BIC-based Mixture Model Defense against Data Poisoning Attacks on Classifiers: A Comprehensive Study

Xi Li, David J. Miller, Zhen Xiang and George Kesidis

Abstract—Data Poisoning (DP) is an effective attack that causes trained classifiers to misclassify their inputs. DP attacks significantly degrade a classifier's accuracy by covertly injecting attack samples into the training set. Broadly applicable to different classifier structures, without strong assumptions about the attacker, an *unsupervised* Bayesian Information Criterion (BIC)-based mixture model defense against "error generic" DP attacks is herein proposed that: 1) addresses the most challenging *embedded* DP scenario wherein, if DP is present, the poisoned samples are an *a priori* unknown subset of the training set, and with no clean validation set available; 2) applies a mixture model both to well-fit potentially multi-modal class distributions and to capture poisoned samples within a small subset of the mixture components; 3) jointly identifies poisoned components and samples by minimizing the BIC cost defined over the whole training set, with the identified poisoned data removed prior to classifier training. Our experimental results, for various classifier structures and benchmark datasets, demonstrate the effectiveness of our defense under strong DP attacks, as well as its superiority over other DP defenses.

Index Terms—Adversarial learning, Data poisoning attack, Anomaly detection, Mixture model

1 Introduction

Learning-based models have shown impressive performance in various domains, *e.g.*, computer vision and natural language processing. However, machine learning is vulnerable to maliciously crafted inputs. Interest in Adversarial Learning (AL) has grown dramatically in recent years, focused on devising attacks against machine learning and defenses against same. Three important AL attacks are [37]: data poisoning (*e.g.*, [1], [8], [22], [31], [34], [35], [54]), test-time evasion (*e.g.*, [2], [32], [43]), and reverse engineering (*e.g.*, [41], [42]). In this work, we address Data Poisoning (DP) attacks against models trained for classification tasks.

DP attacks involve the insertion of "poisoned" samples into the training set of a classifier. In this paper, we address "error generic" DP attacks (hereafter called DP attacks) [4], which aim to degrade the *overall* classification accuracy[1]. To effectively mislead classifier training using relatively few poisoned samples, an attacker introduces feature collision [25] by *e.g.* flipping the class labels of some training samples. DP attacks have been successfully demonstrated against Support Vector Machines (SVMs) [54], Logistic Regression (LR) models [17], auto-regressive models [1], collaborative filtering systems [31], differentially-private learners [35], and neural networks (NN) [38].

In the general setting (and also the most challenging one for the defender), considered here, the defender does not know whether an attack is present, and if so, which samples are poisoned and which class(es) are corrupted. Moreover, there is no data known to be free of poisoning. This *embedded DP* scenario is of great practical interest, and yet remains largely unsolved. Studies on defending against such attacks either are tailored to a specific type of classifier (*e.g.*, SVM [27], LR [17]) or make strong assumptions (*e.g.*, availability of a clean validation set for use by the defender [40]). The proposed method does not make any such assumptions, does not require a clean (attack-free) validation set, and can be deployed to protect various types of classifiers.

Poisoned samples are generally *atypical* of the distribution of the class to which they are labeled. We thus apply mixture modeling [16], [36] to accurately explain the potentially multi-modal data and to capture poisoned samples within a subset of mixture components. We make the following observations. If the poisoned samples are typical of another class (different from the class to which they are labeled), we expect that re-distributing them to other classes should increase the overall data likelihood. Furthermore, removing a poisoned component will reduce the model complexity of a mixture. Thus, both the data likelihood and model complexity terms that constitute the Bayesian Information Criterion (BIC) model selection criterion [47] should improve when data poisoning is mitigated. Accordingly, we propose to make poisoned sample inferences consistent with minimizing BIC. We first apply mixture modelling separately to each class, with the number of components chosen to minimize the BIC criterion. Then we assess components for possible poisoning, with a detected component either removed or revised (whichever results in a lower BIC cost). After poisoned samples have been detected and removed, the classifier is trained on the sanitized data set.

In summary, our BIC-based mixture model defense is:

- *Novel*: We are the first to formulate a BIC-based defense for *unsupervised* DP attack mitigation.

- *Xi Li, David J. Miller, Zhen Xiang and George Kesidis are with the School of Electrical Engineering and Computer Science, Pennsylvania State University, State College, PA, 16803.*
  E-mail: {xzl45, djm25, zux49, gik2}@psu.edu

1. Error-specific attacks, particularly backdoor attacks involving specific backdoor patterns and source and target classes, *e.g.*, [8], [22], [34], are not the focus herein.

- *Practical*: We address the practical and challenging embedded DP attack scenario.
- *Effective*: Extensive experiments on benchmark datasets, for various classifier structures, demonstrate the effectiveness of our defense and its superiority over other defenses.

The rest of the paper is organized as follows: We first review several existing DP studies in Sec. 2. Then we define our threat model in Sec. 3. In Sec. 4, we propose our BIC-based mixture model defense. Experiments on two-class and multi-class tasks are presented in Sec. 5.2 and Sec. 5.3, respectively. Sec. 6 assesses computational complexity of our method. Sec. 7 identifies limitations and future work.

2 Related Work

A strategy for defending against data poisoning attacks is "data sanitization", *i.e.*, identifying and cleansing the attack samples as training set outliers. Outlier identification is divided into two types – supervised and unsupervised. Some supervised detection methods train a binary discriminator based on labeled examples of anomalies and normalities, *e.g.*, [55]. However, such a learned discriminator may only reliably identify *known* anomalies, similar to those seen during the discriminator's training. Furthermore, anomalies (attack instances) may be rarer and more difficult to collect than "normalities", resulting in class imbalance in training, thus negatively impacting the discriminator's performance.

Other supervised detection methods are more akin to *unsupervised* anomaly detection methods, except that they possess hyper-parameters whose setting requires either a clean validation set or a labeled set of "normalities" and "anomalies", *e.g.*, [6], [15], [25], [30], [44], [45], [49], [50], [52]. On the other hand, truly unsupervised detection methods do not require labeled examples of "normalities" and "anomalies", and are analogous to unsupervised clustering methods. They model data distributions and flag potential outliers, *e.g.*, [5], [33], [53], [57] and the method proposed herein. [5] proposed to leverage human intelligence to improve the accuracy of unsupervised detectors by correctly identifying outliers from machine-suggested candidates. However, such an approach is time-consuming, costly, and only suitable for domains where humans are skilled at analyzing data.

In practice there will always be *unknown* attacks, with no labeled examples available. It has been observed that the performance of supervised detectors may fare poorly on unknown attacks [37]. On the theoretical front, [50] generates an upper bound on the efficacy of any DP attack against a defender performing outlier removal and margin-based loss minimization. They also generate an attack that nearly achieves this upper bound. However, their attack requires full knowledge of the clean training set and cannot handle non-convex loss functions, which limits its application in practice. [45] proposes a method for producing certificates of robustness for two-layer neural networks. Such certificates are differentiable and can be jointly optimized with the network parameters, providing an adaptive regularizer that encourages robustness against attacks. However, they do not discuss the inherent tradeoff between robustness and model bias (*i.e.*, degradation in accuracy on *clean* data that results from making the classifier robust to attacks).

[44] relabels a sample based on the plurality label of its K nearest neighbors (KNN) to enforce label homogeneity. However, this defense will fail when the number of poisoned samples is sufficiently large such that some of the neighbors of an attack sample are also attack samples. This defense also relies on the availability of a clean validation set to tune the hyper-parameter K. This choice highly impacts the detector's performance, as will be seen (*cf*. Sec. 5.2 and 5.3). [6] introduces an attack-agnostic defense against DP attacks, employing a Generative Adversarial Network (GAN) for synthetic clean data generation based on the clean dataset possessed by the defender, on which a mimic model is trained. Samples with different predictions for the mimic model and the target model are deemed poisoned. Thus, [6] requires sufficient clean data to accurately train the GAN – but it is possible that such data could instead be used to train a clean classifier from scratch.

[15], [25] posited a unified view of effects of DP on learned classifier parameters: (1) the l_2 norm of the gradient from a poisoned sample is larger than that of a clean sample, on average; (2) there is an orientation difference between poisoned and clean sample gradients. [15] detects such effects by singular value decomposition (SVD) applied to the matrix of the sample-wise gradients of the training loss function with respect to the model parameters. They define an outlier score for each sample as the projection of its gradient onto the top right singular vector. At each detection step, the top $\frac{\epsilon}{\beta}$ fraction of samples with highest scores are removed, where β is the total number of detection steps and ϵ is the fraction of samples ultimately removed after β steps. The performance of the detector sensitively depends on the choice of the hyperparameters β and ϵ. Also, it is only applicable to linear classifiers. Furthermore, [15] is computationally expensive as it requires performing an SVD for each class, at each detection step, and retraining the classifier after each detection step.

[25] mitigates DP by gradient shaping (GS), *i.e.*, constraining the magnitude and orientation of poisoned gradients to make them close to clean gradients. For example, one can adopt, *e.g.*, a differentially-private stochastic gradient descent (DP-SGD) optimizer in training. It modifies gradients by clipping and adding noise, both controlled by hyperparameters. [25] is computationally cheap – it does not require extra computation pre-training/post-training. However, their method only reduces the effect of poisoning, rather than eliminating the poisoned samples. Efficacy of their defense is dramatically degraded as more and more attack samples are injected, as will be seen from our results.

[30], [52] mitigate DP attacks by aggregating multiple base classifiers trained on separate data subsets. The premise is that each data subset will have fewer poisoned samples, resulting in less classifier degradation. [30] partitions the training set into disjoint subsets using a hash function. To further improve performance, each base classifier is first trained on the whole training set by semi-supervised learning, focusing on predicting image rotation angles [20], then is fine-tuned on its dedicated partition through supervised learning.

[33] applied a BIC-based defense against DP attacks for binary classification tasks. The fundamental difference between [33] and our work pertains to *untainted data avail-*

ability. They assume the attacker only poisons one of the two classes, with this class known to the defender. Thus, the defender can always take the clean class as reference to identify poisoned samples in the corrupted class. However, in practice, the attacker is able to poison more than one class, and the defender does not know which class(es) are poisoned. Under this most realistic scenario, [33] may fail even if only one class is poisoned, as the defender might sanitize the clean class based on the poisoned one (*cf*. Sec. 5.2). *By contrast, our defense strategy applies to multi-class classifiers and addresses the practical and challenging DP attack scenario where the poisoned samples and the poisoned classes are a priori unknown. Also, we do not make any assumption of clean data availability.*

[6], [15], [25], [30], [44], [52] are supervised detection methods, with their performances highly impacted by the choices of hyper-parameters. By contrast, our method is unsupervised. At each optimization step, it separately assesses the hypothesis that each individual mixture component, in each class, is poisoned. Only the component whose trial-sanitization yields the lowest BIC cost is actually sanitized. This process is repeated until the total BIC cost, defined over all classes, converges.

3 Threat Model

We consider W-class ($W \geq 2$) tasks, where the classifier, denoted $f : \mathbb{R}^d \to \{1, \ldots, W\}$, is trained on $\mathcal{D}_{\text{Train}}$ and then tested on $\mathcal{D}_{\text{Test}}$, both with (assumed) i.i.d. samples. Each feature x_l, $l = 1, \ldots, d$, may be either discrete or continuous-valued.

We assume the attacker: 1) has sufficient knowledge of the classification domain to generate or acquire samples that are legitimate instances of the different classes, in order to launch label flipping poisoning attacks [3], [54]; 2) covertly inserts poisoned samples into the training set ($\mathcal{D}_{\text{Train}} = \mathcal{D}_{\text{Clean}} \cup \mathcal{D}_{\text{Attack}}$); 3) May simultaneously poison any subset of the classes, possibly with different numbers of poisoned samples for each class; 4) is unaware of any deployed defense. The *goal* of the attacker is to degrade the classifier's (test set) generalization accuracy as much as possible.

The defender: 1) only has the training set $\mathcal{D}_{\text{Train}}$ manipulated by the attacker, not any additional samples known to be clean (attack-free); 2) does not know whether an attack is present, and if so, does not know the subset of attacking samples ($\mathcal{D}_{\text{Attack}}$), nor which class(es) are corrupted. The defender *aims* to: 1) identify and remove as many poisoned samples and as few clean samples as possible, before classifier training/retraining, and in so doing: 2) achieve classification accuracy close to that of a classifier trained on clean (unpoisoned) data.

4 BIC Mixture-based Sanitization Strategy

First, we hypothesize that poisoned samples labeled to a particular class form different sub-populations from normal samples with the same label. Thus, we apply *mixture modeling* to accurately represent the dataset and to concentrate poisoned samples within several distinct components. **Second**, the likelihood of the whole dataset should increase if poisoned samples are re-assigned to their true classes, and the complexity of the mixture model should decrease if a component formed by poisoned samples is removed or revised. *We thus aim to identify poisoned samples, and to remove or revise poisoned components, such that the overall data likelihood increases and the model complexity decreases*, *i.e.* consistent with *minimizing the BIC* [47] objective function.

To accurately represent the possibly poisoned dataset, we first apply mixture modeling to each class. Mixture modeling is a sound statistical approach for well-fitting potentially multi-modal data [16], [36] and also gives the potential for concentrating the poisoned samples into just a few components, which assists in accurately identifying and removing them. In practice, poisoned components may own both poisoned and untainted samples, with the poisoning ratio for each component unknown.

After learning the initial class-specific mixtures, we propose to identify poisoned samples in the training set as those with greater *likelihood* under a different class than the class to which they are labeled. We effectively *re-assign* such samples to the class (and mixture component) under which they have the greatest likelihood. Then the parameters of the component that has samples re-assigned to another class are updated based on its remaining samples.

Now, suppose the vast majority of a mixture component's samples are re-assigned in this way to another class. In this case, there may be insufficient remaining samples to reliably (or even in a well-posed fashion[2]) estimate the component's parameters. Thus, rather than retaining this component, it may be better to *remove* it, with its remaining samples reassigned. Since these samples are not deemed poisoned, they are re-assigned to other components within the same class, those under which they have the greatest likelihood. To decide between revising and (wholesale) removing a mixture component from a class's model, we apply BIC, which expresses an inherent, fundamental trade-off between data likelihood fit and model complexity. A poisoned component is either removed or revised, based on whichever results in a lower BIC cost.

In short: *a component is identified as poisoned if removing or revising it and re-assigning its samples reduces the BIC cost; moreover, samples which are redistributed to other class(es) are deemed poisoned.* Thus, our anomaly detection method is *unsupervised* and consistent with solving a BIC minimization problem[3]. In the sequel, we develop an algorithm for mitigating data poisoning via (locally optimal) minimization of the BIC objective.

4.1 Bayesian Information Criterion

The BIC objective function for a given data set $\mathcal{D}$ is:

$$\text{BIC} = |\theta|k - L(\mathcal{D}; \theta), \tag{1}$$

where θ is the set of free parameters specifying a density function model for the data, $|\theta|$ denotes its size, k is the cost (penalty) for describing an individual model parameter, and $L(\mathcal{D}; \theta)$ is the log-likelihood of the data set $\mathcal{D}$, based on the density function model. In [47], under suitable assumptions,

2. For example, for a multivariate Gaussian component, with a full covariance matrix of size $d \times d$, one needs at least d samples to estimate a (full-rank) covariance matrix.

3. Apart from poisoned samples, our method might also remove any outliers, if it is BIC-efficacious to do so.

BIC is shown to be a consistent estimator of model order. In [29], within an approximate Bayesian setting, the BIC penalty is derived, and found to be $k = 0.5 * \log(|\mathcal{D}|)$. This model penalty will be used in the following.

The form of the BIC objective seen above is equivalent to the minimum description length (MDL) [46], and is amenable to interpretation as a two-part codelength: i) the first term, which we will denote by Ω in the sequel, is the number of bits needed to describe the model parameters; ii) the second, negative log-likelihood term is the number of bits to describe the data set, given the model.

In this work, we model the training data labeled to each class by a class-specific mixture of density functions (or probability mass functions in the case of discrete data), *i.e.*, for an individual sample $\boldsymbol{x}$, labeled to class c, its density (likelihood) is:

$$P[\boldsymbol{x};\theta_c] = \sum_{j=1}^{M_c} \alpha_j^c P[\boldsymbol{x};\Lambda_j^c], \tag{2}$$

where α_j^c is the probability mass of mixture component j for class c (*i.e.*, $\sum_{j=1}^{M_c} \alpha_j^c = 1$, $\alpha_j^c \geq 0\ \forall j$), $P[\cdot;\Lambda_j^c]$ is the j^{th} component density under class c, Λ_j^c is the set of parameters specifying the component density, and M_c is the number of mixture components for class c. Note that $\theta_c = \{\Lambda_j^c\} \bigcup \{\alpha_j^c\}$ and $\theta = \bigcup_c \theta_c$.

In Eq. 2, a data sample from class c is associated probabilistically with all mixture components from the class's mixture density. Alternatively, in this work, we consider the *complete data* BIC objective function, based on the complete data log-likelihood function [12], wherein each data sample is hard (fully) assigned to the mixture component under which it has the greatest log-likelihood. That is, the complete data log-likelihood for the data from class c is[4]

$$L_j^c = \sum_{\boldsymbol{x} \in \mathcal{X}_j^c} \log P[\boldsymbol{x};\Lambda_j^c],$$

where $\boldsymbol{x} \in \mathcal{X}_j^c$ if and only if, for $\boldsymbol{x}$ labeled to class c, $P[\boldsymbol{x};\Lambda_j^c] \geq P[\boldsymbol{x};\Lambda_{j'}^c]\ \forall j' \neq j$.

Likewise the complete data BIC objective is:

$$\text{BIC}_{\text{cmplt}} = |\theta|k - \sum_{c=1}^{W}\sum_{j=1}^{M_c} L_j^c. \tag{3}$$

4.2 BIC-based Defense

Let $T = |\mathcal{D}_{\text{Train}}|$ be the total number of training samples. Let $(\boldsymbol{x}_i, y_i) \in \mathbb{R}^d \times \{1,\ldots,W\}$ represent the feature vector (in the continuous-valued case) and label of training sample i. Denote Ω and L as the model complexity and data log-likelihood, respectively.

The model parameters θ_c of the mixture for class c are estimated via the Expectation-Maximization (EM) algorithm [12], applied to the subset of $\mathcal{D}_{\text{train}}$ labeled as class c. The chosen model order M_c is the one that yields the least BIC cost (1) over the set $\{1,\ldots,M_{\max}^c\}$ [47], with $M_{\max}^c$ an upper bound on the number of components in class c's mixture[5]. Finally, we let $\mathcal{S} = \{(c,j)|c = 1,\ldots,W, j = 1,\ldots,M_c\}$ be the set of components across all classes.

To reiterate, we identify poisoned components and samples by minimizing the complete data BIC cost (3). This BIC minimization involves sample re-distribution, component removal/revision, and parameter updates. To reflect these model changes, we introduce several types of "indicator" variables:

- The class t_i and component under this class j_i that best-explain sample $\boldsymbol{x}_i$ are $(t_i, j_i) = \arg\max_{t=\{1,\ldots,W\},\ j=\{1,\ldots,M^t\}} P[\boldsymbol{x}_i;\Lambda_j^t]$,
- $r_j^c = \begin{cases} 1 & \text{component } j \text{ in class } c \text{ is poisoned} \\ 0 & \text{else} \end{cases}$
- $q_j^c = \begin{cases} 1 & \text{component } j \text{ in class } c \text{ needs to be revised} \\ 0 & \text{component } j \text{ in class } c \text{ needs to be removed} \end{cases}$

Note that q_j^c is configured only when $r_j^c = 1$.

The complete data BIC cost to be minimized can then be expressed as:

$$\text{BIC}_{\text{cmplt}}(\theta) = \sum_{c=1}^{W}\sum_{j=1}^{M_c}((1 - r_j^c(1 - q_j^c))k|\Lambda_j^c| + 1 + \delta(r_j^c, 1))$$
$$-\sum_{c=1}^{W}\sum_{j=1}^{M_c}((1 - r_j^c)L_j^c(\Lambda_j^c) + r_j^c \sum_{\boldsymbol{x}_i \in \mathcal{X}_j^c} \log P[\boldsymbol{x}_i;\Lambda_{j_i}^{t_i}]). \tag{4}$$

In (4), the model parameters are $\theta = \{\{\Lambda_j^c\},\{r_j^c\},\{q_j^c\}\}$, where the structural parameters r_j^c and q_j^c each require one bit to specify (hence the '1' and $\delta(r_j^c, 1)$ contributions to the model complexity term). By contrast, t_i and j_i are hidden data assignments (as part of the complete data log-likelihood, and complete data BIC), not model parameters.

To minimize (4) in a locally optimal fashion, our approach cycles over the mixture components, one at a time, effecting changes to the mixture models that reduce this objective. The new BIC cost, in light of changes to component j from class c, can be expressed as the "old" BIC cost plus the (negative) change resulting from sample re-assignments or component removal/revision, denoted ΔBIC_j^c.

Each feasible joint configuration of the variables for component j in class c corresponds to one of three cases:

1) $r_j^c = 0$: The component is formed by clean samples, and there is no need to re-distribute its samples or modify the component (*i.e.*, $\Delta\Omega_{j,1}^c = 0$, $\Delta L_{j,1}^c = 0$). The change in BIC in this case is thus

$$\Delta\text{BIC}_j^c = \Delta\Omega_{j,1}^c + \Delta L_{j,1}^c = 0.$$

2) $r_j^c = 1, q_j^c = 0$: Component j is poisoned, and we are choosing to remove it from the mixture, changing the model complexity term by

$$\Delta\Omega_{j,2}^c = -|\Lambda_j^c|\frac{1}{2}\log T,$$

where $|\Lambda_j^c|$ is the number of parameters in component j from class c. The component's samples are re-distributed consistent with maximizing the log-likelihood: each sample $\boldsymbol{x}_i \in \mathcal{X}_j^c$ is re-assigned to component j_i of class t_i, where

$$(t_i, j_i) = \arg\max_{(t,j') \in \mathcal{S}\setminus\{(c,j)\}} \log P[\boldsymbol{x}_i;\Lambda_{j'}^t].$$

Let

$$\mathcal{Q} = \{(t_i, j_i)|\forall i,\ \boldsymbol{x}_i \in \mathcal{X}_j^c\}$$

4. Here we assume the component priors are uniform and hence they are absent from the complete data log likelihood. In practice, these terms do not affect detection performance significantly.

5. $M_{\max}^c$ is in fact not really a hyperparameter, as one can observe the changes of BIC to adjust the range of model orders. For example, if $M_{\max}^c$ yields the least BIC, one can increase $M_{\max}^c$ and repeat model selection until $M_c \neq M_{\max}^c$.

be the set of components which receive the re-assigned samples. For each component $(w, j') \in \mathcal{Q}$, we re-estimate its parameters on $\widehat{\mathcal{X}_{j'}^{w}}$ by maximum likelihood estimation (MLE):

$$\Lambda_{j'}^{w,\text{new}} = \arg\max_{\Lambda} \sum_{\boldsymbol{x}_i \in \widehat{\mathcal{X}_{j'}^{w}}} \log P[\boldsymbol{x}_i; \Lambda],$$

where

$$\widehat{\mathcal{X}_{j'}^{w}} = \mathcal{X}_{j'}^{w} \cup \{\boldsymbol{x}_i \in \mathcal{X}_j^c | t_i = w, j_i = j'\}.$$

This optimization has a closed form, globally optimal solution for the component density model forms considered in this paper. The total data log-likelihood changes by

$$\Delta L_{j,2}^{c} = -\sum_{(w,j')\in\mathcal{Q}} \sum_{\boldsymbol{x}_i \in \widehat{\mathcal{X}_{j'}^{w}}} \log P[\boldsymbol{x}_i; \Lambda_{j'}^{w,\text{new}}] + \sum_{(w,j')\in\mathcal{Q}} \sum_{\boldsymbol{x}_i \in \mathcal{X}_{j'}^{w}} \log P[\boldsymbol{x}_i; \Lambda_{j'}^{w}] + \sum_{\boldsymbol{x}_i \in \mathcal{X}_j^c} \log P[\boldsymbol{x}_i; \Lambda_j^c].$$

The change in BIC in this case is

$$\Delta\text{BIC}_j^c = \Delta\Omega_{j,2}^c + \Delta L_{j,2}^c.$$

3) $r_j^c = 1, q_j^c = 1$: Similar to case (2) but instead of removing it, we re-estimate the parameters of component j by its surviving samples (*i.e.*, samples with $t_i = c$). Revising a component does not change the model complexity cost, since the code length is untouched (*i.e.*, $\Delta\Omega_{j,3}^c = 0$). The parameters Λ_j^c are re-estimated by MLE on the surviving samples:

$$\Lambda_j^{c,new} = \arg\max_{\Lambda} \sum_{\boldsymbol{x}_i \in \widehat{\mathcal{X}_j^c}} \log P[\boldsymbol{x}_i; \Lambda],$$

where

$$\widehat{\mathcal{X}_j^c} = \{\boldsymbol{x}_i \in \mathcal{X}_j^c | t_i = c\}.$$

Samples that are best represented by class $w \neq c$ (*i.e.*, $t_i = w$, $w \neq c$) are re-distributed to their fittest components in class w, but the remaining samples (with $t_i = c$) are explained by the updated component j. Let

$$\mathcal{Q}' = \{(w, j') \in \mathcal{Q} | w \neq c\} \cup \{(c, j)\}$$

be the set of components to be updated. The total data log-likelihood changes by

$$\Delta L_{j,3}^{c} = -\sum_{(w,j')\in\mathcal{Q}'} \sum_{\boldsymbol{x}_i \in \widehat{\mathcal{X}_{j'}^{w}}} \log P[\boldsymbol{x}_i; \Lambda_{j'}^{w,\text{new}}] + \sum_{(w,j')\in\mathcal{Q}'} \sum_{\boldsymbol{x}_i \in \mathcal{X}_{j'}^{w}} \log P[\boldsymbol{x}_i; \Lambda_{j'}^{w}],$$

where $\widehat{\mathcal{X}_{j'}^{w}}$ and $\Lambda_{j'}^{w,\text{new}}$ $\forall (w, j') \in \mathcal{Q}' \setminus \{(c, j)\}$ are defined in the same way as in case 2. The BIC change in this case is

$$\Delta\text{BIC}_j^c = \Delta L_{j,3}^c.$$

To minimize the complete data BIC objective, for each component in class $c \in \{1, \ldots, W\}$, we should choose the configuration of the parameters that reduces BIC the most. However, the optimal configuration for any component j depends on the configurations for other components. It is thus intractable to define an algorithm guaranteed to find a globally optimal configuration over all components (*e.g.*, by exhaustively evaluating over the huge combinatorial space of component configurations). Instead, at each optimization step, we separately *trial*-update each component's configuration, and then only permanently update for the component that yields the greatest reduction in BIC. This is repeated until there are no further changes. This optimization approach is non-increasing in the BIC objective and results in a locally optimal solution.

The null hypothesis of our detection inference is that the training set is not poisoned (and is generated according to the existing mixture model). If there is data poisoning, the training set is hypothesized to be generated by an alternative model (with some components removed and/or modified). Thus, we perform the following hypothesis testing: after BIC minimization, if $r_j^c = 0$ holds for all components in all classes, and $t_i = y_i$ holds for all samples, then no components and samples are inferred to be poisoned, and we accept the null hypothesis. Otherwise, we reject the null hypothesis, and the training set is deemed poisoned. The samples that were re-assigned to other classes via the BIC minimization are deemed poisoned, and are removed from the training set.

4.3 Implementation

Consistent with the above description, we apply an iterative, locally optimal approach to minimize the total BIC cost and optimize its parameters, as shown in Algorithm 1. As we discussed before, at each algorithm step, we first evaluate the reduction in the total BIC cost ΔBIC_j^c caused by trial removal/revision of each component j in each class c. Then, we sanitize the component j^* in class c^* which decreases the total BIC cost the most, *i.e.*,

$$(c^*, j^*) = \arg\min_{c\in\{1,\ldots,W\}, j=1,\ldots,M^c} \Delta\text{BIC}_j^c,$$

where

$$\Delta\text{BIC}_j^c = \min_{m=1,2,3} \{\Delta\Omega_{j,m}^c + \Delta L_{j,m}^c\}.$$

This is repeated until no trial component updates further reduce the BIC cost. Finally, all samples with $t_i \neq y_i$ are removed from the training set, and we have the sanitized training set

$$\widehat{\mathcal{D}}_{\text{Train}} = \{\boldsymbol{x}_i \in \mathcal{D}_{\text{Train}} | t_i = y_i\}.$$

Note that: 1) the same component may be re-optimized multiple times during the course of this algorithm; 2) But removal of a component is permanent, *i.e.* once removed, a component cannot be reinstated.

5 Experiments

5.1 Experiment Setup

Dataset and mixture model: For binary classification, we use the TREC 2005 spam corpus (**TREC05**) [10]. TREC05 contains 39,999 real ham and 52,790 spam emails which are labeled based on the sender/receiver relationship. We apply pre-processing techniques such as normalization, stop word removal, stemming, and low-frequency word filtering to the corpus. For multi-class ($W > 2$) classification, we use the 20-Newsgroups dataset (**20NG**) [28], **MNIST** [14], **CIFAR10** [26], and **STL10** [9]. 20NG collects news documents from 20 topics, with each topic containing around

Algorithm 1: BIC-Based DP Attack Defense

```
Input  : $\mathcal{D}_{\text{Train}} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}$, $\{\Lambda_j^c\}_{j=1,\ldots,M^c,\ c\in\{1,\ldots,W\}}$
Output: $\widehat{\mathcal{D}}_{\text{Train}}$
$r_j^c = 0$, $q_j^c = 0$, $\forall c, j$ ;
$t_i = y_i$, $\forall i$;
$\Delta\text{BIC}_j^c = 0, \forall j, c$;
do
    for $c \in \{1, \ldots, W\}$ do
        for each component $j$ in class $c$ do
            compute BIC reduction from $j$
              $\Delta\text{BIC}_j^c = \min\{\Delta\Omega_{j,m}^c + \Delta L_{j,m}^c\}_{m=1}^{3}$;
            configure $\{t_i | \forall \boldsymbol{x}_i \in \mathcal{X}_j^c\}$, $r_j^c$, $q_j^c$ consistent
              with $\Delta\text{BIC}_j^c$;
    $(c^*, j^*) = \arg\min_{c\in\{1,\ldots,W\}, j=1,\ldots,M^c} \Delta\text{BIC}_j^c$;
    if $r_{j^*}^{c^*} = 1$ then
        For $\boldsymbol{x}_i \in \mathcal{X}_{j^*}^{c^*}$, if $t_i \neq c^*$, re-distribute $\boldsymbol{x}_i$ to
          component $m = \arg\max_{m'} \log P[\boldsymbol{x}_i; \Lambda_{m'}^{t_i}]$ in class
          $t_i$ and then update component $m'$s parameters
          via MLE;
        if $q_{j^*}^{c^*} = 0$ then
            remove component $j^*$ from $\{\Lambda_j^{c^*}\}_{j=1,\ldots,M^{c^*}}$;
            re-distribute each $\boldsymbol{x}_i \in \mathcal{X}_{j^*}^{c^*}$ to component
              $m = \arg\max_{m'} P[\boldsymbol{x}_i; \Lambda_{m'}^{c^*}]$ and update
              component $m'$s parameters;
        else
            update component $j^*$'s parameters on $\mathcal{X}_{j^*}^{c^*}$;
while $\sum_{c,j} \Delta\text{BIC}_j^c < 0$;
$\widehat{\mathcal{D}}_{\text{Train}} = \{\boldsymbol{x}_i \in \mathcal{D}_{\text{Train}} | t_i = y_i\}$;
```

1000 samples. MNIST is a dataset of 28x28 gray-scale images. It contains 6000 images per class. CIFAR10 consists of 32x32 color images, with 6000 images per class. STL10 is composed of 96x96 color images, with 1300 labeled images per class[6]. The initial experiments on each dataset involved 5 classes. For 20NG, we chose classes "rec.sport.baseball", "soc.religion.christian", "comp.graphics", "rec.autos", and "misc.forsale". For MNIST, CIFAR10, and STL10, we chose the first 5 classes.

We split each dataset into disjoint subsets as follows: (1) For TREC05, we randomly select 9000 ham emails and 9000 spam emails[7] for training, and randomly select 3000 ham and 3000 spam emails to form the test set. The remaining samples are used for poisoning. (2) For 20NG, for each class, we randomly choose 600 samples for training, 220 for testing, and 160 for poisoning. (3) For MNIST, we randomly select 2000 images per class for training, 1000 per class for testing, and 800 per class for poisoning. (4) For CIFAR10, for each class, 4000 images are used for training, 1000 for testing, and 800 for poisoning. (5) For STL10, for each class, 500 images are used for training, 700 for testing and 100 for poisoning.

For each dataset, we train a mixture model for each of its classes. We apply Parsimonious Mixture Modeling (**PMM**) [21] on TREC05 and 20NG. PMMs allow parameter sharing across multiple components, which greatly reduces the number of model parameters compared with standard (unstructured) mixtures, and which allows BIC to choose good model orders in high feature dimensions, rather than grossly underestimating the model order [21]. Note that for PMMs, due to parameter sharing, the number of parameters $|\Lambda_j^c|$, specifying a component density function, is not necessarily the same, across all components, from all classes. PMMs are learned through the generalized expectation-maximization (GEM) framework [21]. After pre-processing, the dictionary of TREC05 and 20NG contains around 30000 and 10000 unique words, respectively. For both datasets, the training and test samples are represented using a bag-of-words. Each PMM component is a multinomial joint probability mass function. Fig. 1 shows that, for class "soc.religion.christian" of the clean 20-Newsgroups dataset, PMM chooses a reasonable model order (14) that minimizes the training set BIC cost and also has good generalization (*i.e.*, test set log-likelihood). Initially we chose $M_{\max}^c = 25, \forall c$. If the chosen model order $M_c = M_{\max}^c$, then $M_{\max}^c$ is increased and the model is retrained.

We apply Gaussian mixture models (**GMM**) on MNIST, CIFAR10, and STL10. For MNIST, we train GMMs on the image features (flattened as 784-dimension vectors), while for CIFAR10 and STL10 we train GMMs on the 512-dimensional penultimate layer features of the victim NN-based classifier, as the raw images are not very suitable for clustering. Before GMM training, we normalize the feature values to $[0, 1]$ and center the feature vectors. To reduce model complexity, we used a diagonal covariance matrix for each Gaussian component. The GMMs are learned through the Expectation-Maximization (EM) framework [12].

DP attack and target classifiers: We primarily considered label-flipping attacks, randomly mislabeling a sample from class c to a class $w \neq c$. The attack strength – the number of poisoned samples – may not be the same for each class. The poisoned samples are randomly selected (from the sample pool left over from the training and test sets).

We launched 12 poisoning attacks on TREC05. For half of the attacks, we only poisoned one class (*e.g.*, spam), with attack strength from 1000 to 6000 samples. For the other half of the attacks, we simultaneously poisoned the ham and spam training sets with various attack strengths (*cf*. Tab. 1).

For the multi-class tasks, we launched 5 DP attacks on all datasets. In attack $i = 1, \ldots, 5$, we used samples of the first i classes for poisoning. Each poisoned sample from a given class c is randomly mislabeled to one of the other classes.

We chose linear **SVM** [11], **LR** [39], and bi-directional one-layer long short-term memory (**LSTM**) [24] recurrent neural networks with 128 hidden units as the target classifiers for TREC05 and 20NG, since they are effective in the classification of text data. For image datasets, we chose ResNets [23] as the target classifiers[8]. For MNIST, we also consider linear SVM and LR.

Evaluation criteria: The performance criteria are: 1) improvement in test classification accuracy (**ACC**) after data sanitization; 2) true positive rate (**TPR**) — the fraction of

6. The STL10 dataset is an image recognition dataset with 100000 unlabeled images for developing unsupervised feature learning, deep learning, self-taught learning algorithms. Here we only use the labeled set.

7. There are 8651 ham emails and 8835 spam emails remaining in the training set after pre-processing. Some emails are removed since there are no tokens left after *e.g.*, stop word removal and low-frequency word filtering.

8. Since the number of labeled training samples of STL10 is not sufficient for training a complicated NN from scratch, we fine-tuned a pre-trained ResNet-34 on STL10.

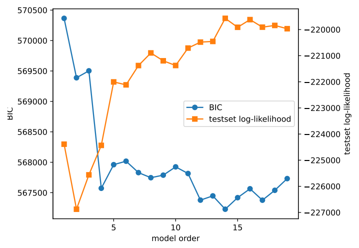


Fig. 1: Training set BIC cost and test set log-likelihood versus the model order of the PMM on class "soc.religion.christian" of clean 20NG.

poisoned samples that are detected; and 3) false positive rate (**FPR**) – the fraction of non-poisoned samples falsely detected.

Hyperparameter setting: We apply the BIC-based defense with clean data samples (**BIC-C-D**) [33], the KNN-based defense (**KNN-D**) [44], the GS-based defense (**GS-D**), and the SVD-based defense (**SVD-D**) [15] on the same poisoned training sets. As [33] was only proposed for binary classification (and assumes the poisoned class is known), we do not apply this method on the multi-class classification task. We fail to apply the SVD-D on TREC05, since it is too expensive to perform SVD on the matrix of gradients, whose size is around (20000×60000). We also do not apply it on the LSTM classifier, since it is only applicable to linear classifiers. For ResNet models, we apply SVD-D on the (linear) output layer. We also apply **DPA** [30] with semi-supervised learning and **FA** [52] on the image datasets and the ResNet architecture, as the code provided by [30] was designed specifically for image datasets. We set the hyperparameters of these detection methods as follows: 1) For KNN-D, we set the number of neighbors at $K = 10$, which is suggested by [44]. 2) For GS-D, we applied a DP-SGD optimizer with clip-norm of 2.0 and noise multiplier of 0.1 for training, which were suggested in [25]. 3) For SVD-D, we set the number of detection steps at $\beta = 2$, which is the same as in [15]. To show the best performance for that method, we set ϵ to the real poisoning ratio if the training set is poisoned, and set it to 0.01 if there is no poisoning. 4) For DPA and FA, we set the number of partitions as $L = 5$ to limit the computational cost. Since the code of FA is not available, we simply fine-tune the lth base classifier $(l = 1, \ldots, L)$ on partition l and $(l + 1) \mod L$.

5.2 Experimental Results on Binary Classification

Tab. 1 shows the performance of victim classifiers as a function of attack strength on poisoned and sanitized TREC05 datasets. We first trained the target classifiers on the clean dataset (Attack (0,0) in Tab. 1), yielding clean ACC (baselines) for SVM, LR, and LSTM of 0.9522, 0.9616, and 0.9632, respectively. As the total number of poisoned samples is increased to 6000, the classification accuracies of SVM/LR drop below 0.75 and LSTM drops to 0.8. Thus, embedding real ham into the spam set and real spam into the ham set is indeed a significant poisoning attack on SVM/LR/LSTMs.

Then, we apply BIC-D, KNN-D, GS-D, and BIC-C-D on the corrupted training sets and retrain the victim classifiers on the sanitized datasets. Since BIC-C-D [33] unrealistically assumes the defender knows which class is clean, we alternately apply it on ham and spam until the total BIC cost converges. The sanitization is always initiated from class ham. As expected, the ACC with BIC-C-D drops rapidly when the poisoned samples exceed 4000. The ACC with KNN-D and GS-D also decline gradually as the attack strength increases, while BIC-D performs well and stably. In all cases, our defense surpasses the other three defenses in classification accuracy (marked in bold). When the attack is strengthened to 5000 mislabeled ham emails, KNN-D exhibits little improvement in ACC, and GS-D performs even worse than the poisoned classifiers. Our defense significantly improves ACC even under strong attacks, restoring it close to the clean baselines. For LSTM, GS-D performs even worse than the poisoned classifiers in all cases, which may be attributable to hyperparameter settings (We followed [25], which set hyperparameters for LR, not LSTM). Note that for the embedded data poisoning scenario considered here, there is no clean validation set available for hyperparameter tuning.

Tab. 2 shows the TPRs and FPRs of the detection methods (GS-D is not included since it does not identify the poisoned samples). Compared with KNN-D and BIC-C-D, our defense has relatively low FPRs for all cases. Also, our defense gives higher TPRs than KNN-D when only the spam set is poisoned and comparable TPRs when both of the classes are poisoned. When the poisoned samples exceed 4000, BIC-C-D fails to detect poisoned samples and falsely removes a large number of clean samples.

When there is no attack, our defense falsely detects 789 clean samples as poisoned, but these samples have similar likelihoods under both classes[9]. Also, for an SVM trained on the clean training set excluding these samples, the ACC on these removed samples is only 0.5855 – these samples are close to the decision boundary and fit both classes. Given the similar likelihoods, it is BIC-efficacious to remove/revise the components formed by these samples. Besides, removing these ambiguous samples in fact slightly *increases* the ACC.

Fig. 2a and 2b shows how the total BIC cost, SVM test accuracy, and the number of detected poisoned samples change with the number of components removed/revised under the attack with 3000 poisoned samples for TREC05[10]. As emphasized in Sec. 4, our method guarantees strict descent in the BIC objective. But we cannot guarantee that the test accuracy or the number of detected poisoned samples are strictly non-decreasing. Samples that were previously deemed poisoned might be restored clean at subsequent detection steps and vice versa, resulting in slight fluctuation in the curves of SVM ACC and detected poisoned samples. Overall, the strong trend of the two curves is an increase with increasing detection steps – the two curves increase sharply initially and converge in the final stages, as the BIC cost is further decreased.

We show the number of components, revised components and removed components for both ham and spam under all attack cases in Tab. 3. In general, the number of removed and revised components increases as the attack is strengthened. But for most cases, our method prefers revis-

9. The average log likelihoods under ham and spam are -879.77 and -852.88, respectively.

10. In practice, we only remove the detected poisoned samples and retrain the classifier once the detection procedure terminates.

# Poisoned Ham, # Poisoned Spam	0, 0	0, 1000	0, 2000	0, 3000	0, 4000	0, 5000	0, 6000	1000, 1000	1000, 2000	2000, 1000	2000, 2000	2000, 4000	4000, 2000
SVM													
Poisoned	0.9522	0.8867	0.8461	0.8215	0.7932	0.7731	0.7495	0.8339	0.7924	0.7833	0.7488	0.7142	0.7114
BIC-D	**0.9684**	**0.9611**	**0.9530**	**0.9425**	**0.9411**	**0.9394**	**0.9329**	**0.9454**	**0.9284**	**0.9429**	**0.9143**	**0.8998**	**0.8731**
KNN-D	0.9001	0.8974	0.8828	0.8660	0.8358	0.7958	0.7751	0.9049	0.8917	0.8880	0.8793	0.8421	0.8367
GS-D	0.9645	0.9372	0.9225	0.9023	0.8131	0.7042	0.6314	0.9129	0.8807	0.8738	0.8568	0.8159	0.7711
BIC-C-D	0.9579	0.9434	0.9124	0.8519	0.6882	0.6039	0.5697	0.9217	0.9088	0.9061	0.8288	0.6385	0.7153
LR													
Poisoned	0.9616	0.9175	0.8828	0.8443	0.8172	0.7803	0.7488	0.8843	0.8501	0.8481	0.8183	0.7591	0.7438
BIC-D	**0.9699**	**0.9660**	**0.9559**	**0.9519**	**0.9461**	**0.9394**	**0.9368**	**0.9511**	**0.9402**	**0.9507**	**0.9315**	**0.9126**	**0.8799**
KNN-D	0.9099	0.9073	0.8955	0.8831	0.8529	0.8069	0.7776	0.9169	0.9039	0.9044	0.8968	0.8646	0.8656
GS-D	0.9598	0.9384	0.9184	0.8606	0.8158	0.7099	0.6655	0.9258	0.9137	0.8948	0.8797	0.8091	0.7847
BIC-C-D	0.9622	0.9554	0.9247	0.8633	0.6909	0.6192	0.5801	0.9375	0.9222	0.9152	0.8358	0.6427	0.7158
LSTM													
Poisoned	0.9632	0.9363	0.9111	0.8852	0.8668	0.8159	0.8028	0.8788	0.8681	0.8691	0.8521	0.7738	0.7985
BIC-D	**0.9701**	**0.9682**	**0.9619**	**0.9588**	**0.9513**	**0.9465**	**0.9424**	**0.9584**	**0.9476**	**0.9551**	**0.9431**	**0.9223**	**0.8991**
KNN-D	0.9313	0.9281	0.9183	0.8941	0.8744	0.8449	0.8009	0.9317	0.9131	0.9013	0.9125	0.8905	0.8821
GS-D	0.8339	0.8205	0.8123	0.7792	0.7347	0.7153	0.6824	0.8383	0.8176	0.8198	0.8208	0.7718	0.7949
BIC-C-D	0.9629	0.9607	0.9217	0.8712	0.6915	0.6149	0.5906	0.9359	0.9232	0.9277	0.8404	0.6514	0.7368

TABLE 1: Classification ACC of victim classifiers on poisoned and sanitized TREC05.

# Poisoned Ham, # Poisoned Spam	0,0	0, 1000	0, 2000	0, 3000	0, 4000	0, 5000	0, 6000	1000, 1000	1000, 2000	2000, 1000	2000, 2000	2000, 4000	4000, 2000
True Positive Rates (TPRs)													
BIC-D	-	**0.8898**	**0.9044**	**0.9036**	**0.8689**	**0.9014**	**0.8865**	0.8633	**0.8678**	0.8874	0.8351	0.8113	0.8142
KNN-D	-	0.8393	0.8154	0.7856	0.7342	0.6478	0.5761	**0.8996**	0.8518	**0.9082**	**0.8842**	**0.8362**	**0.8261**
BIC-C-D	-	0.8846	0.8340	0.7303	0.3644	0.1951	0.1122	0.8628	0.8420	0.8284	0.7446	0.2102	0.4390
False Positive Rates (FPRs)													
BIC-D	**0.0177**	**0.0249**	**0.0841**	**0.0877**	**0.0553**	**0.0885**	**0.0652**	**0.0499**	**0.0629**	**0.0586**	**0.0737**	**0.0809**	**0.1131**
KNN-D	0.0745	0.0826	0.0936	0.1095	0.1377	0.1798	0.2122	0.0888	0.1057	0.1012	0.1099	0.1339	0.1452
BIC-C-D	0.0505	0.0724	0.0775	0.0881	0.3209	0.3621	0.3868	0.0598	0.0681	0.0630	0.2128	0.2958	0.2698

TABLE 2: TPRs and FPRs of three defenses on TREC05.

# Poisoned Ham, # Poisoned Spam	0,0	0, 1000	0, 2000	0, 3000	0, 4000	0, 5000	0, 6000	1000, 1000	1000, 2000	2000, 1000	2000, 2000	2000, 4000	4000, 2000
# Components	(21,18)	(29,16)	(22,18)	(25,17)	(19,20)	(24,20)	(24,31)	(49,27)	(25,15)	(37,29)	(48,28)	(40,29)	(36,28)
# Revised Components	(1,5)	(0,6)	(6,11)	(5,10)	(1,16)	(2,9)	(7,11)	(19,18)	(11,7)	(17,12)	(9,7)	(14,11)	(14,13)
# Removed Components	(0,1)	(5,3)	(2,6)	(1,2)	(2,4)	(3,4)	(4,11)	(7,4)	(4,2)	(4,6)	(12,5)	(10,5)	(8,11)

TABLE 3: The number of components, revised components, and removed components of each class of TREC05.

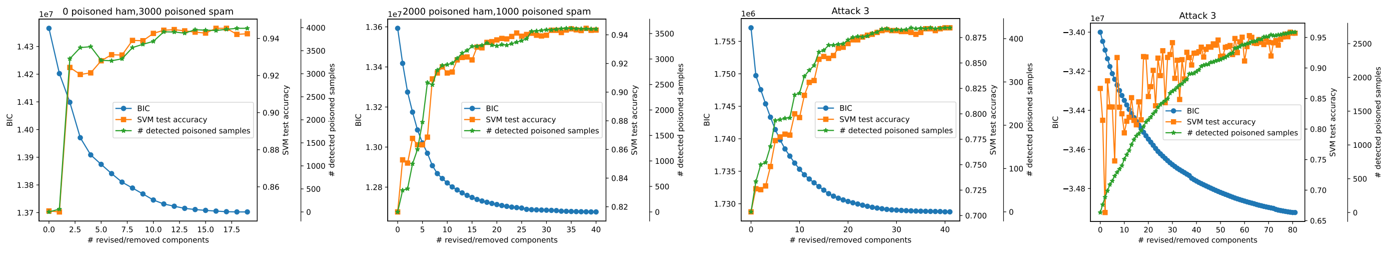
ing a poisoned component rather than removing it (most components consist of both clean and poisoned samples, and it is apparently most BIC-efficacious to revise them).

5.3 Experimental Results on Multi-class Classification

Table 4, 6, 8, and 10 show the test accuracy of victim classifiers as a function of the number of attacked classes on poisoned and sanitized 20NG, MNIST, CIFAR10, and STL10 datasets, respectively. We first trained the target classifiers on the attack-free datasets to get baseline test accuracies (*i.e.*, column 0 of poisoned classifiers). Then we trained the classifiers on training sets poisoned by the 5 DP attacks described in Sec. 5.1, with the resulting test accuracies in columns 1-5 of “Poisoned”, respectively. For 20NG and MNIST, as the number of classes used for poisoning is increased to 5, the classification accuracies of SVM and LR drop by over 30% (absolute percentage drop). The test accuracy of the NN-based classifier drops by nearly 30% on 20NG and 20% on MNIST. The test accuracies of ResNets drop by over 10% on CIFAR10 and nearly 10% on STL10. Thus, the attacks designed in Sec. 5.1 are indeed effective poisoning attacks against all target classifiers, on all datasets.

Then we applied our defense and KNN-D, GS-D, SVD-D, DPA, and FA on the poisoned training sets. Generally, our defense outperforms the other defenses in ACC (marked in bold). For 20NG (*cf*., Tab. 4), our defense performs the best in all attacking cases. Although other methods improve the ACC for most cases, they incur a 10%-25% drop in ACC as the attack strength increases, compared with the clean baseline. Our BIC-D performs well and stably – the ACC drops by 5% at most.

For CIFAR10 (*cf*., Tab. 8), our defense still beats the other defenses in all attacking cases, excluding attack 0 (attack-free), where SVD-D performs the best due to the proper setting of ϵ. However, in practice, it will be difficult to find an appropriate value for ϵ without a clean validation set. The ResNet-18 with KNN-D and GS-D performs even worse than the poisoned classifier in all attacking cases. SVD-D only improves the ACC by 4% at most. DPA struggles under weak attacks due to fine-tuning base classifiers on small partitions which are individually inadequate for learning an accurate DNN. However, it offers a slight improvement in ACC (around 3%) under attack 4 and 5, due to the inherent robustness of ensemble models. FA improves the test accuracy by around 2% in all attacking cases. By contrast, our method only incurs a drop of 2% in ACC under the



(a) (b) (c) (d)

Fig. 2: BIC cost, SVM ACC, and the number of detected poisoned samples versus the number of visited components under attacks: (a) TREC05 with 0 poisoned ham, 3000 poisoned spam; (b) TREC05 with 2000 poisoned ham, 1000 poisoned spam; (c) attack 3 against 20NG; (d) attack 3 against MNIST.

strongest attack, compared with the clean baseline.

For STL10 (*cf*., Tab. 10), the ResNet-34 with KNN-D performs worse than the poisoned classifier in all attacking cases. GS-D mitigates the negative effect of DP, but the ACC still drops by 6% compared with the clean baseline. SVD-D has little effect in improving ACC. On the other hand, the ACCs with our method drop by 1.31% at most, compared with the clean baseline. Similar to the results on CIFAR-10, DPA and FA perform worse than the single poisoned classifier.

For MNIST, our method outperforms GS-D, SVD-D, DPA, and FA, while KNN-D with $K = 10$ gives slightly better results than ours. $K = 10$ was suggested by [44] and chosen based on a clean validation set. However, without access to a trusted validation set under the embedded attack scenario considered here, the choice of K becomes crucial, as discussed in Sec. 2. For $K = 3$, the ACC with KNN-D drops significantly under attack 5. On the attack-free training set, SVD-D performs best on SVM and LR, and KNN-D is best for ResNet-18, given well-chosen hyperparameters (but BIC-D performs comparably). Unlike the results on CIFAR-10, the ensemble classifiers of DPA and FA maintain an ACC close to the clean baseline, but still lower than KNN-10-D and ours under all attack cases.

Tab. 5, 7, 9, and 11 show the TPRs and FPRs of the detection methods on 20NG, MNIST, CIFAR10, and STL10 datasets, respectively. Since the performance of SVD-D depends on the classifier architecture and training loss function (it evaluates the gradients of same), we respectively show its TPR/FPR on SVM, LR, and ResNet as SVD-D-S, SVD-D-L, and SVD-D-R. We reiterate here that both KNN-D and SVD-D evaluated here are supervised methods, *i.e.* with appropriately chosen hyper-parameters. For 20NG, CIFAR10, and STL10, compared with the other two defenses, our defense has relatively high TPRs and low FPRs for all cases. Almost no clean samples are falsely reported by our defense, while a large number of poisoned samples are correctly identified. KNN-D falsely detects lots of clean samples in all attack cases, even when there is no poisoning. By contrast, SVD-D only detects a small amount of poisoned samples, especially when the attack is weak. For MNIST, SVD-D has lower TPRs and FPRs than ours on SVM and LR, but does not perform well on ResNet. KNN-D with $K = 10$ has higher TPRs and lower FPRs than our method under most attacking cases. Again, the performance of KNN-D is affected by the choice of K, and $K = 10$ was chosen based on the detector's performance evaluated on a clean validation set for MNIST. With $K = 3$, the detector is less "aggressive" – it reports fewer detected poisoned images and has much lower TPRs.

On CIFAR-10, we also applied a GAN-based anomaly detector (**GAN-AD**) [56]. We report the ACC after sanitization and TPR at FPR at 0.1, as GAN-AD needs a detection threshold specified. As shown in Tab. 8 and 9, the improvement on test set ACC brought by GAN-AD is limited, due to the relatively low TPR. [51] suggests to apply the KNN based detectors on feature representations extracted by self-contrastive (SC) learning [7]. We report the ACC, TPR, and FPR of both KNN-D and BIC-D on the SC-learned feature representations in Tab. 12 (denoted as **KNNSC-D** and **BICSC-D** respectively). Recall that, for our main experiments on image datasets, we apply both methods on features extracted by the poisoned models, which yielded lower FPRs under weak attacks. On the other hand, the SC-based encoder improves TPRs and FPRs (under strong attacks) for both methods, as it removes the impact of mislabeling on feature representations. Despite that, training an additional encoder on the poisoned training set significantly increases the time complexity.

Similar to the results in Sec. 5.2, it is BIC-efficacious to re-distribute samples that are well-explained by more than one class, resulting in removal of a few samples from clean datasets. In Fig. 2c and 2d, we respectively show how the total BIC cost, SVM ACC, and the number of detected poisoned samples change with the number of components removed/revised under attack 3 against 20NG and MNIST. Again, our method strictly reduces the BIC objective, but cannot guarantee strict increases in ACC and the number of detected poisoned samples. For 20NG, both metrics are almost strictly non-decreasing, especially initially. For MNIST, ACC fluctuates heavily at first and finally converges at around 0.95, while detected poisoned samples steadily increases.

We show the number of total components, revised and removed components of each class under all attack cases on all datasets in Table 13. For all datasets, the total number of removed and revised components is nearly strictly increasing as the attack is strengthened. On CIFAR10 and STL10, BIC-D is applied in the internal layer feature space of ResNets, where the poisoned samples are well separated from clean ones; thus it prefers removing rather than revising poisoned components. By contrast, for 20NG, and

Attack	0	1	2	3	4	5
			SVM			
Poisoned	0.9172	0.8681	0.7936	0.7036	0.6072	0.5281
BIC-D	**0.9254**	**0.9127**	**0.9027**	**0.8845**	**0.8818**	**0.8736**
KNN-D	0.9009	0.8891	0.8672	0.8236	0.7736	0.7391
GS-D	0.9127	0.8918	0.8691	0.8391	0.8073	0.7645
SVD-D	0.9181	0.8663	0.8272	0.7745	0.7254	0.6600
			LR			
Poisoned	0.9309	0.8781	0.8291	0.7536	0.6718	0.5909
BIC-D	**0.9354**	**0.9218**	**0.9172**	**0.8954**	**0.8872**	**0.8809**
KNN-D	0.8909	0.8791	0.8681	0.8309	0.7963	0.7700
GS-D	0.9181	0.8873	0.8855	0.8536	0.8373	0.7973
SVD-D	0.9309	0.9081	0.8690	0.8618	0.8463	0.8400
			LSTM			
Poisoned	0.8063	0.7427	0.7163	0.6736	0.6055	0.5336
BIC-D	**0.8073**	**0.8064**	**0.8018**	**0.7800**	**0.7627**	**0.7481**
KNN-D	0.7454	0.7409	0.7200	0.7136	0.7000	0.6664
GS-D	0.2636	0.2555	0.2400	0.2282	0.2545	0.2536

TABLE 4: Classification ACC of victim classifiers on poisoned and sanitized 20NG.

Attack	0	1	2	3	4	5
			True Positive Rates (TPRs)			
BIC-D	-	**0.8325**	**0.8203**	**0.7958**	**0.7843**	**0.7762**
KNN-D	-	0.7687	0.7412	0.7296	0.7281	0.7104
SVD-D-S	-	0.2500	0.4156	0.4708	0.4828	0.4900
SVD-D-L	-	0.5937	0.6218	0.7020	0.7562	0.7650
			False Positive Rates (FPRs)			
BIC-D	**0.0084**	**0.0145**	**0.0135**	**0.0145**	**0.0168**	**0.0155**
KNN-D	0.1993	0.2001	0.1991	0.1997	0.2017	0.2091
SVD-D-S	0.01	0.0405	0.0632	0.0858	0.1118	0.1378
SVD-D-L	0.01	0.0219	0.0408	0.0483	0.0527	0.0581

TABLE 5: TPRs and FPRs of three defenses on 20NG.

Attack	0	1	2	3	4	5
			SVM			
Poisoned	0.9621	0.8791	0.8717	0.8665	0.7773	0.5711
BIC-D	0.9536	0.9519	0.9551	0.9569	0.9537	0.9441
KNN-10-D	0.9560	**0.9583**	**0.9564**	**0.9591**	**0.9618**	**0.9544**
KNN-3-D	0.9616	0.9416	0.9367	0.9019	0.8772	0.8464
GS-D	0.9508	0.8846	0.8007	0.7867	0.7048	0.6213
SVD-D	**0.9624**	0.9537	0.9497	0.9415	0.9377	0.9239
			LR			
Poisoned	0.9606	0.8927	0.8521	0.8005	0.6592	0.6390
BIC-D	0.9628	0.9569	0.9508	0.9583	0.9552	0.9455
KNN-10-D	0.9636	**0.9584**	**0.9560**	**0.9611**	**0.9565**	**0.9534**
KNN-3-D	0.9604	0.9450	0.9359	0.8927	0.8814	0.8484
GS-D	0.9545	0.9241	0.8004	0.7186	0.6079	0.5754
SVD-D	**0.9659**	0.9536	0.9452	0.9377	0.9353	0.9392
			ResNet-18			
Poisoned	0.9976	0.9548	0.8986	0.8735	0.8597	0.8266
BIC-D	0.9986	0.9918	0.9951	0.9908	0.9911	0.9869
KNN-10-D	**0.9988**	**0.9988**	**0.9976**	**0.9964**	**0.9961**	**0.9953**
KNN-3-D	0.9974	0.9935	0.9706	0.9688	0.9622	0.9568
GS-D	0.9968	0.9787	0.9311	0.8846	0.8246	0.8001
SVD-D	0.9986	0.9920	0.9644	0.9322	0.9073	0.8433
DPA	0.9904	0.9892	0.9883	0.9846	0.9814	0.9784
FA	0.9933	0.9909	0.9875	0.9861	0.9789	0.9729

TABLE 6: Classification ACC of victim classifiers on poisoned and sanitized MNIST.

as observed for TREC05 in Sec. 5.2, our detector prefers revising rather than removing poisoned components.

5.4 Statistical Test

We utilized a paired t-test [13] to assess the performance of our method relative to others:

1) We performed 5-fold cross-validation for BIC-D, KNN-D, SVD-D, and GS-D on 20NG and for BIC-D, DPA, and FA on CIFAR-10, recording the test fold accuracy after sanitization of each training fold. More specifically, we merged the original training and test sets, and then divided their union evenly into 5 folds. Each fold is used as the test set and the remaining folds constitute

Attack	0	1	2	3	4	5
			True Positive Rates (TPRs)			
BIC-D	-	0.9562	0.9556	0.9429	0.9568	0.9315
KNN-10-D	-	**0.9950**	**0.9918**	**0.9867**	**0.9834**	**0.9827**
KNN-3-D	-	0.8662	0.8106	0.7833	0.7543	0.7397
SVD-D-S	-	0.8600	0.8418	0.8591	0.8821	0.8832
SVD-D-L	-	0.8812	0.8668	0.8875	0.9009	0.9082
SVD-D-R	-	0.7725	0.6550	0.5287	0.4581	0.4352
			False Positive Rates (FPRs)			
BIC-D	0.0465	0.0723	0.0503	0.0406	0.0561	0.0531
KNN-10-D	0.0182	0.0175	**0.0166**	**0.0169**	**0.0186**	**0.0194**
KNN-3-D	0.0088	0.0109	0.0191	0.0394	0.0533	0.0695
SVD-D-S	**0.0100**	0.0112	0.0253	0.0338	0.0377	0.0467
SVD-D-L	0.0100	**0.0095**	0.0213	0.0270	0.0317	0.0367
SVD-D-R	0.0100	0.0182	0.0552	0.1131	0.1734	0.2259

TABLE 7: TPRs and FPRs of three defenses on MNIST.

Attack	0	1	2	3	4	5
Poisoned	0.8634	0.8502	0.8302	0.7974	0.7634	0.7430
BIC-D	0.8638	**0.8616**	**0.8528**	**0.8452**	**0.8446**	**0.8416**
KNN-D	0.7150	0.7154	0.6688	0.6758	0.6602	0.6752
GS-D	0.8272	0.8074	0.7866	0.7288	0.7036	0.6852
SVD-D	**0.8668**	0.8584	0.8466	0.8164	0.8046	0.7812
DPA	0.8044	0.8028	0.7971	0.7958	0.7852	0.7782
FA	0.8406	0.8274	0.8280	0.8210	0.8082	0.7984
GAN-AD	0.8250	0.8312	0.8208	0.8170	0.7854	0.7874

TABLE 8: Classification ACC of ResNet-18 on poisoned and sanitized CIFAR10.

Attack	0	1	2	3	4	5
			True Positive Rates (TPRs)			
BIC-D	-	**0.9275**	**0.9263**	**0.9133**	**0.9378**	**0.9290**
KNN-D	-	0.9025	0.8050	0.8112	0.7922	0.8010
SVD-D	-	0.3650	0.2662	0.3587	0.4171	0.3655
GAN-AD		0.736	0.7156	0.7052	0.663	0.716
			False Positive Rates (FPRs)			
BIC-D	0.0267	0.0494	0.0717	0.0881	0.1405	0.1626
KNN-D	0.4596	0.4628	0.4514	0.4533	0.4545	0.4484
SVD-D	**0.0100**	**0.0254**	**0.0587**	**0.0769**	**0.0932**	**0.1269**
GAN-AD		0.1	0.1	0.1	0.1	0.1

TABLE 9: TPRs and FPRs of four defenses on CIFAR10.

Attack	0	1	2	3	4	5
Poisoned	0.9028	0.8902	0.8722	0.8548	0.8328	0.8165
BIC-D	0.9028	**0.9080**	**0.9085**	**0.9020**	**0.9068**	**0.8897**
KNN-D	0.8000	0.7825	0.7440	0.7391	0.7217	0.7065
GS-D	0.9008	0.8972	0.8882	0.8631	0.8508	0.8405
SVD-D	**0.9068**	0.8954	0.8714	0.8622	0.8282	0.8111
DPA	0.8374	0.8345	0.8294	0.8274	0.8194	0.8008
FA	0.8502	0.8428	0.8337	0.8325	0.8242	0.8191

TABLE 10: Classification ACC of ResNet-34 on poisoned and sanitized STL10.

Attack	0	1	2	3	4	5
			True Positive Rates (TPRs)			
BIC-D	-	0.9100	**0.8850**	**0.8700**	**0.8575**	**0.8040**
KNN-D	-	**0.9700**	0.8500	0.8133	0.8100	0.8040
SVD-D	-	0.3800	0.2800	0.3066	0.3050	0.2220
			False Positive Rates (FPRs)			
BIC-D	**0**	**0.0012**	**0.0008**	**0.0008**	**0.0012**	**0.0028**
KNN-D	0.5028	0.4964	0.4936	0.4976	0.4944	0.4856
SVD-D	0.0100	0.0248	0.0576	0.0832	0.1112	0.1556

TABLE 11: TPRs and FPRs of three defenses on STL10.

Attack	0	1	2	3	4	5
			ACC			
Poisoned	0.8634	0.8502	0.8302	0.7974	0.7634	0.7430
BICSC-D	0.8516	0.8672	0.8608	0.8598	0.863	0.8586
KNNSC-D	0.8324	0.8502	0.8436	0.8546	0.8566	0.8648
			TPR			
BICSC-D	0	0.97125	0.9875	0.9613	0.9613	0.9535
KNNSC-D	0	0.97625	0.9862	0.97	0.9662	0.9715
			FPR			
BICSC-D	0.0914	0.0679	0.0934	0.082	0.0891	0.0791
KNNSC-D	0.09015	0.0708	0.07915	0.07115	0.0684	0.0672

TABLE 12: Performance of BICSC-D and KNNSC-D on poisoned CIFAR-10 datasets using deep feature representations extracted by an encoder trained via self contrastive learning. Attack refers to the number of classes involved in poisoning, and the poisoning ratio is 20% per class involved. "Poisoned" indicates the ACC of ResNet-18 trained on poisoned datasets.

the training set. Within each fold, we conducted the five attacks described in Sec. 5.1 using the poisoning samples.

2) Then we calculated the difference in accuracy between our method and one of the other methods for each fold.
3) Finally, we performed a t-test on this set of differences by $t = (\bar{x} - \mu_0)/(s/\sqrt{n})$, where $\bar{x}$ is the sample mean of the differences, μ_0 is the population mean under the null hypothesis, s is the sample standard deviation, and n is the sample size.

Our null hypothesis is that the mean difference between our method and others is less than or equal to zero, suggesting our method does not perform better. As shown in Tab. 14, upon conducting the paired t-test, all resulting t-statistics are much larger than 2.13 (except for SVD-D under attack 0). That is, all corresponding p-values are less than (or close to) 0.05. Therefore, we reject the null hypothesis, and conclude that our method performs significantly better than the others.

5.5 Strengthened Attacks

To further demonstrate BIC-D effectiveness against strong attacks, we varied the poisoning fraction for 20NG up to 50% (in Tab. 15) and poisoned CIFAR-100 with up to 100 classes (in Tab. 19). For 20NG, all classes were poisoned. For CIFAR-100, the poisoning ratio per class was fixed at 20%. For both datasets, BIC-D improves the ACC even under the strongest attacks, and outperforms the other defense methods.

5.6 Clean-Label Attacks

To further prove its effectiveness beyond label-flipping attacks, we apply BIC-D on CIFAR-10 poisoned by targeted *clean-label* poisoning attacks, *e.g.*, [48], [58]. Here, the poisoned samples are embedded with perturbations which (1) are human-imperceptible; and (2) cause the perturbed instances to be close to the target class instances in the embedded feature space [48]. After poisoning, the classifier will misclassify specific test instances to the target class while maintaining high classification accuracy on normal instances. We randomly chose class 9 as the target class. The poisoned ResNet-18 achieves accuracy of 0.9062 on clean test samples and misclassification rate of 0.9302 on samples manipulated by the attacker. Our BIC-D reduces the misclassification rate by 0.4744 and the clean accuracy by merely 0.0106. By comparison, KNN-D reduces the misclassification rate by 0.3209 and the clean accuracy by 0.0089.

5.7 Adaptive Attacks

Since BIC-D identifies a suspicious sample based on its "atypicality"(higher likelihood under a class other than its labeled class), as an adaptive attack, we poisoned the training set by flipping labels of only class-confused samples, *i.e.* those close to the decision boundary (with similar likelihoods under multiple classes). Note that this threat model is **stronger** than that described in Sec. 3 – the attacker is aware of the detection method, and has access to the training set to identity class-confused samples. We randomly chose 800 samples per class for training, with the rest used for testing. We trained a "one v.s. rest" SVM on the training set and identified 100 samples (for each attacked class) that are closest to the decision boundary. We generated two different adaptive attacks. In the first (Ada-Attack), we uniformly randomly mislabeled the confusing samples from class c to all other classes. In the second (CM-Ada-Attack), for each attacked class c, we mislabeled the confusing samples to the class with which c is most confusable (based on the test set confusion matrix). Since the poisoning and dataset split strategy is different from that discussed in Sec. 5.1, we also conducted a non-adaptive attack (non-Ada-Attack) for comparison, where 100 training samples per class are randomly selected and mislabeled.

The results for an SVM classifier are shown in Tab. 16. The first two columns show the poisoned classifier ACC/ACC of the classifier after BIC-D is applied. The ACCs of the SVM poisoned by the adaptive attacks on the *whole test set* are higher than that of the non-adaptive attack, since mislabeling samples which the model already struggles to classify may not drastically affect the decision boundary. However, we also show the accuracy on the *top 20% of test samples closest to the decision boundary* (confusing test samples) of the clean model. On these test samples, the effectiveness of the adaptive attacks is evident. Yet, BIC-D remains robust against the adaptive attacks. The ACCs with BIC-D on the whole test set are around 0.9, comparable to 0.9045 (non-Ada-Attack) and 0.9172 (the clean baseline from from Tab. 4). Moreover, BIC-D improves ACC on the class-confused test samples by about 20%, outperforming the improvement against the non-adaptive attack. Note also that the TPR (0.678) is lower than for the non-adaptive attack (0.874), demonstrating the difficulty in detecting poisoned samples near the decision boundary, compared with samples further away.

6 Scalability and Computational Complexity of the BIC-based Defense

6.1 Scalability

To show the scalability of BIC-D to datasets with more classes, we applied it on the complete 20NG dataset (with 20 categories) and on CIFAR-100 (with 100 categories), poisoned by the same strategy described in Sec. 5.1. For 20NG, we conducted attacks poisoning 5, 10, 15, and 20 classes, with the results shown in Tab. 17. For CIFAR-100, we conducted attacks poisoning 20, 40, 60, 80, and 100 classes, with the results shown in Tab. 19. As more classes are poisoned,

Attack	0	1	2	3	4	5
			20NG			
# components	(8,14,9,12,8)	(12,14,11,15,12)	(12,12,14,16,11)	(20,12,12,16,12)	(20,13,16,16,11)	(20,19,13,17,16)
# revised components	(1,1,4,0,3)	(2,5,7,6,8)	(6,6,10,5,6)	(7,5,7,7,7)	(9,6,13,5,9)	(7,7,8,7,8)
# removed components	(0,0,0,0,0)	(0,1,1,1,1)	(0,1,1,3,0)	(0,3,1,3,1)	(1,2,1,2,2)	(2,2,2,5,2)
			MNIST			
# components	(28,28,27,27,28)	(28,31,30,24,33)	(44,31,39,37,43)	(38,41,39,38,44)	(31,33,33,38,47)	(33,40,41,35,42)
# revised components	(6,4,15,10,5)	(4,2,12,15,7)	(3,4,14,14,5)	(3,7,7,3,4)	(6,4,12,11,4)	(7,5,2,7,6)
# removed components	(0,0,2,1,0)	(0,7,5,3,4)	(5,8,10,8,11)	(6,17,9,11,13)	(8,16,9,12,18)	(9,19,15,12,18)
			CIFAR10			
# components	(13,14,12,15,11)	(14,15,21,17,20)	(15,13,19,19,23)	(19,13,18,17,19)	(21,16,17,14,22)	(14,19,16,16,17)
# revised components	(1,2,4,3,1)	(0,0,1,0,1)	(0,0,0,1,1)	(0,0,0,0,1)	(0,0,0,0,0)	(0,0,0,0,0)
# removed components	(0,0,1,1,0)	(3,2,3,2,2)	(5,2,5,4,4)	(6,3,8,5,4)	(7,5,6,6,5)	(7,6,6,8,6)
			STL10			
# components	(8,10,11,8,10)	(9,9,16,9,12)	(10,8,16,12,12)	(15,13,17,17,11)	(15,15,16,16,11)	(17,14,12,14,16)
# revised components	(0,0,0,0,0)	(0,1,0,0,2)	(1,0,1,1,0)	(1,0,0,1,4)	(2,1,1,1,0)	(1,0,1,0,1)
# removed components	(0,0,0,0,0)	(0,1,2,2,1)	(2,1,4,3,3)	(3,3,3,5,3)	(4,5,4,6,4)	(4,5,4,5,6)

TABLE 13: The number of components, revised components, and removed components of each class of 20NG, MINIST, CIFAR10, and STL10.

Attack	0	1	2	3	4	5
			20NG			
vs KNN-D	14.17	5.69	9.27	9.27	20.04	9.05
vs SVD-D	2.12	24.24	7.58	16.57	19.95	17.85
vs GS-D	3.81	4.98	10.03	17.65	16.71	12.50
			CIFAR10			
vs DPA	7.18	12.65	11.89	7.57	7.74	22.22
vs FA	6.09	8.20	15.31	15.39	12.92	7.61

TABLE 14: T-statistics comparing the performance of our method to other methods.

Poisoning Ratio	0.1	0.2	0.3	0.4	0.5
Poisoned	0.838	0.719	0.602	0.508	0.422
BIC-D	0.948	0.935	0.928	0.865	0.807
KNN-D	0.894	0.854	0.803	0.738	0.628
SVD-D	0.867	0.806	0.646	0.527	0.398

TABLE 15: Classification ACC of BIC-D, KNN-D, and SVD-D on 20NG datasets at attack 5 (all 5 classes are involved in poisoning) with varied poisoning ratios. “Poisoned” indicates the ACC of ResNet-18 trained on poisoned datasets.

Attack	ACC on the whole test set	ACC on the confusing test samples	TPR/FPR
Ada-Attack	0.7939/0.9031	0.6192/0.8249	0.678/0.009
CM-Ada-Attack	0.7807/0.9061	0.6015/0.8173	0.678/0.010
Non-Ada-Attack	0.7335/0.9045	0.6903/0.8477	0.874/0.015

TABLE 16: Performance of BIC-D against the adaptive attacks (Ada-Attack and CM-Ada-Attack) on 20NG at attack 5. The first two columns are the classification ACC of the poisoned classifier/the classifier post BIC-D data sanitization.

the classifier ACC decreases greatly. But BIC-D improves the ACC under all attack cases. For comparison, we also show the performance of KNN-D. BIC-D outperforms KNN-D under all the attacks.

6.2 Computational Cost

We report execution times of all defenses across datasets in Tab. 18. We also show the time for BIC-D on CIFAR-100 in Tab. 19. All the experiments are conducted on a compute platform with an Intel i9-10900K CPU, NVIDIA GeForce 3080 GPU and 32GB memory. For TREC05, we consider the attack with 1000 poisoned ham and spam. For the other datasets, we consider attack 1. We chose SVM for TREC05 and 20NG, and ResNet-18 for the other datasets. For BIC-D, KNN-D, and SVD-D, the execution time represents anomaly detection and sample removal. Since GS-D, DPA, and FA mitigate DP attacks during DNN training, their reported execution time is the training time.

Attack	0	5	10	15	20
Poisoned	0.7602	0.6795	0.5689	0.4616	0.3871
BIC-D	0.7696	0.7411	0.7091	0.6792	0.6315
KNN-D	0.7304	0.7213	0.6696	0.6184	0.5846

TABLE 17: Test set ACC of SVM classifiers on poisoned and sanitized complete 20NG datasets.

	TREC05	20NG	MNIST	CIFAR-10	STL-10
BIC-D	77.81	760.01	427.32	315.07	192.81
KNN-D	19.20	3.26	2.08	283.49	163.02
GS-D	14.35	6.91	1808.68	2762.91	1709.62
SVD-D	-	49.45	1122.42	1721.55	455.47
DPA	-	-	837.14	2264.56	547.06
FA	-	-	1098.15	3025.18	703.54

TABLE 18: Time (in seconds) used for deploying all defense methods on different datasets.

KNN-D is efficient in all cases, while BIC-D takes more time. However, BIC-D execution is less than/comparable to the time to train the DNN. For example, it takes 497s, 835s, and 1300s to train a ResNet-18 on MNIST, CIFAR-10, and CIFAR-100, respectively. On MNIST, CIFAR-100 for attack 1, CIFAR-100 for attack 20, BIC-D takes 427s, 315s and 375s, respectively (all less than the training times). Moreover, BIC-D is more efficient than other defenses (excluding KNN-D) for MNIST, CIFAR-10, and STL-10. SVD-D is affordable for SVMs but is costly for ResNets due to repeated SVD calculation and DNN retraining. GS-D greatly increases the training time of ResNets due to the DP-SGD optimizer. As expected, DPA and FA are time-consuming, and their computational cost increases with the number of partitions of the training set.

7 Conclusions and Future Work

We proposed an *unsupervised* BIC-based mixture model defense against DP attacks on classifiers, where poisoned samples are an unknown subset of the training set. Our

Attack	0	20	40	60	80	100
Poisoned	0.5605	0.5444	0.5224	0.4979	0.4950	0.4693
BIC-D	0.5584	0.5611	0.5325	0.5313	0.5074	0.4857
KNN-D	0.5523	0.5393	0.5112	0.5015	0.4985	0.4733
BIC-Time	133.48	375.5	542.62	727.9	778.9	942.36

TABLE 19: Classification ACC of BIC-D and KNN-D on poisoned CIFAR-100 datasets, and time (in seconds) used for BIC-D. “Poisoned” indicates the ACC of ResNet-18 trained on poisoned datasets. Attack refers to the number of classes involved in poisoning, and the poisoning ratio is 20% per class involved. For reference, the training time of ResNet-18 on clean CIFAR-100 is 1300s.

defense applies mixture modeling to accurately explain the poisoned dataset and concentrate poisoned samples into several mixture components. It jointly identifies poisoned components and poisoned samples within them by minimizing the BIC cost, with the identified poisoned samples purged from the training set prior to classifier training. Experiments demonstrate the effectiveness of our defense against strong attacks and is superiority over other defenses.

Our approach does rely on distributional assumptions that may not always hold – we model text by multinomial mixtures, and internal DNN features for images by Gaussian mixtures. To address this, we suggest to determine the best-fitting distribution among *several* distribution candidates. Also, while we assumed diagonal covariances for Gaussians, “parsimonious” full covariances could be considered, *e.g.* [18], [19]. In our approach, component removal is aggressive – once a component is removed, it cannot be restored. This is beneficial if the removed component is mainly composed of poisoned data. But it may be harmful if the component contains a significant proportion of legitimate data, leading to information loss and degradation in accuracy. To mitigate this, we may consider “soft” removal instead of “hard” removal. For instance, for a deemed poisoned component, rather than completely removing it, we can reduce its influence by reducing its component mass (and that of its samples).

Acknowledgments

This research was supported in part by AFOSR and ONR grants and by a Cisco gift.

References

[1] Scott Alfeld, Xiaojin Zhu, and Paul Barford. Data poisoning attacks against autoregressive models. In *Proc. AAAI*, 2016.

[2] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Srndic, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *ECML PKDD*, 2013.

[3] Battista Biggio, Blaine Nelson, and Pavel Laskov. Support vector machines under adversarial label noise. In *ACML*, 2011.

[4] Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. In *Proc. ACM CCS*, 2018.

[5] Chengliang Chai, Lei Cao, Guoliang Li, Jian Li, Yuyu Luo, and Samuel Madden. Human-in-the-loop outlier detection. In *Proc. Int'l Conference on Management of Data*, 2020.

[6] Jian Chen, Xuxin Zhang, Rui Zhang, Chen Wang, and Ling Liu. De-pois: An attack-agnostic defense against data poisoning attacks. *IEEE Trans. Inf. Forensics Secur.*, 2021.

[7] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. A simple framework for contrastive learning of visual representations. In *ICML*, 2020.

[8] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv*, abs/1712.05526, 2017.

[9] Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, PMLR 15*, 2011.

[10] Gordon V. Cormack and Thomas R. Lynam. Trec 2005 spam public corpora. https://plg.uwaterloo.ca/~gvcormac/trecspamtrack05, 2005.

[11] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, pages 273–297, 1995.

[12] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society, Series B*, pages 1–38, 1977.

[13] Janez Demsar. Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, pages 1–30, 2006.

[14] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, pages 141–142, 2012.

[15] Ilias Diakonikolas, Gautam Kamath, Daniel Kane, Jerry Li, Jacob Steinhardt, and Alistair Stewart. Sever: A robust meta-algorithm for stochastic optimization. In *Proc. ICML*, 2019.

[16] RO Duda, PE Hart, and DG Stork. *Pattern Classification, Second Edition*. Wiley, 1999.

[17] Jiashi Feng, Huan Xu, Shie Mannor, and Shuicheng Yan. Robust logistic regression and classification. In *Proc. NeurIPS*, 2014.

[18] Zoubin Ghahramani and Matthew Beal. Variational inference for bayesian mixtures of factor analysers. *Adv. Neural Inf. Process*, 1999.

[19] Zoubin Ghahramani and Geoffrey E Hinton. The em algorithm for mixtures of factor analyzers. Technical report, Technical Report CRG-TR-96-1, University of Toronto, 1996.

[20] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations. In *ICLR*, 2018.

[21] Michael W. Graham and David J. Miller. Unsupervised learning of parsimonious mixtures on large spaces with integrated feature and component selection. *IEEE Trans. Signal Process.*, pages 1289–1303, 2006.

[22] Tianyu Gu, Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Evaluating backdooring attacks on deep neural networks. *IEEE Access*, pages 47230–47244, 2019.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[24] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, pages 1735–1780, 1997.

[25] Sanghyun Hong, Varun Chandrasekaran, Yigitcan Kaya, Tudor Dumitras, and Nicolas Papernot. On the effectiveness of mitigating data poisoning attacks with gradient shaping. *arXiv*, abs/2002.11497, 2020.

[26] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf, 2009.

[27] Ricky Laishram and Vir Virander Phoha. Curie: A method for protecting SVM classifier from poisoning attack. *arXiv*, abs/1606.01584, 2016.

[28] Ken Lang. Newsweeder: Learning to filter netnews. In *Proc. ICML*, pages 331–339, 1995.

[29] Aaron D. Lanterman. Schwarz, Wallace, and Rissanen: Intertwining Themes in Theories of Model Selection. *International Statistical Review*, page 185–212, 2001.

[30] Alexander Levine and Soheil Feizi. Deep partition aggregation: Provable defenses against general poisoning attacks. In *ICLR*, 2021.

[31] Bo Li, Yining Wang, Aarti Singh, and Yevgeniy Vorobeychik. Data poisoning attacks on factorization-based collaborative filtering. In *Proc. NeurIPS*, pages 1885–1893, Dec. 2016.

[32] Jintang Li, Tao Xie, Chen Liang, Fenfang Xie, Xiangnan He, and Zibin Zheng. Adversarial attack on large scale graph. *IEEE TKDE*, 2021.

[33] Xi Li, David J. Miller, Zhen Xiang, and George Kesidis. A scalable mixture model based defense against data poisoning attacks on classifiers. In *Proc. DDDAS*, 2020.

[34] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning attack on neural networks. In *Proc. NDSS*, 2018.

[35] Yuzhe Ma, Xiaojin Zhu, and Justin Hsu. Data Poisoning against Differentially-Private Learners: Attacks and Defenses. In *Proceedings IJCAI*, Aug. 2019.
[36] G McLachlan and D Peel. *Finite mixture models*. Wiley, 2004.
[37] David J. Miller, Zhen Xiang, and George Kesidis. Adversarial learning targeting deep neural network classification: A comprehensive review of defenses against attacks. *Proceedings of the IEEE*, pages 402–433, 2020.
[38] Luis Muñoz-González, Battista Biggio, Ambra Demontis, Andrea Paudice, Vasin Wongrassamee, Emil C. Lupu, and Fabio Roli. Towards poisoning of deep learning algorithms with back-gradient optimization. In *Proc. ACM AISec*, 2017.
[39] John Ashworth Nelder and Robert WM Wedderburn. Generalized linear models. *Journal of the Royal Statistical Society: Series A (General)*, pages 370–384, 1972.
[40] Blaine Nelson, Marco Barreno, Fuching Jack Chi, Anthony D. Joseph, Benjamin I. P. Rubinstein, Udam Saini, Charles Sutton, J. D. Tygar, and Kai Xia. Misleading learners: Co-opting your spam filter. In *Proc. Machine Learning in Cyber Trust: Security, Privacy, and Reliability*, 2009.
[41] Seong Joon Oh, Max Augustin, Mario Fritz, and Bernt Schiele. Towards reverse-engineering black-box neural networks. In *Proc. ICLR*, 2018.
[42] Nicolas Papernot, Patrick D. McDaniel, Ian J. Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proc. ACM AsiaCCS*, 2017.
[43] Nicolas Papernot, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *Proc. EuroS&P*, 2016.
[44] Andrea Paudice, Luis Muñoz-González, and Emil C. Lupu. Label sanitization against label flipping poisoning attacks. In *Proc. ECML PKDD Workshops*, 2018.
[45] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. In *Proc. ICLR*, 2018.
[46] J. Rissanen. Modeling by shortest data description. *Automatica*, September 1978.
[47] Gideon Schwarz. Estimating the Dimension of a Model. *The Annals of Statistics*, pages 461 – 464, 1978.
[48] Ali Shafahi, W. Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. Poison frogs! targeted clean-label poisoning attacks on neural networks. In *NeurIPS*, 2018.
[49] Shaoxu Song, Fei Gao, Ruihong Huang, and Yihan Wang. On saving outliers for better clustering over noisy data. In *Proc. International Conference on Management of Data*, June 2021.
[50] Jacob Steinhardt, Pang Wei Koh, and Percy Liang. Certified defenses for data poisoning attacks. In *Proc. NeurIPS*, pages 3517–3529, 2017.
[51] Yiyou Sun, Yifei Ming, Xiaojin Zhu, and Yixuan Li. Out-of-distribution detection with deep nearest neighbors. In *ICML*, 2022.
[52] Wenxiao Wang, Alexander Levine, and Soheil Feizi. Improved certified defenses against data poisoning with (deterministic) finite aggregation. In *ICML*, 2022.
[53] Wentai Wu, Ligang He, Weiwei Lin, Yi Su, Yuhua Cui, Carsten Maple, and Stephen A. Jarvis. Developing an unsupervised real-time anomaly detection scheme for time series with multi-seasonality. *IEEE TKDE*, 2020.
[54] Huang Xiao, Battista Biggio, Blaine Nelson, Han Xiao, Claudia Eckert, and Fabio Roli. Support vector machines under adversarial label contamination. *Neurocomputing*, pages 53–62, 2015.
[55] Yugen Yi, Wei Zhou, Yanjiao Shi, and Jiangyan Dai. Speedup two-class supervised outlier detection. *IEEE Access*, 2018.
[56] Vitjan Zavrtanik, Matej Kristan, and Danijel Skocaj. Dræm - A discriminatively trained reconstruction embedding for surface anomaly detection. In *ICCV*, 2021.
[57] Yuxin Zhang, Yiqiang Chen, Jindong Wang, and Zhiwen Pan. Unsupervised deep anomaly detection for multi-sensor time-series signals. *IEEE TKDE*, 2021.
[58] Chen Zhu, W. Ronny Huang, Hengduo Li, Gavin Taylor, Christoph Studer, and Tom Goldstein. Transferable clean-label poisoning attacks on deep neural nets. In *ICML*, 2019.

Xi Li is currently a Ph.D. candidate supervised by Dr. George Kesidis and Dr. David J. Miller with the School of Electrical Engineering and Computer Science, the Penn State University. She received the B.S. degree in Electrical Engineering from the Southeast University, Nanjing, China, in 2016, and the M.S. degree in Computer Science from the Penn State University in 2018. Her research interests mainly include adversarial learning, anomaly detection, deep learning.

PLACE PHOTO HERE

David J. Miller (Senior Member, IEEE) received the B.S.E. degree from Princeton University in 1987, the M.S.E. degree from the University of Pennsylvania in 1990, and the Ph.D. degree from the University of California at Santa Barbara in 1995, all in electrical engineering. He has been with the Department of Electrical Engineering at the Pennsylvania State University since 1995. His research interests include machine learning, source coding, and network security. He was a member of the Machine Learning for Signal Processing Technical Committee within the IEEE Signal Processing Society from 1997 to 2010 and from 2017-2022 and was its chair from 2007 to 2009.

Zhen Xiang (Graduate Student Member, IEEE) is a fifth-year PhD student in the Department of Electrical Engineering at Pennsylvania State University (PSU), advised by Prof David J. Miller and Prof George Kesidis. Before that, he received his B.Sc. degree in Electronics and Computer Engineering from Hong Kong University of Science and Technology with an outstanding student award in 2014, and his M.Sc. degree in Electrical Engineering from University of Pennsylvania in 2016. His research interests include adversarial machine learning and statistical signal processing. His PhD thesis focuses on defending backdoor attacks against deep neural network classifiers, which received the Dr. Nirmal K. Bose Dissertation Excellence Award in 2022. He served as reviewer for journals including IEEE TNNLS, Computers & Security, and IEEE SPM, and conferences including ICASSP, MLSP.

George Kesidis (Senior Member, IEEE) received the B.A.Sc. degree in EE from the University of Waterloo in 1988, and the M.S. degree (neural networks and stochastic optimization) and the Ph.D. degree (networking and performance evaluation) in EECS from U.C. Berkeley in 1990 and 1992, respectively. Following eight years as a professor of ECE with the University of Waterloo, he has been a Professor of EE and CSE with the Pennsylvania State University since 2000. His research interests include problems in networking, cyber security, machine learning, performance evaluation, and cloud computing. Currently his research is supported by grants from NSF, ONR and Cisco. Dr. Kesidis has served as a Technical Program Committee (TPC) Co-Chair of IEEE INFOCOM and an Associated Editor of the Computer Networks Journal, ACM Transactions on Modeling and Computer Simulation (TOMACS), and IEEE Communications Surveys and Tutorials (ComST).