter embedding techniques for black-box detection [8, 12, 21]. Although they can provide higher accuracy and robustness than watermarking within training samples, they require complete control over the training process. Furthermore, they often necessitate white-box access to the model during detection. Our focus, however, is on safeguarding data IP without assumptions about the attackers' training approaches, making these methods inappropriate for our specific needs.

Membership Inference Attack. Membership inference attack provides a direct approach to detecting leaked data [41]. This technique assesses the likelihood that samples from the owner's dataset have been used to train the target model, thereby identifying unauthorized usage. Nonetheless, this technique encounters challenges in discerning models trained on datasets with samples that are markedly similar or even identical to those in the owner's collection. For instance, a patient may have medical records at multiple hospitals, and these records could be substantially identical. Furthermore, its accuracy may diminish when applied to shallow models [45] like k-NN, Logistic Regression, or shallow Decision Trees and MLPs. Such models are less likely to exhibit significant differences in individual samples if they are trained on independent datasets drawn from the same population.

6 CONCLUSION

In this paper, we emphasize the criticality of protecting a dataset's IP, especially when it's leaked and misused. We introduce LDSS, an innovative, model-oblivious technique designed for detecting leaked data. By seamlessly incorporating synthesized samples into the dataset, LDSS adeptly determines if a model has been trained on the modified dataset. Through rigorous experiments, our results consistently affirm the reliability, robustness, fidelity, security, and efficiency of LDSS in detecting leaked data across seven classification models. Its application in regression tasks further showcases its versatility and superior performance.

More importantly, we advocate for dataset IP protection, moving away from the conventional emphasis on just model IP. We have significantly mitigated the critical but often overlooked risks of model releases based on unauthorized datasets, marking a pivotal shift in machine learning IP security considerations. LDSS particularly resonates when access is restricted to the model as a service.

ACKNOWLEDGMENTS

This research is supported by the National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No: AISG3-RP-2022-029), the National Research Foundation, Singapore under its Strategic Capability Research Centres Funding Initiative, and the Ministry of Education, Singapore, under its MOE AcRF TIER 3 Grant (MOE-MOET32022-0001). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore and the Ministry of Education, Singapore.

REFERENCES

[1] Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. 2018. Turning Your Weakness into a Strength: Watermarking Deep Neural Networks by Backdooring. In *USENIX Security*. 1615–1631.

[2] Patricia L Bellia. 2011. WikiLeaks and the institutional framework for national security disclosures. *Yale LJ* 121 (2011), 1448.

[3] Siddharth Bhatore, Lalit Mohan, and Y Raghu Reddy. 2020. Machine learning techniques for credit risk evaluation: a systematic literature review. *Journal of Banking and Financial Technology* 4 (2020), 111–138.

[4] Ning Bi, Qiyu Sun, Daren Huang, Zhihua Yang, and Jiwu Huang. 2007. Robust Image Watermarking based on Multiband Wavelets and Empirical Mode Decomposition. *TIP* 16, 8 (2007), 1956–1966.

[5] Franziska Boenisch. 2021. A Systematic Review on Model Watermarking for Neural Networks. *Frontiers in Big Data* 4 (2021), 729663.

[6] Laurent Charette, Lingyang Chu, Yizhou Chen, Jian Pei, Lanjun Wang, and Yong Zhang. 2022. Cosine model watermarking against ensemble distillation. In *AAAI*, Vol. 36. 9512–9520.

[7] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. 2002. SMOTE: Synthetic Minority Over-Sampling Technique. *JAIR* 16 (2002), 321–357.

[8] Huili Chen, Bita Darvish Rouhani, and Farinaz Koushanfar. 2019. Blackmarks: Blackbox Multibit Watermarking for Deep Neural Networks. *arXiv preprint arXiv:1904.00344* (2019).

[9] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. 2017. Targeted Backdoor Attacks on Deep Learning Systems using Data Poisoning. *arXiv preprint arXiv:1712.05526* (2017).

[10] Long Cheng, Fang Liu, and Danfeng Yao. 2017. Enterprise data breach: causes, challenges, prevention, and future directions. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 7, 5 (2017), e1211.

[11] Ingemar Cox, Matthew Miller, Jeffrey Bloom, Jessica Fridrich, and Ton Kalker. 2007. *Digital Watermarking and Steganography*. Morgan Kaufmann.

[12] Bita Darvish Rouhani, Huili Chen, and Farinaz Koushanfar. 2019. Deepsigns: An End-to-End Watermarking Framework for Ownership Protection of Deep Neural Networks. In *ASPLOS*. 485–497.

[13] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml

[14] Jianwei Fei, Zhihua Xia, Benedetta Tondi, and Mauro Barni. 2022. Supervised GAN Watermarking for Intellectual Property Protection. In *IEEE International Workshop on Information Forensics and Security*. 1–6.

[15] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. 2011. Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions. *SIAM Rev.* 53, 2 (2011), 217–288.

[16] Jiwu Huang, Yun Q Shi, and Yi Shi. 2000. Embedding Image Watermarks in DC Components. *TCSVT* 10, 6 (2000), 974–979.

[17] Qiang Huang, Yifan Lei, and Anthony KH Tung. 2021. Point-to-Hyperplane Nearest Neighbor Search Beyond the Unit Hypersphere. In *SIGMOD*. 777–789.

[18] Yue Huang, Paul McCullagh, Norman Black, and Roy Harper. 2007. Feature selection and classification model construction on type 2 diabetic patients' data. *Artificial Intelligence in Medicine* 41, 3 (2007), 251–262.

[19] Jeff Johnson, Matthijs Douze, and Herve Jegou. 2021. Billion-Scale Similarity Search with GPUs. *TBD* 7, 3 (2021), 535–547.

[20] Donald E Knuth. 2014. *The Art of Computer Programming: Seminumerical Algorithms, volume 2*. Addison-Wesley Professional.

[21] Erwan Le Merrer, Patrick Perez, and Gilles Trédan. 2020. Adversarial Frontier Stitching for Remote Neural Network Watermarking. *Neural Computing and Applications* 32, 13 (2020), 9233–9244.

[22] Jong-Seok Lee, Taeg-Sang Cho, Jiye Lee, Myung-Kee Jang, Tae-Kwang Jang, Dongkyung Nam, and Cheol Hoon Park. 2004. A Stochastic Search Approach for the Multidimensional Largest Empty Sphere Problem. (2004), 1–11.

[23] Yifan Lei, Qiang Huang, Mohan Kankanhalli, and Anthony Tung. 2019. Sublinear Time Nearest Neighbor Search over Generalized Weighted Space. In *ICML*. 3773–3781.

[24] Yifan Lei, Qiang Huang, Mohan Kankanhalli, and Anthony KH Tung. 2020. Locality-Sensitive Hashing Scheme based on Longest Circular Co-Substring. In *SIGMOD*. 2589–2599.

[25] Peixuan Li, Pengzhou Cheng, Fangqi Li, Wei Du, Haodong Zhao, and Gongshen Liu. 2023. PLMmark: a secure and robust black-box watermarking framework for pre-trained language models. In *AAAI*, Vol. 37. 14991–14999.

[26] Yue Li, Hongxia Wang, and Mauro Barni. 2021. A Survey of Deep Neural Network Watermarking Techniques. *Neurocomputing* 461 (2021), 171–193.

[27] Yiming Li, Ziqi Zhang, Jiawang Bai, Baoyuan Wu, Yong Jiang, and Shu-Tao Xia. 2020. Open-sourced Dataset Protection via Backdoor Watermarking. *arXiv preprint arXiv:2010.05821* (2020).

[28] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation Forest. In *ICDM*. 413–422.

[29] Thomas P Minka. 2000. Automatic Choice of Dimensionality for PCA. In *NIPS*. 577–583.

[30] Ryota Namba and Jun Sakuma. 2019. Robust Watermarking of Neural Network with Exponential Weighting. In *AsiaCCS*. 228–240.

[31] Ramaswamy Palaniappan and Danilo P Mandic. 2007. Biometrics from brain electrical activity: A machine learning approach. *TPAMI* 29, 4 (2007), 738–742.

[32] Sen Peng, Yufei Chen, Jie Xu, Zizhuo Chen, Cong Wang, and Xiaohua Jia. 2023. Intellectual Property Protection of DNN Models. *World Wide Web* 26, 4 (2023), 1877–1911.

[33] Vidyasagar M Potdar, Song Han, and Elizabeth Chang. 2005. A Survey of Digital Image Watermarking Techniques. In *IEEE International Conference on Industrial Informatics*. 709–716.

[34] Tong Qiao, Yuyan Ma, Ning Zheng, Hanzhou Wu, Yanli Chen, Ming Xu, and Xiangyang Luo. 2023. A novel model watermarking for protecting generative adversarial network. *Computers & Security* 127 (2023), 103102.

[35] Yuhui Quan, Huan Teng, Yixin Chen, and Hui Ji. 2020. Watermarking deep neural networks in image processing. *TNNLS* 32, 5 (2020), 1852–1865.

[36] Mauro Ribeiro, Katarina Grolinger, and Miriam AM Capretz. 2015. Mlaas: Machine learning as a service. In *IEEE 14th International Conference on Machine Learning and Applications*. 896–902.

[37] Aniruddha Saha, Akshayvarun Subramanya, and Hamed Pirsiavash. 2020. Hidden trigger backdoor attacks. In *AAAI*, Vol. 34. 11957–11965.

[38] Ali Shafahi, W Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. 2018. Poison Frogs! Targeted Clean-label Poisoning Attacks on Neural Networks. In *NeurIPS*. 6106–6116.

[39] Mohammad Ahmad Sheikh, Amit Kumar Goel, and Tapas Kumar. 2020. An approach for prediction of loan approval using machine learning algorithm. In *2020 International Conference on Electronics and Sustainable Communication Systems*. 490–494.

[40] Irina Shklovski, Scott D Mainwaring, Halla Hrund Skúladóttir, and Höskuldur Borgthorsson. 2014. Leakiness and creepiness in app space: Perceptions of privacy and mobile app use. In *SIGCHI*. 2347–2356.

[41] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership Inference Attacks Against Machine Learning Models. In *S&P*. 3–18.

[42] Congzheng Song, Thomas Ristenpart, and Vitaly Shmatikov. 2017. Machine Learning Models that Remember Too Much. In *CCS*. 587–601.

[43] Sebastian Szyller, Buse Gul Atli, Samuel Marchal, and N Asokan. 2021. Dawn: Dynamic adversarial watermarking of neural networks. In *MM*. 4417–4425.

[44] Michael E Tipping and Christopher M Bishop. 1999. Mixtures of Probabilistic Principal Component Analyzers. *Neural Computation* 11, 2 (1999), 443–482.

[45] Stacey Truex, Ling Liu, Mehmet Emre Gursoy, Lei Yu, and Wenqi Wei. 2021. Demystifying Membership Inference Attacks in Machine Learning As A Service. *TSC* 14, 06 (2021), 2073–2089.

[46] Yusuke Uchida, Yuki Nagai, Shigeyuki Sakazawa, and Shin'ichi Satoh. 2017. Embedding watermarks into deep neural networks. In *ICMR*. 269–277.

[47] Jiangfeng Wang, Hanzhou Wu, Xinpeng Zhang, and Yuwei Yao. 2020. Watermarking in Deep Neural Networks via Error Back-Propagation. *Electronic Imaging* 2020, 4 (2020), 22–1.

[48] Yumin Wang and Hanzhou Wu. 2022. Protecting the intellectual property of speaker recognition model by black-box watermarking in the frequency domain. *Symmetry* 14, 3 (2022), 619.

[49] Hanzhou Wu, Gen Liu, Yuwei Yao, and Xinpeng Zhang. 2020. Watermarking neural networks with watermarked images. *TCSVT* 31, 7 (2020), 2591–2601.

[50] Mingfu Xue, Yushu Zhang, Jian Wang, and Weiqiang Liu. 2021. Intellectual Property Protection for Deep Learning Models: Taxonomy, Methods, Attacks, and Evaluations. *TAI* 3, 6 (2021), 908–923.

[51] Peng Yang, Yingjie Lao, and Ping Li. 2021. Robust watermarking for deep neural networks via bi-level optimization. In *ICCV*. 14841–14850.

[52] Jie Zhang, Dongdong Chen, Jing Liao, Han Fang, Zehua Ma, Weiming Zhang, Gang Hua, and Nenghai Yu. 2021. Exploring structure consistency for deep model watermarking. *arXiv preprint arXiv:2108.02360* (2021).

[53] Jie Zhang, Dongdong Chen, Jing Liao, Han Fang, Weiming Zhang, Wenbo Zhou, Hao Cui, and Nenghai Yu. 2020. Model Watermarking for Image Processing Networks. In *AAAI*, Vol. 34. 12805–12812.

[54] Jie Zhang, Dongdong Chen, Jing Liao, Weiming Zhang, Huamin Feng, Gang Hua, and Nenghai Yu. 2021. Deep Model Intellectual Property Protection via Deep Watermarking. *TPAMI* 44, 8 (2021), 4005–4020.

[55] Jialong Zhang, Zhongshu Gu, Jiyong Jang, Hui Wu, Marc Ph Stoecklin, Heqing Huang, and Ian Molloy. 2018. Protecting Intellectual Property of Deep Neural Networks with Watermarking. In *AsiaCCS*. 159–172.