Proving Robustness of KNN Against Adversarial Data Poisoning

Yannan Li, Jingbo Wang, and Chao Wang
University of Southern California, Los Angeles CA 90089, USA
{yannanli, jingbow, wang626}@usc.edu

***Abstract*—We propose a method for verifying data-poisoning robustness of the k-nearest neighbors (KNN) algorithm, which is a widely-used supervised learning technique. Data poisoning aims to corrupt a machine learning model and change its inference result by adding polluted elements into its training set. The inference result is considered *n-poisoning robust* if it cannot be changed by up-to-n polluted elements. Our method verifies n-poisoning robustness by soundly overapproximating the KNN algorithm to consider all possible scenarios in which polluted elements may affect the inference result. Unlike existing methods which only verify the inference phase but not the significantly more complex learning phase, our method is capable of verifying the entire KNN algorithm. Our experimental evaluation shows that the proposed method is also significantly more accurate than existing methods, and is able to prove the n-poisoning robustness of KNN for popular supervised-learning datasets.**

I. Introduction

Data poisoning is an attack aimed to corrupt a machine learning model by polluting its training data, and thus affect the inference results for test data [33]. Prior work shows that even a small amount of polluted data, e.g., $\leq 0.4\%$ of the training set, is enough to affect the inference result [34], [6], [8]. Thus, verifying the robustness of the inference result in the presence of data poisoning is a practically important problem. Specifically, given a potentially-polluted training set T, and the assumption that at most n elements in T are polluted, if we can prove that the inference result for a test input x remains unchanged by any n polluted elements in T, the inference result can still be considered trustworthy.

This work is concerned with n-poisoning robustness of the *k-nearest neighbors* (KNN) algorithm, which is a widely used supervised learning technique in applications such as e-commerce, video recommendation, document categorization, and anomaly detection [18], [2], [41], [1], [30], [14], [27], [36], [44]. However, the verification problem is challenging for two reasons. First, KNN relies heavily on numerical analysis, which involves a large number of non-linear arithmetic computations and complex statistical analysis techniques such as p-fold cross validation. They are known to be difficult for existing verification techniques. Second, even with a small n, there can be an extremely large number of possible scenarios in which polluted elements in T may affect the trained model and hence the inference result.

Specifically, let $m = |T|$ be the number of elements in T and $i \leq n$ be the actual number of polluted elements in T, the number of *clean subsets* of T (where polluted elements have been removed) is $\binom{m}{i}$. Since $i = 1, \ldots, n$, the total number of clean subsets of T is $\sum_{i=0}^{n} \binom{m}{i}$. Thus, it is impractical to *explicitly* check, for each clean subset $T' \subseteq T$, whether the inference result produced by the model trained using T' remains the same as the inference result produced by the model trained using T.

A practical approach, which is the one used by our method, is to *soundly over-approximate* the impact of all the clean subsets while analyzing the machine learning algorithm, following the *abstract interpretation* [9] paradigm for static program analysis. Here, the word *soundly* means that our method guarantees that, as long as the over-approximated inference result is proved robust, the actual inference result is robust. In addition to being sound, our method is efficient in that, instead of training a model for each clean subset T', it combines all clean subsets together to compute a set of abstract models in a single pass.

For KNN, in particular, each model corresponds to an optimal value of the parameter K, indicating how many neighbors in T are used to infer the output label of a test input x. Thus, our method computes an over-approximated set of K values, denoted $KSet$. Then, it over-approximates the KNN's inference phase, to check if the output label of x remains the same for all $K \in KSet$. If the output label remains the same, the inference result for x is considered robust against any of the possible n-poisoning attacks of the training set T.

To the best of our knowledge, our method is the first method that can soundly verify n-poisoning robustness of the entire KNN algorithm, consisting of both the learning (K parameter tuning) phase and the inference phase. In the literature, there are two closely related prior works. The first one, by Jia et al. [21], aims to verify the robustness of KNN's inference phase only; in other words, they require the K value to be fixed and given, with the implicit assumption that the optimal K value is not affected by data poisoning. Unfortunately, this is not a valid assumption, as shown by the motivating examples presented in Section II. Furthermore, by fixing the K value, the more challenging part of the verification problem has been sidestepped, which is verifying the *p-fold cross validation* during KNN's learning phase. How to over-approximate KNN's learning phase soundly and efficiently is a main contribution of our work.

The other closely-related prior work, by Drews et al. [12], aims to prove robustness of a different machine learning technique, namely the decision tree learning (DTL) algo-

https://doi.org/10.34727/2022/isbn.978-3-85448-053-2_6

rithm. Since DTL differs significantly from KNN in that it relies primarily on logical operations (such as And, Or, and Negation) as opposed to nonlinear arithmetic computations, their verification method relies on a fundamentally different technique (symbolic path exploration) from ours, and is not directly applicable to KNN.

At a high level, our verification method works as follows. Given a tuple $\langle T, n, x \rangle$, where T is the potentially-polluted training set, n is the maximum number of polluted elements in T, and x is a test input, our method tries to prove that, no matter which of the $i \leq n$ elements in T are polluted, the KNN's inference result for x remains the same. By default, the training set T corresponds to a model M, whose inference result for x is $y = M(x)$. Using an overapproximated analysis, our method checks if the output label $y' = M'(x)$ produced by a model M' corresponding to any clean subset of $T' \subseteq T$ remains the same as the default label $y = M(x)$. If that is the case, our method verifies the robustness of the inference result. Otherwise, it remains inconclusive.

We have implemented our method and conducted experimental evaluation using six popular machine learning datasets, which include both small and large datasets. The small datasets are particularly useful in evaluating the accuracy of the verification result because, when datasets are small, even the baseline approach of explicitly enumerating all clean subsets $T' \subseteq T$ is fast enough to complete and obtain the ground truth. The large datasets, some of which have more than 50,000 training data elements and thus are well beyond the reach of the baseline enumeration approach, are useful in evaluating the efficiency of our method. For comparison, we also evaluated the method of Jia et al. [21] with fixed K values.

Our experimental results show that, for KNN's inference phase only, our method is significantly more accurate than the method of Jia et al. [21] and as a result, proves robustness for many more cases. Overall, our method is able to achieve similar empirical accuracy as the ground truth on small datasets, while being reasonably accurate on large datasets and several orders-of-magnitudes faster than the baseline method. In particular, our method is the only one that can finish the complete verification of 10,000 test inputs for a training dataset with more than 50,000 elements within half an hour.

To summarize, this paper has the following contributions:

- We propose the first method for soundly verifying data-poisoning robustness of the entire KNN algorithm, consisting of both the learning phase and the inference phase.
- We evaluate the method on popular supervised learning datasets to demonstrate its advantages over both the baseline and a state-of-the-art technique.

The remainder of this paper is organized as follows. First, we review the definition of n-poisoning robustness and the basics of the k-nearest neighbors (KNN) algorithm in Section II. Then, we present the intuition and overview of our method in Section III. Next, we present our method for verifying the KNN learning phase in Section IV and verifying the KNN inference phase in Section V. We present our experimental results in Section VI, review the related work in Section VII, and give our conclusions in Section VIII.

II. Background

A. Data-Poisoning Robustness

Let L be a supervised learning algorithm that takes a set $T = \{(x, y)\}$ of training data elements as input and returns a learned model $M = L(T)$ as output. Within each data element, input $x \in \mathcal{X} \subseteq \mathbb{R}^D$ is an D-dimensional real-valued feature vector, and output $y \in \mathcal{Y} \subseteq \mathbb{N}$ is a natural number that represents a class label. The model is a prediction function $M : \mathcal{X} \to \mathcal{Y}$ that maps a test input $x \in \mathcal{X}$ to its class label $y \in \mathcal{Y}$. Following Drews et al. [12], we define data-poisoning robustness as follows.

a) n-Poisoning Model: Let T be a potentially-polluted training set, $m = |T|$ be the total number of elements in T, and n be the maximum number of polluted elements in T. Assuming that we do not know *which elements in T are polluted*, the set of all possible scenarios is captured by the set of *clean subsets*, denoted $\Delta_n(T) = \{T' \subseteq T : |T \setminus T'| \leq n\}$. In other words, each T' may be the result of removing all of the polluted elements from T.

b) n-Poisoning Robustness: We say the inference result $y = M(x)$ for a test input $x \in \mathcal{X}$ is robust to n-poisoning attacks of T if and only if, for all $T' \in \Delta_n(T)$ and the corresponding model $M' = L(T')$, we have $M'(x) = M(x)$. In other words, the predicted label remains the same.

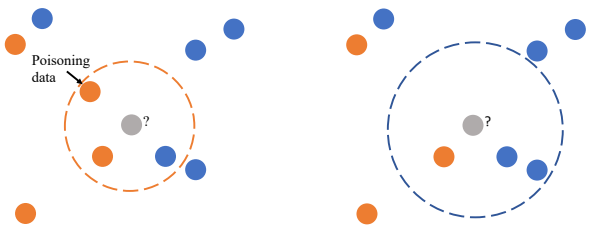
For example, when $T = \{a, b, c, d\}$ and $n = 1$, the clean subsets are $T_1 = \{b, c, d\}$, $T_2 = \{a, c, d\}$, $T_3 = \{a, b, d\}$ and $T_4 = \{a, b, c\}$, which correspond to models $M_1 - M_4$ and inference results $x_1 = M_1(x)$, $x_2 = M_2(x)$, $x_3 = M_3(x)$ and $x_4 = M_4(x)$. Let M be the default model obtained by T and $x = M(x)$ be the default output label. The inference result is 1-poisoning robust if and only if $x_1 = x_2 = x_3 = x_4 = x$.

This robustness definition has two advantages. First, whenever the inference result for a test input x is proved to be robust, it provides a strong guarantee of trustworthiness. Second, the verification procedure does not require the actual label of x to be known, which means it is applicable to unlabeled test data, which are common in practice.

B. k-Nearest Neighbors (KNN)

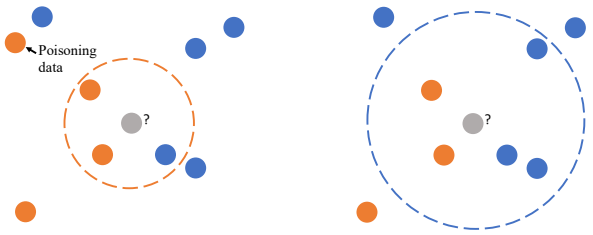
KNN is a supervised learning algorithm with two phases. During the learning phase, the training set T is used to compute the optimal value of the parameter K, which indicates how many neighbors in T to consider when deciding the output label for a test input x. During the inference phase, given an unlabeled test input $x \in \mathcal{X}$, the K nearest neighbors of x in T are used to compute the most frequent label, which is returned as the output label of x.

The distance between data elements, which is used to find the nearest neighbors of x in T, is defined on the input feature vectors. The most widely used metric is the Euclidean distance: given two elements $x_a, x_b \in \mathcal{X} \subseteq \mathbb{R}^D$, where D is the dimension of the input feature vector, the Euclidean distance is $\sqrt{\sum_{i=1}^{D} (x_a[i] - x_b[i])^2}$.



(a) polluted set (K=3) (b) clean set (K=3)

Fig. 1. Example of *direct influence* of the polluted data.



(a) polluted dataset (K=3) (b) clean dataset (K=5)

Fig. 2. Example of *indirect influence* of polluted data.

The optimal K value is the one that has the smallest average misclassification error on the training set T. The misclassification error is computed using *p-fold cross validation*, which randomly divides T into p groups of approximately equal size and, for each group, compute the misclassification error by treating this group as the test set and the union of all the other $p-1$ groups as the training set. Finally, the misclassification errors of the individual groups are used to compute the average misclassification error among all p groups.

III. The Intuition and Overview of Our Method

We first present the intuition behind our method, and then give an overview of the method in contrast to the baseline.

A. Two Ways of Affecting the Inference Result

In general, there are two ways in which polluted training elements in T affect the inference result. One of them, called *direct influence*, is to change the neighbors of x and thus their most frequent label. The other one, called *indirect influence*, is to change the parameter K itself.

Fig. 1 shows how polluted data may change the test input's neighbors and thus the inference result. Here, the gray dot represents the test input x, while the orange and blue dots represent elements in the training set T. There is only one polluted element, which is an orange dot marked in Fig. 1 (a). This element no longer exists in Fig. 1 (b). Assume that the optimal value for the parameter K is 3. For the clean set shown in Fig. 1 (b), the result is 'blue' since two of the three nearest neighbors of the test input x are blue. For the polluted set shown in Fig. 1 (a), however, the result is 'orange' since two of the three nearest neighbors are orange.

Fig. 2 shows how polluted data may change the inference result by changing the optimal value of the parameter K. In this case, the polluted element in Fig. 2 (a) is far away from the test input x. However, its presence changes the optimal value of the parameter K during the p-fold cross validation phase. While the K value for the clean set is 5, the K value for the polluted set is 3. As a result, the most frequent label of the neighbors is changed from 'blue' in Fig. 2 (b) to 'orange' in Fig. 2 (a).

These two examples highlight the importance of analyzing both the learning phase and the inference phase of the KNN algorithm. Otherwise, the verification result may be unsound, which is the case for Jia et al. [21] due to their implicit (and incorrect) assumption that K is not affected by polluted elements in T. In contrast, our method soundly verifies both phases of the KNN algorithm.

While verifying the KNN inference phase itself is already challenging, verifying the KNN learning phase is even more challenging, since it uses p-fold cross validation to compute the optimal K value.

B. Overview of Our Method

Before presenting our method, we present a conceptually-simple, but computationally-expensive, baseline method. It will help explain why the verification problem is challenging.

Algorithm 1: Baseline method $\mathsf{KNN\_Verify}(T, n, x)$.

```
for each T' ∈ Δ_n(T) do
    K' ← KNN_learn(T')
    y' ← KNN_predict(T', K', x)
    YSet ← YSet ∪ {y'}
end
robust ← (|YSet| = 1)
```

a) The Baseline Method: This method relies on checking whether the inference result remains the same for all possible ways in which the training set is polluted. Algorithm 1 shows the pseudo code, where T is the training set, n is the maximal polluted number, and x is a test input. For each clean subset $T' \in \Delta_n(T)$, the parameter K is computed using the standard KNN_learn subroutine, and used to predict the label of x using the standard KNN_predict subroutine. Here, $YSet$ stores the set of predicted labels; thus, $|YSet| = 1$ means the prediction result is always the same (and hence robust).

The baseline method is both sound and complete, and thus may be used to obtain the ground truth when the size of the dataset is small enough. However, it is not a practical solution for large datasets because of the combinatorial blowup – it has to explicitly enumerate all $|\Delta_n(T)| = \sum_{i=0}^{n} \binom{m}{i}$ cases. Even for $m = 100$ and $n = 5$, for example, the number becomes as large as $8 * 10^7$. For realistic datasets, often with tens of thousands of elements, the baseline method would not finish in a billion years.

b) The Proposed Method: Our method avoids enumerating the individual scenarios in $\Delta_n(T)$. As shown in Algorithm 2, it first analyzes, in a single pass, the KNN's learning phase while simultaneously considering the impact of up-to-n

Algorithm 2: Our method abs_KNN_Verify(T, n, x).

$KSet \leftarrow$ abs_KNN_learn(T, n)
$YSet \leftarrow$ abs_KNN_predict$(T, n, KSet, x)$
$robust \leftarrow (|YSet| = 1)$

Algorithm 3: Subroutine for the baseline: KNN_learn(T).

Divide T into p groups $\{G_i\}$ of equal size;
for *each* $K \in CandidateKset$ **do**
  for *each group* G_i **do**
    $errCnt_i^K = 0$
    for *each sample* $(x, y) \in G_i$ **do**
      $errCnt_i^K$++ when (KNN_predict$(T \setminus G_i, K, x) \neq y$);
    $error_i^K = errCnt_i^K / |G_i|$
  $error^K = \frac{1}{p} \sum_{i=1}^{p} error_i^K$
return the K value with the smallest $error^K$

Algorithm 4: Subroutine $KSet =$ abs_KNN_learn(T, n).

Divide T into p groups $\{G_i\}$ of equal size;
for *each* $K \in CandidateKset$ **do**
  for *each group* G_i **do**
    $errCntLB_i^K = errCntUB_i^K = 0$;
    for *each sample* $(x, y) \in G_i$ **do**
      $errCntLB_i^K$++ if (abs_KNN_cannot_obtain_correct_label$(T \setminus G_i, n, K, x, y) == True$);
      $errCntUB_i^K$++ if (abs_KNN_may_obtain_wrong_label$(T \setminus G_i, n, K, x, y) == True$);
    $errorLB_i^K = \max\{0,\ (errCntLB_i^K - n)/(|G_i| - n)\}$;
    $errorUB_i^K = \min\{errCntUB_i^K/(|G_i| - n),\ 1\}$;
  $errorLB^K = \frac{1}{p} \sum_{i=1}^{p} errorLB_i^K$;
  $errorUB^K = \frac{1}{p} \sum_{i=1}^{p} errorUB_i^K$;
Let $minUB =$ the smallest $errorUB^K$ for all K;
$KSet = \{K \mid errorLB^K \leq minUB\}$;

polluted elements in T. The result of this over-approximated analysis is a superset of possibly-optimal K values, stored in $KSet$. Details of the subroutine abs_KNN_learn is presented in Section IV.

Then, for each $K \in KSet$, our method analyzes the KNN's inference phase while considering all possible ways in which up-to-n elements in T may have been polluted. The result of this over-approximated analysis is a superset of possible output labels, denoted $YSet$. We say the inference result for x is robust if the cardinality of $YSet$ is 1; that is, the label of x remains the same regardless of how T may have been polluted. Details of the subroutine abs_KNN_predict is presented in Section V.

IV. Analyzing the KNN Learning Phase

To understand why soundly analyzing the KNN learning phase is challenging, we need to compare our method with the the original subroutine, KNN_learn, shown in Algorithm 3, which computes the optimal K value using p-fold cross-validation. Note that both the value of p and the $CandidateKset$ are hyper-parameters of the KNN algorithm itself, not part of the verification method. In practice, they typically do not depend on the size of T (see Section II-B for a detailed explanation).

A. The Algorithm

In contrast, our method shown in Algorithm 4 computes an over-approximated set of K values. The input consists of the training set T and the maximal polluted number n, while the output $KSet$ is a superset of the optimal K values.

Inside Algorithm 4, our method first computes the lower and upper bounds of the misclassification error for each K value, by considering the best case ($errorLB^K$) and the worst case ($errorUB^K$) when up-to-n elements in T are polluted.

After computing the interval $[errorLB^K, errorUB^K]$ for each K value, it computes $minUB$, which is the minimal upper bound among all K values.

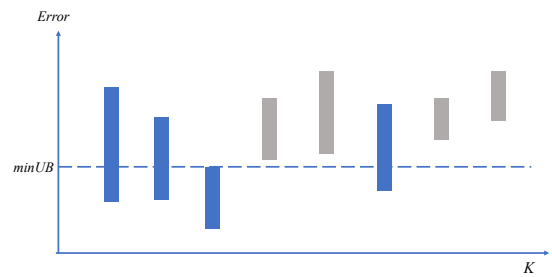


Fig. 3. Example of comparing the error bounds.

Then, by comparing $minUB$ with the $errorLB^K$ for each K, it over-approximates the set of possible K values that *may become the optimal K value* for some $T' \in \Delta_n(T)$.

Here, the intuition is that, by excluding K values that are *definitely not the optimal* K for any $T' \in \Delta_n(T)$ — they are the ones whose $errorLB^K$ is larger than $minUB$ — we obtain a sound over-approximation in $KSet$.

a) Example for $minUB$: Fig. 3 shows an example, where each vertical bar represents the interval $[errorLB^K, errorUB^K]$ of a candidate K value, and the blue dashed line represents $minUB$. The selected K values are those corresponding to the blue bars, since their $errorLB^k$ are smaller than $minUB$. The K values corresponding to the gray bars are dropped, since they definitely cannot have the smallest misclassification error.

b) The Soundness Guarantee: To understand why the $KSet$ computed in this manner is an over-approximation, assume that $minUB = errorUB^{K'}$ for some value K'. We now explain why K cannot be the optimal value (with the smallest error) when $errorLB^K > minUB$. Let the actual errors be $error^K \in [errorLB^K, errorUB^K]$ and $error^{K'} \in [errorLB^{K'}, errorUB^{K'}]$. Since we have $errorLB^K > errorUB^{K'}$, we know $error^K$ must be larger than $error^{K'}$. Therefore, K cannot have the smallest error.

To compute the interval $[errorLB^K, errorUB^k]$, we add up the misclassification error for each element $(x, y) \in G_i$,

where $x \in \mathcal{X}$ is the input and $y \in \mathcal{Y}$ is the (correct) label. For each element (x, y), there is a misclassification error if, for some reason, y differs from the predicted label.

Here, $errCntLB_i^K$ corresponds to the best case scenario — removing n elements from T in such a way that prediction becomes *as correct as possible*. In contrast, $errCntUB_i^K$ corresponds to the worst case scenario — removing n elements from T in such a way that prediction becomes *as incorrect as possible*. These two error counts are computed by two subroutines, which will be presented later in this section.

To convert $errCntLB_i^K$ and $errCntUB_i^K$ to error rates, we consider removing n misclassified elements when computing the lower bound $errorLB_i^K$, and removing n correctly-classified data elements when computing the upper bound $errorUB_i^K$. We assume $n < |G_i|$, which is a reasonable assumption in practice.

To explain subroutines abs_cannot_obtain_correct_label and abs_may_obtain_wrong_label, we need to introduce some notations, including label counter and removal strategy.

B. The Label Counter

Nearest Neighbors T_x^K. Let T_x^K be a subset of T consisting of the K nearest neighbors of x. For example, given T = {((0.1, 0.1), l_2), ((1.1, 0.1), l_1), ((0.1, 1.1), l_1), ((2.1, 3.1), l_3), ((3.3, 3.1), l_3)}, test input $x = (1.1, 1.1)$, and $K = 3$, the set is T_x^3 = {((0.1, 0.1), l_2), ((1.1, 0.1), l_1), ((0.1, 1.1), l_1)}. Here, we assume each neighbor has two real-valued input features and three possible output class labels $l_1 - l_3$.

Label Counter $\mathcal{E}(T_x^K)$. Given any dataset Z, including T_x^K, we use $\mathcal{E}(Z) = \{ (l_i : \#l_i) \}$ to represent the label counts, where l_i is a class label, and $\#l_i \in \mathbb{N}$ is the number of elements in Z that have the label l_i. For example, given T_x^3 above, we have $\mathcal{E}(T_x^3) = \{(l_1 : 2), (l_2 : 1)\}$, meaning it has two elements with label l_1 and one with label l_2.

Most Frequent Label $Freq(\mathcal{E}(T_x^K))$. Given a label counter $\mathcal{E}$, the most frequent label, denoted $Freq(\mathcal{E})$, is the label with the largest count. Similarly, we can define the second most frequent label. Thus, the KNN inference phase can be described as computing $Freq(\mathcal{E}(T_x^K))$ for the training set T, test input x, and K value.

Tie-Breaker $\mathbb{1}_{(l_i<l_j)}$. If two labels have the same frequency, the KNN algorithm may use their lexicographic order as a tie-breaker to ensure that $Freq(\mathcal{E})$ is unique: Let $<$ be the order relation, $(l_i < l_j)$ must be either true or false. Thus, we define an indicator function, $\mathbb{1}_{(l_i<l_j)}$, to return the numerical value 1 (or 0) when $(l_i < l_j)$ is true (or false).

C. The Removal Strategy

The removal strategy is an abstract way of modeling the impact of polluted data elements. In contrast, the removal set is a concrete way of modeling the impact.

The Removal Set. Given a dataset Z, the removal set $R \subset Z$ can be any subset of Z. Given T_x^3 above, for example, there are 6 possible removal sets: $R_1 = \{(x_1, y_1)\}$, $R_2 = \{((x_2, y_2))\}$, $R_3 = \{(x_3, y_3)\}$, $R_4 = \{(x_1, y_1), (x_2, y_2)\}$, $R_5 = \{(x_1, y_1), (x_3, y_3)\}$, and $R_6 = \{(x_2, y_2), (x_3, y_3)\}$. In particular, R_1 means removing element (x_1, y_1) from Z.

The Removal Strategy. The removal strategy is simply the label counter of a removal set R, denoted $\mathcal{S} = \mathcal{E}(R)$. In the above example, the six removal sets correspond to only four removal strategies $\mathcal{S}_1 = \{(l_1 : 1)\}$, $\mathcal{S}_2 = \{(l_2 : 1)\}$, $\mathcal{S}_3 = \{(l_1 : 1), (l_2 : 1)\}$, and $\mathcal{S}_4 = \{(l_1 : 2)\}$. In particular, $\mathcal{S}_2$ means removing an element labeled l_2; however, it does not say which of the l_2 elements is removed. Thus, it captures any removal set that has the same label counter.

The Strategy Size. Let the removal strategy be denoted $\mathcal{S} = \{(l_i : \#l_i)\}$, we define the size as $||\mathcal{S}|| = \sum_{(l_i,\#l_i)\in\mathcal{S}} \#l_i$ — it is the total number of removed elements. For $\mathcal{S}_1 = \{(l_1 : 1)\}$, $\mathcal{S}_2 = \{(l_2 : 2)\}$, and $\mathcal{S}_3 = \{(l_1 : 1), (l_3 : 3)\}$, the strategy size would be $||\mathcal{S}_1||$ = 1, $||\mathcal{S}_2||$ = 2, and $||\mathcal{S}_3||$ = 4.

In the context of the abstract interpretation paradigm [9], the removal sets can be viewed as the *concrete domain* while the removal strategies can be viewed as the *abstract domain*. Focusing on the abstract domain during verification makes our method more efficient. Let $|\mathcal{L}|$ be the total number of class labels, which is often small in practice (e.g., 2 or 10). Since the count of each label in a removal set is at most n, the number of removal strategies is at most $\sum_{i=0}^{n} \binom{i+|\mathcal{L}|-1}{i}$. This can be exponentially smaller than the number of possible removal sets, which is $\sum_{i=0}^{n} \binom{|T|}{i}$.

D. Misclassification Error Bounds

Using the notations defined so far, we present our method for computing the lower and upper bounds, $errCntLB_i^K$ and $errCntUB_i^K$, as shown in Algorithms 5 and 6.

Both bounds rely on computing T_x^{K+n}, the $K+n$ neighbors of x in T, and the label counter $\mathcal{E}(T_x^{K+n})$.

- The first subroutine checks whether it is impossible, even after removing up-to-n elements from T, that the correct label y becomes the most frequent label.
- The second subroutine checks whether it is possible, after removing up-to-n elements from T, that some wrong label becomes the most frequent label.

Before explaining the details, we present Theorem 1, which states the correctness of these checks. It says that, to model the impact of all subsets $T' \in \Delta_n(T)$, we only need to analyze the $(K + n)$ nearest neighbors of x, stored in T_x^{K+n}.

Theorem 1 *$\forall T' \in \Delta_n(T)$, we have $Freq(\mathcal{E}((T')_x^K)) \in \{Freq(\mathcal{E}(T_x^{K+n}) \setminus \mathcal{S}) | \mathcal{S} \subset \mathcal{E}(T_x^{K+n}), ||\mathcal{S}|| \leq n\}$.*

For brevity, we omit the detailed proof. Instead, we give the intuition behind the proof as follows:

- For each clean training subset $T' \in \Delta_n(T)$, we can always find a label counter $\mathcal{E}(T_x^{K+i})$ and a removal strategy $\mathcal{S} \in \mathcal{E}(T_x^{K+i})$, where $||\mathcal{S}|| = i \leq n$, satisfying $\mathcal{E}(T_x^{K+i} \setminus \mathcal{S}) = \mathcal{E}((T')_x^K)$.
- If we want to check all the predicted labels of x generated by all $T' \in \Delta_n(T)$, we need to search through all of $\mathcal{E}(T_x^K)$, $\mathcal{E}(T_x^{K+1})$, …, $\mathcal{E}(T_x^{K+n})$, which is expensive when n is large.

Algorithm 5: Subroutine used in our Algorithm 4 $flag =$ abs_KNN_cannot_obtain_correct_label(T, n, K, x, y).

Let $\mathcal{E}(T_x^{K+n})$ be the label counter of T_x^{K+n};
Define removal strategy $\mathcal{S} = \{ (y' : \#y' - \#y + \mathbb{1}_{y'<y}) \mid (y' : \#y') \in \mathcal{E}(T_x^{K+n}), y' \neq y, \#y' \geq \#y\}$;
return $(||\mathcal{S}|| > n)$;

Algorithm 6: Subroutine used in our Algorithm 4 $flag =$ abs_KNN_may_obtain_wrong_label(T, n, K, x, y).

Let $\mathcal{E}(T_x^{K+n})$ be the label counter of T_x^{K+n};
Let y' be the most frequent label in $\mathcal{E}(T_x^{K+n})$ except the label y;
Define removal strategy
$\mathcal{S} = \{ (y' : \max\{0, \#y - \#y' + \mathbb{1}_{y<y'}\}) \}$;
return $(||\mathcal{S}|| \leq n)$;

- Fortunately, $\mathcal{E}(T_x^{K+n}) \setminus \mathcal{S}$, where $||\mathcal{S}|| \leq n$, contains all the possible scenarios denoted by $\mathcal{E}(T_x^{K+i}) \setminus \mathcal{S}$, where $||\mathcal{S}|| = i$ and $i = 0, \ldots, n-1$.

As a result, we only need to analyze $\mathcal{E}(T_x^{K+n})$, which corresponds to the $(K+n)$ nearest neighbors of x; other elements which are further away from x can be safely ignored.

E. Algorithm 5

To compute the lower bound $errCntLB_i^K$, Algorithm 5 checks if all the strategies $\mathcal{S}$ satisfying $Freq(\mathcal{E}(T_x^{K+n}) \setminus \mathcal{S}) = y$ and $\mathcal{S} \subset \mathcal{E}(T_x^{K+n})$ must have $||\mathcal{S}|| > n$.

Fig. 4 shows two examples. In each example, the gray dot is the test input x and the other dots are neighbors of x in T_x^{K+n}. In Fig. 4 (a), $\#orange = 2$ is the number of orange dots (votes of the correct label). In contrast, $\#blue = 5$ and $\#green = 2$ are votes of the incorrect labels. By assuming the lexicographic order $blue < green < orange$, we define the indicator functions (tie-breakers) as $\mathbb{1}_{blue<orange} = 1$ and $\mathbb{1}_{green<orange} = 1$.

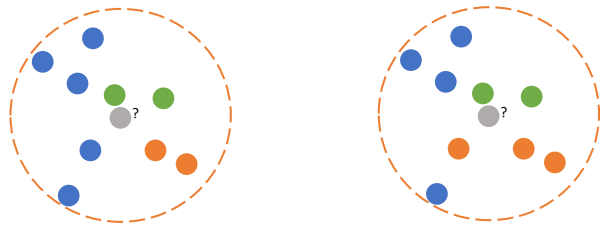
Given the removal strategy $\mathcal{S} = \{(blue : 4), (green : 1)\}$, we know $||\mathcal{S}|| = 5$ and, since $n = 4$, we have $||\mathcal{S}|| > n$. Thus, removing up to n =4 dots cannot make the test input x correctly classified (as orange). As a result, $errCntLB_i^K++$ is executed to increase the lower bound.

In Fig. 4 (b), however, since $\#blue = 4$, $\#orange = 3$, $\mathbb{1}_{blue<orange} = 1$, and $\mathcal{S} = \{(blue : 2)\}$, we have $||\mathcal{S}|| = 2$. Since $||\mathcal{S}|| \leq n$, removing up to n =4 dots can make the test data x correctly classified (as orange). As a result, $errCntLB_i^K++$ is not executed.

F. Algorithm 6

To compute the upper bound $errCntUB_i^K$, Algorithm 6 checks if there exists a strategy $\mathcal{S}$ that satisfies the condition: $Freq(\mathcal{E}(T_x^{K+n}) \setminus \mathcal{S}) \neq y$, $\mathcal{S} \subset \mathcal{E}(T_x^{K+n})$, and $||\mathcal{S}|| \leq n$.

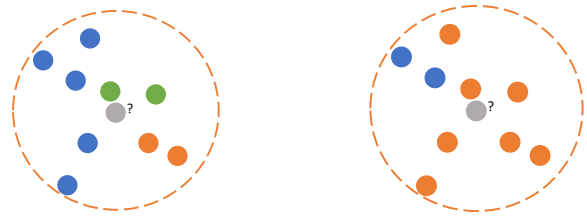
Fig. 5 shows two examples. In Fig. 5 (a), $\#orange = 2$ is the number of correct label, and $\#blue = 5$ is the number of dots with the most frequent wrong label. Thus, $\mathcal{S} = \emptyset$ and since $||\mathcal{S}|| \leq n$, we know that removing up to $n = 4$ dots can make the test data misclassified. As a result, $errCntUB_i^K++$ is executed.



(a) $\mathcal{S} = \{(blue : 4), (green : 1)\}$ and return value is $true$.

(b) $\mathcal{S} = \{(blue : 2)\}$ and return value is $false$.

Fig. 4. Examples for Algorithm 5 with $K = 5$, $n = 4$, and $y = orange$ being the correct label.



(a) $\mathcal{S} = \emptyset$ and return value is $true$.

(b) $\mathcal{S} = \{(orange : 5)\}$ and return value is $false$.

Fig. 5. Example for Algorithm 6 with $K = 5$, $n = 4$, $y = orange$ as correct label, and $y' = blue$ as the most frequent wrong label.

In Fig. 5 (b), $\#orange = 7$ is the number of orange dots, $\#blue = 2$ is the number of dots with the most frequent wrong label. Here, we assume $\mathbb{1}_{orange<blue} = 0$. Thus, $\mathcal{S} = \{(orange : 5))\}$ and since $||\mathcal{S}|| > n$, we know that removing up to $n = 4$ dots cannot make 'blue' (or any other wrong label) the most frequent label. As a result, $errCntUB_i^K++$ is not executed.

V. Analyzing the KNN Inference Phase

In this section, we present our method for analyzing the KNN inference phase, implemented in Algorithm 2 as the subroutine $YSet = abs\_KNN\_predict(T, n, KSet, x)$, which returns a set of output labels for test input x, by assuming that T contains up-to-n polluted elements.

A. Computing the Classification Labels

Algorithm 7 shows our method, which first checks whether the second most frequent label (y') can become the most frequent one after removing at most n elements. This is possible only if there exists a strategy $\mathcal{S}$ such that (1) it removes at most n elements labeled y, and (2) after the removal, y' becomes the most frequent label. This is captured by the condition $||\mathcal{S}|| = (\#y - \#y' + \mathbb{1}_{y<y'}) \leq n$. Otherwise, the predicted label is not unique.

We do not attempt to compute more than two labels, as shown by the return statement in the then-branch, because they are not needed by the top-level procedure (Algorithm 2), which only needs to check if $|YSet| = 1$ for the purpose of proving n-poisoning robustness.

Algorithm 7: Method abs_KNN_predict$(T, n, KSet, x)$.

$YSet = \{\ \}$
$visited = \{\ \}$
while $\exists K \in (KSet \setminus visited)$ **do**
  Let $\mathcal{E}(T_x^{K+n})$ be the label counter of T_x^{K+n};
  Let y be the most frequent label of $\mathcal{E}(T_x^{K+n})$;
  Let y' be the second most frequent label of $\mathcal{E}(T_x^{K+n})$;
  Let removal strategy $\mathcal{S} = \{\ (y : \#y - \#y' + \mathbb{1}_{y<y'})\ \}$;
  if $||\mathcal{S}|| \leq n$ **then**
    $YSet = YSet \cup \{y, y'\}$;
    return $YSet$;
  else
    $YSet = YSet \cup \{y\}$;
    $K^{LB} = K - (\#y - \#y' - n - \mathbb{1}_{y'<y})$;
    $K^{UB} = K + (\#y - \#y' - n - \mathbb{1}_{y'<y})$;
    $visited = visited \cup [K^{LB}, K^{UB}]$
return $YSet$;

B. Pruning Redundant K Values

Inside Algorithm 7, after checking $K \in KSet$, our method puts K into the $visited$ set to make sure it will never be checked again for the same test input x. In addition, it identifies other values in $KSet$ that are guaranteed to be *equivalent to* K, and prunes away these redundant values. Here, equivalent K values are defined as those with the same inference result for test input x.

To be conservative, we underapproximate the set of equivalent K values. As a result, these K values can be safely skipped since the (equivalent) inference result has been checked. This optimization is implemented using the $visited$ set in Algorithm 7. The $visited$ set is computed from K and $\mathcal{E}(T_x^{K+n})$ based on the expression $(\#y - \#y' - n - \mathbb{1}_{y'<y})$ over the removal strategy.

a) The Correctness Guarantee: We now explain why this pruning technique is safe. The intuition is that, if the most frequent label $Freq(\mathcal{E}(T_x^{K+n}))$ is the label with significantly more counts than the second most frequent label, then it may also be the most frequent label for another value K'. There are two possibilities:

- If $(K' < K)$, then $T_x^{K'+n}$ has $(K - K')$ fewer elements than T_x^{K+n}. Since removing elements from the neighbors will not increase the label count $\#y'$, the only way to change the inference result is decreasing the label count $\#y$. When $(K - K') \leq (\#y - \#y' - n - \mathbb{1}_{y'<y})$, decreasing $\#y$ will not make any difference. Thus, the lower bound of K' is $K - (\#y - \#y' - n - \mathbb{1}_{y'<y})$.
- If $(K' > K)$, then $T_x^{K'+n}$ has $(K' - K)$ more elements than T_x^{K+n}. Since adding elements to the neighbors will not decrease the label count $\#y$, the only way to change the inference result is increasing the label count $\#y'$. However, as long as $(K' - K) \leq (\#y - \#y' - n)$, increasing $\#y'$ will not make any difference. Thus, the upper bound of K' is $K + (\#y - \#y' - n - \mathbb{1}_{y'<y})$.

For example, consider K = 13, n = 2, and $\mathcal{E}(T_x^{15}) = \{(l_1 : 12), (l_2 : 2), (l_3 : 1)\}$. According to Algorithm 7, $\#y - \#y' - n - \mathbb{1}_{y'<y} = 12 - 2 - 2 = 8$ and thus we compute the interval $[13 - 8, 13 + 8] = [5, 21]$. As a result, candidate K values in the set $\{5, 6, 7, \ldots, 21\}$ can be safely skipped.

TABLE I
STATISTICS OF THE SUPERVISED LEARNING DATASETS.

Name	# training data ($\|T\|$)	# test data ($\|XSet\|$)	# output label ($\mathcal{L}$)	# input dimension (D)
Iris [15]	135	15	3	4
Digits [17]	1,617	180	10	64
HAR [3]	9,784	515	6	561
Letter [16]	18,999	1,000	26	16
MNIST [24]	60,000	10,000	10	36
CIFAR10 [23]	50,000	10,000	10	288

VI. EXPERIMENTS

We have implemented our method in Python and using the machine learning library scikit-learn 0.24.2, and evaluated it on two sets of supervised learning datasets. Table I shows the statistics, including the name, size of the training set, size of the test set, number of output class labels, and dimension of the input feature space. For MNIST and CIFAR10, in particular, the features were extracted using the standard histogram of oriented gradients (HOG) method [10].

The first set of datasets consists of Iris and Digits, two small datasets for which even the baseline method as shown in Algorithm 1 can finish and thus obtain the ground truth. We use the ground truth to evaluate the accuracy of our method. The second set of datasets consists of HAR, Letter, MNIST, and CIFAR10, which are larger datasets used to evaluate the efficiency of our method.

For comparison purposes, we also implemented the baseline method in Algorithm 1, and the method of Jia et al. [21], which represents the state of the art. Experiments were conducted on polluted training sets obtained by randomly inserting $\leq n$ input and output mutated samples to the original datasets. Since the same polluted training sets are used to compare all verification methods, and since the verification methods are deterministic, there is no need to run the experiments multiple times and then compute the average. Instead, we run each verification method on each polluted training set once. All experiments were conducted on a computer with a 2 GHz Quad-Core Intel Core i5 CPU and 16 GB of memory.

A. Results on the Small Datasets

We first compared our method with the baseline on the small datasets where the baseline method could actually finish. This is important because the baseline method does not rely on over-approximation, and thus can obtain the ground truth. Here, the ground truth means *which of the test data have inference results that are actually robust against n-poisoning attacks*. By comparing the ground truth with our result, we were able to evaluate the accuracy of our method.

Table II shows the results. Column 1 shows the name of the dataset and the polluted number n. Columns 2-3 show the result of the baseline method, consisting of the number of verified test data and the time taken. Similarly, Columns 4-5

TABLE II
RESULTS OF OUR METHOD AND THE BASELINE METHOD ON THE SMALL DATASETS WITH THE MAXIMAL POLLUTED NUMBER n=1, 2, AND 3.

Name	Baseline		New Method		Accuracy
	# robust	time (s)	# robust	time (s)	
Iris (n=1)	15/15	60	14/15	1	93.3%
iris (n=2)	14/15	4,770	13/15	1	92.9%
iris (n=3)	-	>9,999	11/15	1	-
Digits (n=1)	179/180	8,032	172/180	1	96.1%
Digits (n=2)	-	>9,999	170/180	1	-
Digits (n=3)	-	>9,999	165/180	1	-

TABLE III
RESULTS OF OUR METHOD ON LARGE DATASETS, AND ON SMALL DATASETS BUT WITH LARGER POLLUTED NUMBERS.

Name	Polluted Number (n)	Verified Percentage (# robust/$\|XSet\|$)	Verification Time (s)
Iris	1∼5 (4%)	93.3%∼73.3%	1 ∼ 1
Digits	1∼16 (1%)	95.6%∼80.6%	1 ∼ 2
HAR	1∼98 (1%)	99.4%∼71.7%	85 ∼ 93
Letter	1∼190 (1%)	94.0%∼5.5%	33 ∼ 43
MNIST	1∼600 (1%)	99.9%∼53.5%	888 ∼ 994
CIFAR10	1∼500 (1%)	99.2%∼2.8%	1,453 ∼ 1,559

show the result of our method. Column 6 shows the accuracy of our method in percentage.

The results indicate that, for test data that are indeed robust according to the ground truth, our method can successfully verify most of them. In *Iris (n=2)*, for example, Column 2 shows that 14 of the 15 test data are robust according to the baseline method, and Column 4 shows that 13 out of these 15 test data are verified by our method. Therefore, our method is 92.9% accurate.

Our method is much faster than the baseline. For *Digits (n=1)*, in particular, our method took only 1 second to verify 172 out of the 180 test data as being robust while the baseline method took 8,032 seconds. As the polluted number n increases, the baseline method ran out of time even for these small datasets. As a result, we no longer have the ground truth needed to directly measure the accuracy of our method. Nevertheless, since all cases verified by our method are guaranteed to be robust, the number of verified test data in Column 4 of Table II serves as a proxy – it decreases slowly as n increases, indicating that the accuracy of our method remains high.

B. Results on the Large Datasets

We also evaluated our method on the large datasets. Table III summarizes the results on these large datasets as well as the two small datasets but with larger polluted numbers (n). Since these verification problems are out of the reach of the baseline method, we no longer have the ground truth. Thus, instead of measuring the accuracy, we measure the percentage of test data that we can verify, shown in Column 3 of Table III.

For example, in $Iris$, $n = 1 \sim 5$ (4%) in Column 2 means that these experiments were conducted for each poisoning number $n = 1, 2, \ldots 5$. Since the training dataset has 135 elements, $n = 5$ means 4% (or 5/135) of these training data may have been polluted. In Column 3, 93.3% is the percentage of verified test data for $n = 1$, while 73.3% is the percentage of verified test data for $n = 5$. Except for *Iris*, which has a small number of training data, we set the poisoning number n to be less than 1% of the training dataset.

Overall, our method remains fast as the sizes of T, $XSet$ and n increase. For *MNIST*, in particular, our method finished analyzing both 10-fold cross validation and KNN inference in 26 minutes, for all of the 60,000 data elements in the training set and 10,000 data elements in the test set. In contrast, the baseline method failed to verify any of the test data within the 9999-second time limit.

Without the ground truth, the verified percentage provides a lower bound on the number of test data that remain robust against data-poisoning attacks. When n=1, the verified percentage in Column 3 is high for all datasets. As the polluted number n increases to 1% of the entire training set T, the verified percentage decreases. Furthermore, the decrease is more significant for some datasets than for other datasets. For example, In MNIST, at least 53.5% of the test data remain robust under 1% (or 600) poisoning attacks. In CIFAR10, however, only 2.8% of the test data remains robust under 1% (or 500) poisoning attacks. Thus, the relationship between the verified percentage and the polluted number reflects more about the unique characteristics of these datasets. By this, we mean that if one dataset has more *truly-non-robust* cases than another dataset, then the verifier will report more *cannot-be-verified* cases.

The reason why the accuracy is low for Letter and CIFAR10 datasets is because they have larger attack surfaces in the extracted feature space: elements from the same class are not sufficiently concentrated in one area, and the neighbors include many elements from other classes. Thus, small changes to the neighbors can lead to significant changes of the class label. While we believe that the accuracy (measured by the verified percentage) may improve if a better feature extractor is used (to improve the quality of extracted features), it is out of the scope of the verification task.

C. Compared with the Existing Method

While our method is the only one that can verify the entire KNN algorithm, there are existing methods that can verify part of the KNN algorithm. The most recent method proposed by Jia et al. [21], in particular, aims to verify the KNN inference step with a given K value; thus, it can be regarded as functionally equivalent to the subroutine of our method as presented in Algorithm 7. However, our method is significantly more accurate due to its tighter approximation. To experimentally demonstrate the advantage of our method, we used their method to replace Algorithm 7 in our own method before conducting the experimental comparison. Since an open-source implementation of their method is not available, we have implemented it ourselves.

Fig. 6 shows the results, where *blue* lines represent our method and orange lines represent their method [21]. Overall,

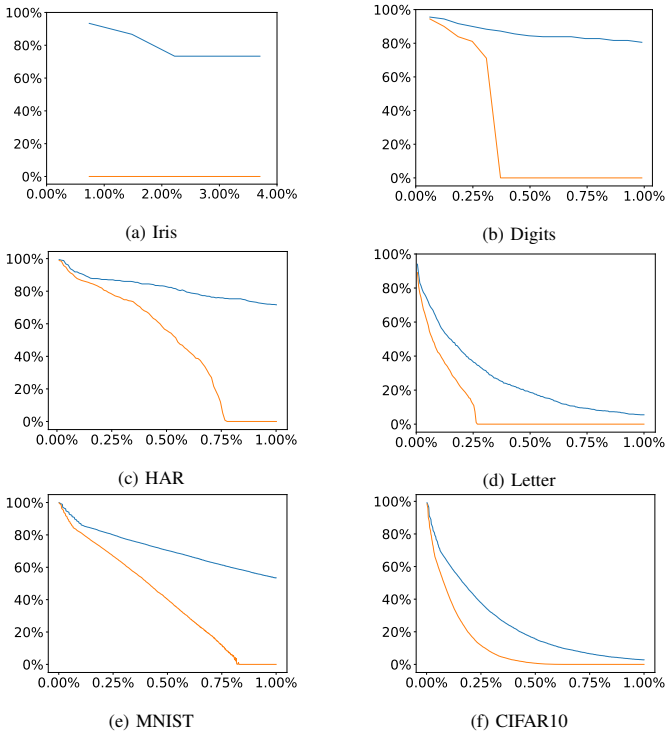


Fig. 6. Comparing our method (blue) with Jia et al. [21] (orange): the x-axis is polluted number n and the y-axis is the percentage of verified test data.

the verified percentage obtained by our method is significantly higher, due to its tighter approximations during the KNN inference phase. For all datasets, the verified percentage obtained by their method drops more quickly than the verified percentage obtained by our method. For *Iris*, in particular, their method cannot verify any of the test data, while our method can verify more than 70% of them as being robust.

VII. Related Work

There is a large body of work on verifying the (local) robustness of machine learning algorithms using formal methods. However, unlike most prior works which focus on adversarial examples in the context of deep neural networks, this work focuses on poisoned datasets for KNN. Unlike neural networks, for which scalability of the verification method typically depends on the network size but not the size of the training data, for KNN, scalability depends on the size of the training data and the number of poisoned elements.

In the context of robustness verification for KNN, our method is a method that can soundly verify n-poisoning robustness of the entire KNN algorithm, while existing methods such as Jia et al. [21] and others [39], [20], [40] are either restricted to a small part of what constitutes a state-of-the-art KNN system or primarily theoretical (and thus not scalable). Since we follow the definition of n-poisoning robustness in Drews et al.[12] instead of Jia et al. [21], our method only handles the removal of elements from already-polluted datasets, but not addition/modification of elements for clean datasets. Extending our method to handle such cases will be future work.

In addition to this line of research, there is a large body of work on adversarial data poisoning in general.

Data Poisoning in General KNN is not the only type of machine learning techniques found vulnerable to adversarial data poisoning; prior work shows that regression models [29], support vector machines (SVM) [6], [43], [42], clustering algorithms [7], and neural networks [34], [37], [11], [45] are also vulnerable. Unlike our work, this line of research is primarily concerned with showing the security threats and identifying the poisoning sets, which is often formulated as a constrained optimization problem.

Mitigating Data Poisoning Techniques have been proposed to mitigate data poisoning for various machine learning algorithms [35], [38], [19], [13], [5]. There are also techniques [22], [28] for assessing the effectiveness of mitigation techniques such as data sanitization [22] and differentially-private countermeasures [28]. More recently, Bahri et al. [4] propose a method that leverages both KNN and a deep neural network to remove mislabeled data.

Certifying the Defenses Probabilistically There are techniques for certifying the defenses [32], [25] such that accuracy is guaranteed probabilistically. For example, Rosenfeld et al. [32] leverage randomized smoothing to guarantee test-time robustness to adversarial manipulation with high probability. Levine et al. [25] certify robustness of a defense by deriving a lower bound of classification error, which relies on their deep partition aggregation (DPA) learning and is not applicable to typical learning approaches.

Leveraging KNN for Attacks or Defenses Orthogonal to our work, there are techniques that leverage KNN to generate attacks or provide defenses for other machine learning models. For example, Li et al. [26] present a data-poisoning attack that leverages KNN to maximize the effectiveness of malicious behavior while mimicking the user's benign behavior. Peri et al. [31] use KNN to defend against adversarial input based attacks, although it focuses only on tweaking the test input during the inference phase.

VIII. Conclusions

We have presented the first method for soundly verifying n-poisoning robustness for the entire KNN algorithm that includes both the learning (K parameter tuning) and the inference (classification) phases. It relies on sound overapproximations to exhaustively and yet efficiently cover the astronomically large number of possible adversarial scenarios. We have demonstrated the accuracy and efficiency of our method, and its advantages over a state-of-the-art method, through experimental evaluation using both small and large supervised-learning datasets. Besides KNN, our method for soundly over-approximating p-fold cross validation may be used to analyze similar cross-validation steps frequently used in other modern machine learning systems.

References

[1] D. A. Adeniyi, Z. Wei, and Y. Yongquan, "Automated web usage data mining and recommendation system using k-nearest neighbor (KNN) classification method," *Applied Computing and Informatics*, vol. 12, no. 1, pp. 90–108, 2016.

[2] M. Andersson and L. Tran, "Predicting movie ratings using KNN," 2020.

[3] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "A public domain dataset for human activity recognition using smartphones." in *Esann*, vol. 3, 2013, p. 3.

[4] D. Bahri, H. Jiang, and M. Gupta, "Deep k-nn for noisy labels," in *International Conference on Machine Learning*, 2020, pp. 540–550.

[5] B. Biggio, I. Corona, G. Fumera, G. Giacinto, and F. Roli, "Bagging classifiers for fighting poisoning attacks in adversarial classification tasks," in *International Workshop on Multiple Classifier Systems*, 2011, pp. 350–359.

[6] B. Biggio, B. Nelson, and P. Laskov, "Poisoning attacks against support vector machines," in *International Conference on Machine Learning*, 2012.

[7] B. Biggio, K. Rieck, D. Ariu, C. Wressnegger, I. Corona, G. Giacinto, and F. Roli, "Poisoning behavioral malware clustering," in *Workshop on Artificial Intelligent and Security*, 2014, pp. 27–36.

[8] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, "Targeted backdoor attacks on deep learning systems using data poisoning," *arXiv preprint arXiv:1712.05526*, 2017.

[9] P. Cousot and R. Cousot, "Abstract interpretation frameworks," *Journal of Logic and Computation*, vol. 2, no. 4, pp. 511–547, 1992.

[10] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *International Conference on Computer Vision and Pattern Recognition*, 2005, pp. 886–893.

[11] A. Demontis, M. Melis, M. Pintor, M. Jagielski, B. Biggio, A. Oprea, C. Nita-Rotaru, and F. Roli, "Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks," in *USENIX Security Symposium*, 2019, pp. 321–338.

[12] S. Drews, A. Albarghouthi, and L. D'Antoni, "Proving data-poisoning robustness in decision trees," in *ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2020, pp. 1083–1097.

[13] J. Feng, H. Xu, S. Mannor, and S. Yan, "Robust logistic regression and classification," *Advances in Neural Information Processing Systems*, pp. 253–261, 2014.

[14] I. Firdausi, A. Erwin, A. S. Nugroho *et al.*, "Analysis of machine learning techniques used in behavior-based malware detection," in *International Conference on Advances in Computing, Control, and Telecommunication Technologies*, 2010, pp. 201–203.

[15] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals of eugenics*, vol. 7, no. 2, pp. 179–188, 1936.

[16] P. W. Frey and D. J. Slate, "Letter recognition using holland-style adaptive classifiers," *Machine learning*, vol. 6, no. 2, pp. 161–182, 1991.

[17] G. Gates, "The reduced nearest neighbor rule (corresp.)," *IEEE Transactions on Information Theory*, vol. 18, no. 3, pp. 431–433, 1972.

[18] G. Guo, H. Wang, D. Bell, Y. Bi, and K. Greer, "KNN model-based approach in classification," in *International Conferences On the Move to Meaningful Internet Systems*, 2003, pp. 986–996.

[19] M. Jagielski, A. Oprea, B. Biggio, C. Liu, C. Nita-Rotaru, and B. Li, "Manipulating machine learning: Poisoning attacks and countermeasures for regression learning," in *IEEE Symposium on Security and Privacy*, 2018, pp. 19–35.

[20] J. Jia, X. Cao, and N. Z. Gong, "Intrinsic certified robustness of bagging against data poisoning attacks," *arXiv preprint arXiv:2008.04495*, 2020.

[21] ——, "Certified robustness of nearest neighbors against data poisoning attacks and backdoor attacks," in *AAAI Conference on Artificial Intelligence*, 2022.

[22] P. W. Koh, J. Steinhardt, and P. Liang, "Stronger data poisoning attacks break data sanitization defenses," *arXiv preprint arXiv:1811.00741*, 2018.

[23] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," *Technical Report, University of Toronto*, 2009.

[24] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[25] A. Levine and S. Feizi, "Deep partition aggregation: Provable defense against general poisoning attacks," *arXiv preprint arXiv:2006.14768*, 2020.

[26] B. Li, Y. Wang, A. Singh, and Y. Vorobeychik, "Data poisoning attacks on factorization-based collaborative filtering," *arXiv preprint arXiv:1608.08182*, 2016.

[27] Y. Li, B. Fang, L. Guo, and Y. Chen, "Network anomaly detection based on TCM-KNN algorithm," in *ACM symposium on Information, Computer and Communications Security*, 2007, pp. 13–19.

[28] Y. Ma, X. Zhu, and J. Hsu, "Data poisoning against differentially-private learners: Attacks and defenses," *arXiv preprint arXiv:1903.09860*, 2019.

[29] S. Mei and X. Zhu, "Using machine teaching to identify optimal training-set attacks on machine learners," in *AAAI Conference on Artificial Intelligence*, 2015.

[30] F. A. Narudin, A. Feizollah, N. B. Anuar, and A. Gani, "Evaluation of machine learning classifiers for mobile malware detection," *Soft Computing*, vol. 20, no. 1, pp. 343–357, 2016.

[31] N. Peri, N. Gupta, W. R. Huang, L. Fowl, C. Zhu, S. Feizi, T. Goldstein, and J. P. Dickerson, "Deep k-nn defense against clean-label data poisoning attacks," in *European Conference on Computer Vision*, 2020, pp. 55–70.

[32] E. Rosenfeld, E. Winston, P. Ravikumar, and Z. Kolter, "Certified robustness to label-flipping attacks via randomized smoothing," in *International Conference on Machine Learning*, 2020, pp. 8230–8241.

[33] A. Schwarzschild, M. Goldblum, A. Gupta, J. P. Dickerson, and T. Goldstein, "Just how toxic is data poisoning? A unified benchmark for backdoor and data poisoning attacks," in *International Conference on Machine Learning*, M. Meila and T. Zhang, Eds., 2021.

[34] A. Shafahi, W. R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein, "Poison frogs! targeted clean-label poisoning attacks on neural networks," *arXiv preprint arXiv:1804.00792*, 2018.

[35] J. Steinhardt, P. W. Koh, and P. Liang, "Certified defenses for data poisoning attacks," *arXiv preprint arXiv:1706.03691*, 2017.

[36] M.-Y. Su, "Real-time anomaly detection systems for denial-of-service attacks by weighted k-nearest-neighbor classifiers," *Expert Systems with Applications*, vol. 38, no. 4, pp. 3492–3498, 2011.

[37] O. Suciu, R. Marginean, Y. Kaya, H. Daume III, and T. Dumitras, "When does machine learning FAIL? generalized transferability for evasion and poisoning attacks," in *USENIX Security Symposium*, 2018, pp. 1299–1316.

[38] B. Tran, J. Li, and A. Madry, "Spectral signatures in backdoor attacks," *arXiv preprint arXiv:1811.00636*, 2018.

[39] Y. Wang, S. Jha, and K. Chaudhuri, "Analyzing the robustness of nearest neighbors to adversarial examples," in *International Conference on Machine Learning*, 2018, pp. 5133–5142.

[40] M. Weber, X. Xu, B. Karlas, C. Zhang, and B. Li, "Rab: Provable robustness against backdoor attacks," *arXiv preprint arXiv:2003.08904*, 2020.

[41] W. Wu, W. Zhang, Y. Yang, and Q. Wang, "Drex: Developer recommendation with k-nearest-neighbor search and expertise ranking," in *Asia-Pacific Software Engineering Conference*, 2011, pp. 389–396.

[42] H. Xiao, H. Xiao, and C. Eckert, "Adversarial label flips attack on support vector machines." in *ECAI*, 2012, pp. 870–875.

[43] H. Xiao, B. Biggio, B. Nelson, H. Xiao, C. Eckert, and F. Roli, "Support vector machines under adversarial label contamination," *Neurocomputing*, vol. 160, pp. 53–62, 2015.

[44] M. Xie, J. Hu, S. Han, and H.-H. Chen, "Scalable hypergrid k-NN-based online anomaly detection in wireless sensor networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 8, pp. 1661–1670, 2012.

[45] C. Zhu, W. R. Huang, H. Li, G. Taylor, C. Studer, and T. Goldstein, "Transferable clean-label poisoning attacks on deep neural nets," in *International Conference on Machine Learning*, 2019, pp. 7614–7623.