Detecting and Mitigating Sampling Bias in Cybersecurity with Unlabeled Data

Saravanan Thirumuruganathan, *Independent Researcher;* Fatih Deniz, Issa Khalil, and Ting Yu, *Qatar Computing Research Institute, HBKU;* Mohamed Nabeel, *Palo Alto Networks;* Mourad Ouzzani, *Qatar Computing Research Institute, HBKU*

https://www.usenix.org/conference/usenixsecurity24/presentation/thirumuruganathan

This paper is included in the Proceedings of the 33rd USENIX Security Symposium.

August 14–16, 2024 • Philadelphia, PA, USA

978-1-939133-44-1

Detecting and Mitigating Sampling Bias in Cybersecurity with Unlabeled Data

Saravanan Thirumuruganathan[†], Fatih Deniz[‡], Issa Khalil[‡], Ting Yu[‡],
Mohamed Nabeel[*], Mourad Ouzzani[‡]
[†]*Independent Researcher*
[‡]*Qatar Computing Research Institute, HBKU*
[*]*Palo Alto Networks*

Abstract

Machine Learning (ML) based systems have demonstrated remarkable success in addressing various challenges within the ever-evolving cybersecurity landscape, particularly in the domain of malware detection/classification. However, a notable performance gap becomes evident when such classifiers are deployed in production. This discrepancy, often observed between accuracy scores reported in research papers and their real-world deployments, can be largely attributed to *sampling bias*. Intuitively, the data distribution in the production differs from that of training resulting in reduced performance of the classifier. How to deal with such sampling bias is an important problem in cybersecurity practice. In this paper, we propose principled approaches to *detect* and *mitigate* the adverse effects of sampling bias. First, we propose two simple and intuitive algorithms based on domain discrimination and distribution of k-th nearest neighbor distance to detect discrepancies between training and production data distributions. Second, we propose two algorithms based on the *self-training* paradigm to alleviate the impact of sampling bias. Our approaches are inspired by domain adaptation and judiciously harness the unlabeled data for enhancing the generalizability of ML classifiers. Critically, our approach does not require any modifications to the classifiers themselves, thus ensuring seamless integration into existing deployments. We conducted extensive experiments on four diverse datasets from malware, web domains, and intrusion detection. In an adversarial setting with large sampling bias, our proposed algorithms can improve the F-score by as much as 10-16 percentage points. Concretely, the F-score of a malware classifier on AndroZoo dataset increases from 0.83 to 0.937.

1 Introduction

Machine learning (ML) based systems have achieved tremendous success in diverse cybersecurity tasks such as malware detection, detecting malicious URLs, and vulnerability analysis, among others [17, 20, 53, 54]. However, when these systems are deployed in production, their performance does not match the lofty performance that is reported in the research papers [43, 62]. One of the central assumptions in ML is that the training and test data are drawn identically and independently (i.i.d.) from the same underlying data distribution [48]. When this assumption is violated, intentionally or inadvertently, the performance of the classifiers goes down. In the worst case, this renders the ML systems unsuitable for cybersecurity applications that have stringent performance requirements.

The discrepancy between training and deployment data distributions is especially high in the cybersecurity domain where the classifiers are deployed in an adversarial environment. For example, adversaries could modify the malware code to bypass detection. Not surprisingly, there has been extensive work [8, 13, 27, 60, 62] on ensuring that the performance of ML classifiers is stable. However, almost all of the prior work focuses on *concept drift* [26]. In this setting, the deployment data distribution slowly *diverges* from the training data distribution over time resulting in performance degradation of the classifier. However, our empirical analysis over multiple datasets from diverse tasks shows the discrepancy arises much earlier in the ML workflow.

Sampling Bias. We use the term *sampling bias* to represent the generic phenomenon where the distributions for training and deployment data are not identical. Hence, the trained classifier provides suboptimal performance in a production setting even when the classifier is freshly deployed (unlike the gradual degradation in concept drift). While some prior work has observed this phenomenon, they often have a narrow focus. For example, Tesseract [38] considers spatial bias where the ratio of goodware to malware in the data varies. In contrast, we consider a generic setting where the discrepancy between training and deployment data distributions is arbitrary. Training ML models in cybersecurity has a number of subtle pitfalls that make this discrepancy likely. For example, the bias could be due to using a benchmark dataset that is no longer representative of the real-world settings (such as using a dataset on Google Play store to design a classifier for Chinese app store or using a dataset from a particular geographical region to another [10, 11]). Alternatively, this could be due to the various shortcuts taken in data collection. Given the high cost of labeling, many researchers often use convenience sampling [51] where the collected data is not representative of the production data. For example, URLs from ".edu" or ".gov" domains or popular websites from Tranco might be overrepresented in a dataset as they are likely to be benign. Another pitfall arises from the labeling heuristics. For example, a researcher could label a data item as malicious if at least k (e.g., 10) engines label from VirusTotal [34] report it as malicious. Similarly, every data item that none of the engines report as malicious (i.e., $k = 0$) could be categorized as benign. While intuitive, this has a subtle issue where the data items with 1 to $k-1$ suspicious labels are underrepresented. Finally, indiscriminate mixing of incompatible datasets – such as apps from Google Play or Chinese app stores – could also cause bias in the training data. Of course, these scenarios are not comprehensive [21]. The adversarial and dynamic nature of cybersecurity and the high cost of accurate labeling necessitates the use of heuristics that exacerbate the sampling bias. It is indisputable that sampling bias is a persistent issue with ML classifiers in the cybersecurity domain [11, 43]. While it is almost impossible to completely eliminate sampling bias, in this paper, we propose novel algorithms that can reduce the performance penalty caused by the bias. Concretely, we make two major contributions - *detection* of sampling bias and *mitigation* of its impact.

Detecting Sampling Bias. ML workflows are often complex and filled with various subtle pitfalls that can result in sampling bias. It is infeasible to make every cybersecurity practitioner well-versed in sampling theory. We focus on the practical problem where we are given a labeled training dataset D_T and an *unlabeled* deployment dataset D_U. Our goal is to detect if this training setup produces a biased classifier. We propose two complementary techniques for bias detection. The first approach is based on the concept of *domain discrimination*. Intuitively, we train a classifier to determine if a data item belongs to D_T or D_U. If the accuracy of this classifier is low (such as 0.5), then D_T and D_U are indistinguishable which implies a low sampling bias. In contrast, a high accuracy means that D_T and D_U are substantially different and likely will result in a biased classifier. Our second approach detects bias based on the distribution of the k-th nearest neighbor distances between D_T and D_U.

Mitigating the Impact of Sampling Bias. Our second contribution consists of two novel algorithms that can mitigate the impact of sampling bias. Given D_T and D_U, our goal is to train a classifier $\mathcal{C}'$ that has a higher performance than the biased classifier $\mathcal{C}$ that is trained only on D_T. Intuitively, we achieve this by judiciously using the unlabeled data from D_U. In order to increase the adoption of our approach, we ensure that the classifier $\mathcal{C}'$ uses the same training regimen as that of $\mathcal{C}$. Concretely, we use the self-training paradigm where the $\mathcal{C}'$ is trained on $D_T \cup D_U^P$, where $D_U^P \subseteq D_U$. Since D_U is unlabeled, we need to obtain the *pseudo-labels* for D_U^P. Instead of obtaining labels for a data item $t \in D_U^P$ using a domain expert, we obtain approximate (pseudo) labels using D_T. We can see that if these pseudo labels are inaccurate, the classifier $\mathcal{C}'$ will have a lower performance than $\mathcal{C}$. Hence, the key challenge here is to design a procedure for obtaining accurate pseudo labels.

We tackle this challenge by designing algorithms that produce better representations for data items in D_T and D_U. Our first algorithm uses contrastive learning [22] for learning the representations. This method is trained such that the representations of similar data items are pushed together while the representations of dissimilar items are pulled apart. The key challenge is to identify similar items without having the label information. Our second algorithm is based on an iterative approach. We train a classifier C_T on D_T and use its predictions as the pseudo label for D_U. Next, we train a classifier C_U on D_U using the pseudo labels and use its predictions to obtain pseudo labels for D_T. The key insight is to use a common encoder for producing representations such that the classifiers C_T and C_U are quite accurate.

Experimental Highlights. We conduct extensive experiments to demonstrate the efficacy of our proposed algorithms. Most of our experiments focus on the malware detection domain. We consider an especially challenging setting where the classifier is trained on one dataset D_T (such as Ember [5]) and then evaluated on a related but sufficiently different dataset D_{test} (such as UCSB-Packed [1] or BODMAS [61]). We also investigate a further adversarial setting where the the classifier is evaluated on the most challenging tuples from D_{test}. As we shall show in the experiments, a classifier C_T that is trained and tested on UCSB achieves an F-score of 0.986. A classifier C_T that is trained on Ember but tested on UCSB achieves an F-score of 0.725. The low performance is to be expected as Ember and USCB has different data distributions. However, our proposed algorithms were able to bridge most of the differences in the performance. Concretely, the classifier based on ConL-BM achieves an F1-score of 0.916 while CyC-BM achieves an F1-score of 0.938. We also demonstrate the generalizability of our algorithms by conducting additional experiments over domains and intrusion detection.

2 Preliminaries

Datasets. We have a labeled training dataset $D_T = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ with N data items that are drawn i.i.d. from a data distribution P. A classifier $\mathcal{C}$ that is trained on D_T is then deployed in a production setting where the data items are drawn i.i.d. from a data distribution Q. Ideally, the joint distribution of the training and deployment data are identical, i.e. $P(x, y) = Q(x, y)$. Very often, methodological issues in collecting dataset D_T produces sampling bias resulting in the divergence of their data distributions, i.e. $P(x,y) \neq Q(x,y)$. When distributions P and Q diverge, any classifier $\mathcal{C}$ trained on D_T will provide sub-optimal performance when evaluated on a testing dataset D_{test} drawn i.i.d. from Q.

We assume the availability of an *unlabeled* dataset $D_U = \{x_1, x_2, \ldots, x_M\}$ with M data items that are drawn i.i.d. from Q. Intuitively, D_U is representative of the deployment data that will be seen by a classifier in a production setting. We can see that, any classifier that achieves high accuracy on D_U will also have a high accuracy in a production setting (and also on D_{test}). While it is desirable to have D_U and D_{test} the same distribution Q, this strict assumption is not always necessary. It is sufficient that D_U is 'closer' to D_{test} than D_T which also ensures that D_U is less biased than D_T. We do not assume that D_T is similar to D_U (or D_{test}).

Sampling Bias. In this paper, we use the *total survey error* (TSE) framework proposed by [21]. Specifically, we focus on *representation error* which corresponds to the divergence between a selected sample and the target population of interest. There are three key sources for this type of error. *Coverage error* occurs when the *sampling frame* is not the same as the target population. Even though the target population is the set of all URLs, millions of URLs that are internal to some enterprise are never exposed to the outside (such as to VirusTotal). Hence the sampling frame consists of public URLs which is only a subset of the target population. A *sampling error* occurs when the subset of data items chosen from the sampling frame is not representative. Given the set of all public URLs, one could preferentially select URLs from ".edu" or ".gov" domains, such as for ease of labeling. Hence the selected sample is not representative of the sampling frame resulting in sampling error. Finally, a *non-response error* occurs if a non-uniform subset of the observed sample is excluded from the final statistical sample. For example, the domain expert could use a labeling heuristic by passing all the URLs from the observed sample to VirusTotal. Then, she could label the URLs that are not flagged by any engine as benign and label those that are flagged by k (e.g. $k = 10$) engines as malicious. However, this has resulted in the drop-off of URLs that were flagged by 1 to $k-1$ engines causing non-responsive errors. Our algorithms can handle bias injected due to common types of representation errors.

Problem Definition. Our goal is to train a less biased classifier $\mathcal{C}'$ that achieves higher performance on D_{test} than a classifier $\mathcal{C}$ trained on D_T. We achieve this by leveraging D_U. Concretely, we construct a new dataset $D = D_T \cup D_U^P$ where $D_U^P \subseteq D_U$. Since D_U is unlabeled, we judiciously select a subset D_U^P for which relatively accurate (pseudo) labels can be obtained. When one trains a classifier $\mathcal{C}'$ on D using the same training regimen as $\mathcal{C}$, it will have a higher accuracy than $\mathcal{C}$.

Problem Scope. Our algorithms are designed to detect and mitigate sampling bias injected due to common types of *representation errors*. Furthermore, we focus on sampling bias that exists between datasets D_T^t and D_{test}^t for a single snapshot of time t. When the classifier has to be retrained (such as after a week for time $t+\Delta$), then our approach can be used between datasets $D_T^{t+\Delta}$ and $D_{test}^{t+\Delta}$. Our approach can be used in a dynamic setting under limited circumstances. For example, one could apply our bias detection algorithm to detect discrepancies between D_T^t and the unlabeled datasets D_{test} seen in production for time periods $t+\Delta, t+2\times\Delta, \ldots$. When the discrepancy crosses some threshold δ, then the classifier can be retrained using our bias mitigation algorithms. Our bias *mitigation* algorithms work best when the datasets D_T, D_U and D_{test} are all collected from *approximately* the same time period t. When this assumption is violated, it is preferable to use other specialized algorithms such as concept drift [8] or continual learning [14].

Collecting Unlabeled Dataset D_U. Each of our proposed algorithms for the detection and mitigation of bias assumes the existence of a less biased and unlabeled dataset D_U. Since D_U can be unlabeled, it is often easier to collect. For example, one could collect a set of domains using passive DNS or from daily logs of VirusTotal. One could collect a set of mobile apps by uniformly sampling Google Play or Apple App stores. Another common source for D_U would be to use recent data from production (such as from last week). Regardless of the collection procedure, it is desirable that D_U is representative of the data items that will be seen by the classifier in the production setting than D_T. The closer D_U is to the production setting, the higher the performance improvement that could be obtained by our algorithms. In contrast, if D_U is dissimilar to D_{test}, then our algorithms (or any ML algorithm for that matter) would provide sub-optimal results.

3 Background

Our paper synthesizes ideas from diverse fields. We provide a basic overview of the ideas that are relevant to the paper.

Unsupervised Domain Adaptation (UDA). Traditional supervised learning relies on the assumption that the training and test data are drawn from the same distribution. When this assumption is violated, a classifier trained on the training data (also called the *source* domain) can provide inferior performance when evaluated on the test data (also called the *target* domain). The field of domain adaptation seeks to learn a classifier using the source data that also performs well on the target data. Of interest to us is the *unsupervised* domain adaptation where we have *labeled* source data and *unlabeled* target data. Different UDA techniques leverage the unlabeled data in different ways to achieve improved performance. A popular technique is *domain alignment* where the goal is to minimize the discrepancy between the two domains. This can be achieved by *reweighting* [56] the source samples so that its distribution is closer to the target data distribution. Alternatively, the samples from both domains can be transformed to a common latent space through *domain mapping* using a learned encoder. However, both of these methods are not appropriate for our setting. The efficacy of reweighting drops off steeply as the divergence between the two domains increases. Domain mapping is not applicable as it typically requires using a completely different set of classifiers that are capable of learning an appropriate latent space.

Self-Training. Recently, self-training has become the dominant paradigm for UDA [4]. Most of the self-training classifiers [59, 64, 65] are iterative and perform two steps in each iteration: (a) create a set of pseudo-labels (i.e., approximate and possibly incorrect labels) for the target domain; and (b) train a classifier using the generated pseudo-labels (and the source data). Different algorithms vary in how pseudo-labels are generated and/or how the classifier is trained. Using a naive method will generate noisy pseudo-labels resulting in a classifier with bad performance. The vast majority of the algorithms using self-training for UDA are developed for computer vision and are non-trivial to adapt to the cybersecurity setting. The key issue is that many of these techniques rely on augmentations - which are label-preserving transformations. For example, if the prediction of a classifier for an image I and (one or more of) its augmentations (such as rotation, random cropping) are the same, then it is plausible to think that the classifier is likely to be correct about the prediction for I. However, it is not obvious how to design such label-preserving transformations for cybersecurity applications.

4 Detection of Sampling Bias

In this section, we describe two principled algorithms for detecting sampling bias caused by representation errors.

Problem Scope. We are given a training dataset D_T and an unlabeled dataset D_U. Our goal is to detect sampling bias between D_T and D_U. Concretely, we are interested in detecting *malignant* sampling bias such that a classifier trained on D_T has a sub-optimal performance on D_U.

A desirable preprocessing step is to ensure that the cardinalities of D_T and D_U are approximately the same. If not, an uniformly random sample from the larger dataset has to be obtained with the same cardinality as the smaller dataset. This ensures that bias detection operates in a 'balanced' setting where neither D_U nor D_T dominate each other. Traditional ML models produce poor results over imbalanced data unless sophisticated balance correction techniques such as reweighting are applied. For example, if $|D_U|/(|D_U|+|D_T|) = 0.66$ (i.e. D_U is twice as large as D_T), then the model has to be trained on a weighted variant of the dataset where tuples from the smaller dataset has a higher weight than the ones from the larger dataset (in this case 2 vs 1).

4.1 Domain Discrimination

Our first algorithm takes a direct approach by treating classifier accuracy as a proxy for the discrepancy between D_T and D_U. Suppose we train a classifier $\mathcal{C}_T$ on the labeled data D_T. Hypothetically, if we have accurate labels for D_U, we can compare the accuracy of $\mathcal{C}_T$ on a held-out dataset of D_T and D_U. If the accuracies are comparable, then it is likely that D_T and D_U have low sampling bias. A recent work [29] identified intriguing connections between using two-sample tests and classifier accuracy as a proxy for determining if two samples came from different distributions.

Our approach is based on the concept of *domain discrimination* [15, 63]. The key challenge in using the aforementioned approach is that we do not have the labels for D_U. Hence, we design an alternative classifier for discriminating two distributions. Concretely, we construct a dataset D by pooling both D_T and D_U. All the data items from D_T have a label of 1 and the data items from D_U have a label of 0. Using this labeled data, we train a logistic regression classifier $\mathcal{C}_D$. Given a data item t, this classifier can determine if t belongs to D_T or D_U. We choose logistic regression as it is efficient to train and since its output, i.e., $p(y=1|x)$, can be interpreted as a probability. If we use another classifier, it is important to calibrate their output so that it can be interpreted as a probability. Intuitively, we can see that the accuracy of the classifier is a good proxy for how different D_T and D_U are. If the accuracy is around 0.5 (comparable to a random coin toss), then it implies that the two distributions are practically indistinguishable. In contrast, if the accuracy is high then it implies that the two distributions are sufficiently different that even a logistic regression classifier could discriminate them. Algorithm 1 provides the pseudocode.

Algorithm 1 Domain Discrimination

```
**Input:** $D_T$, $D_U$ and threshold $\delta$
$D_T = \{(x,1) \quad \forall (x,y) \in D_T\}$
$D_U = \{(x,0) \quad \forall x \in D_U\}$
Randomly split $D_T$ into equal sized partitions $D_T^1$ and $D_T^2$
Randomly split $D_U$ into equal sized partitions $D_U^1$ and $D_U^2$
Train classifier $\mathcal{C}_D$ on $D_T^1 \cup D_U^1$
acc = Accuracy of $\mathcal{C}_D$ on $D_T^2 \cup D_U^2$
**Return** $acc > \delta$
```

4.2 k-NN based Bias Detection

Our second approach is complementary to the parametric approach taken by the domain discriminator. The key intuition is that if two data distributions are similar, then the distribution of the distance to the k-th nearest neighbor will also be similar. It is non-parametric and does not make any distributional assumption on the underlying feature space [52].

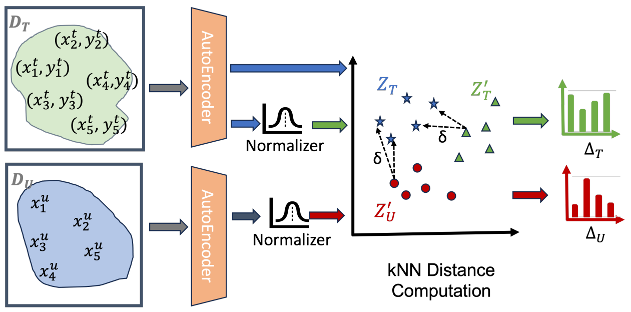

Figure 1: Computing *k*NN Distance Distribution.

Distribution of *k*NN Distance. Assume that we have an effective encoder $h_\phi(\cdot)$ that can take a feature vector x and output its embedding z. First, we apply the encoder on each data item in D_T and obtain the corresponding embedding. Let $Z_T = \{h_\phi(x) \quad \forall (x,y) \in D_T\}$ be the collection of all such embeddings. Given a data item x from D_U, we obtain the *normalized* embedding $z^* = h_\phi(x)/||h_\phi(x)||$. Let $\delta(z, Z, k)$ represents the distance of the k-th nearest neighbor of the normalized vector $z^* = z/||z||$ from the embedding dataset Z. We compute the k-th nearest neighbor distance distribution for both D_T and D_U. Formally,

$$\Delta_T = \{\delta\left(h_\phi(x), Z_T, k\right) \quad \forall x \in D_T\} \tag{1}$$

$$\Delta_U = \{\delta\left(h_\phi(x), Z_T, k\right) \quad \forall x \in D_U\} \tag{2}$$

Given the two distance distributions Δ_T and Δ_U, one could use any heuristic to check if they are similar. This can be achieved by a function $\mathcal{F}$ that takes Δ_T and Δ_U as input and outputs a scalar that is then compared with a threshold λ. A common and robust approach (that we also use in our experiments) would be to compare the distance between median values of Δ_T and Δ_U against a threshold λ. If it is above λ, then the distributions are different from each other and exhibit sampling bias. A domain expert could use other metrics such as the average or any percentile values.

Learning Embeddings. We use the supervised contrastive learning approach SupCon [28] for learning the embeddings. Intuitively, this approach ensures that normalized embeddings from the same class (benign or malicious) are grouped together in the embedding space. In contrast, embeddings of data items from different classes are pulled apart. Given a data item x, SupCon generates multiple positive and negative pairs by randomly selecting samples belonging to the same and different classes respectively. It also uses a label-aware contrastive loss resulting in effective embeddings. Algorithm 2 shows the pseudocode and Figure 1 illustrates the algorithm.

5 Mitigating Sampling Bias

In this section, we introduce some key ideas and assumptions that are relevant for designing algorithms that can mitigate sampling bias.

Algorithm 2 Distribution of k-th Nearest Neighbor Distance

```
Input: $D_T$, $D_U$ and threshold $\lambda$
Learn encoder $h_\phi$ using SupCon($D_T$)
$Z_T = \{h_\phi(x) \quad \forall x \in D_T\}$
$\Delta_T = \{\delta\left(h_\phi(x), Z_T, k\right) \quad \forall (x,y) \in D_T\}$
$\Delta_U = \{\delta\left(h_\phi(x), Z_T, k\right) \quad \forall x \in D_U\}$
Return $\mathcal{F}(\Delta_T, \Delta_U) > \lambda$
```

Generating Accurate Pseudo Labels. The fundamental challenge is to come up with a procedure to generate effective pseudo-labels. The success of any self-training method hinges on pseudo-label accuracy. We cannot use the self-training approaches designed for computer vision or tabular ML. Computer vision often relies on label-preserving transformations (a rotated picture of a cat has the label cat). It is challenging to come up with such general-purpose transformations that can work across multiple domains in cybersecurity. Another related area is that of tabular ML that creates transformations through controlled feature corruption (i.e. replacing a feature value from one tuple with the corresponding value from another randomly picked tuple). This approach is not label preserving as cybersecurity classifiers are more sensitive to individual feature values. Our solution is to sidestep the typical use of label-preserving transformation and instead rely on geometric aspects of ML classifiers over the latent space.

In Sections 6 and 7, we propose two novel algorithms based on the self-training paradigm. The output of these algorithms are the classifiers C_U that have been trained on using pseudo labels. Once we have the classifier C_U, we can then apply it on D_U to make predictions. Of course, it is unlikely that C_U is perfect. We will select a subset of $D_U^P \subseteq D_U$ where the classifier is most confident that is then used by the practitioner to train a classifier C_{final} on $D_T \cup D_U^P$. We would like to note that these assumptions holds on expectation. Hence, it is not necessary that all the tuples in the dataset satisfies these assumptions. A small fraction of the tuples can safely violate them. However, if a large proportion of the tuples violate them then no ML model can achieve good accuracy.

ML Assumptions. In our paper, we rely on *two* widely held assumptions in self-training and semi-supervised learning and use it to design better pseudo-labeling strategies. The first is the *continuity/smoothness* assumption [12] that posits that data items that are close to each other are very likely to share a label. Intuitively, this assumption results in classifiers that have geometrically simple decision boundaries. The second is the *cluster* assumption [25] that claims that data items tend to form discrete clusters and that data items within a cluster are likely to share a label. This does not imply that all the data items of a particular class form a single cluster – data items from the same class might form multiple discrete clusters, each satisfying the cluster assumption.

We can operationalize these assumptions by developing *representation learning* algorithms that transform data items from D_T and D_U into a latent space where the continuity and cluster assumptions hold. Concretely, our goal is to design a latent space such that decision boundaries lie in low-density regions and do not cross high-density regions (which will result in similar points having different labels violating both these assumptions).

Baseline Approaches. One might be tempted to design some simple techniques based on the ML assumptions stated above. For example, one could try to leverage continuity assumption by designing a k-NN-based classifier for obtaining pseudo labels for a data item $x \in D_U$. Since similar items are likely to share similar labels, we first select the k *labeled* data items from D_T that are most similar to x. Then we can use a strategy such as (distance-weighted) majority voting to estimate the pseudo label for x. We then repeat this process for each $x \in D_U$. Another approach would be to leverage the cluster assumption. One could use a hierarchical clustering algorithm such as hclust [37] over $D_T \cup D_U$ to cluster the data items and obtain dendrograms. Then we can use metrics such gap statistic [55] or cophenetic correlation [46] to *cut* the dendrograms into various clusters. Using the cluster assumption, any unlabeled data item is assigned the same class as the majority label of the cluster. While intuitive, both these approaches perform poorly in practice. The key issue is that the original feature space is often sub-optimal for using such simple techniques. In the next two sections, we propose two novel algorithms that leverage these assumptions and design a better latent space to obtain better pseudo labels.

6 Contrastive Learning for Bias Mitigation

In this section, we describe how ideas from contrastive learning can be combined with the two ML assumptions mentioned earlier to design effective pseudo-labeling strategies.

Improving Baseline Methods with Contrastive Learning. The k-NN and clustering-based baseline approaches do not work well. However, both approaches have a kernel of promising ideas whose potential can be fully exploited through contrastive learning. The k-NN classifier-based approach suffers from two issues. First, the k-NN is a non-parametric approach and hence results in overly complex decision boundaries. Such a decision boundary makes pseudo-labeling strategies more fragile. Second, the original feature space is often not conducive to applying the nearest neighbor-based classifiers. It is desirable to map the features of the data items into a new representation space (also called an embedding space). Ideally, this embedding space is semantically aware so that similar items (such as a pair of benign items) will be closer to each other. We can achieve this using contrastive learning. The clustering-based baseline suffers from a similar ailment. The original feature space is not well suited to semantic clustering resulting in a non-homogeneous set of data items within each cluster which then dilutes the accuracy of majority voting-based pseudo labels.

Contrastive Learning (CL) with Pseudo Labels. CL is trained using a set of positive and negative sample pairs. The goal is to design an encoder that converts each data item into an embedding such that the distance between embeddings of the positive sample pairs is minimized while the distance between embeddings of negative sample pairs is maximized. This ensures that the embeddings of similar items are closer to each other while those of dissimilar items are farther apart.

Our key insight is to use pseudo labels to produce the set of positive and negative sample pairs. Suppose an oracle provided us with reasonably accurate pseudo-labels (we will describe the labeling strategy shortly). Given a pair of data items, if they have the same pseudo labels, then we can treat them as a positive sample pair. Alternatively, data items with different pseudo labels will be considered negative sample pairs. Once we have the set of positive and negative sample pairs, we can use them to learn a contrastive encoder. By design, the learned embedding space will be more conducive to pseudo-labeling. Contrastive learning ensures that in the embedding space, similar items are closer to each other and share the same label (i.e., continuity assumption). More interestingly, the embedding space satisfies a simpler version of the cluster assumption where all data items related to a single class label form a *single* cluster. Recall that in the original space, it is possible that data items from the same label might form multiple independent clusters. However, the objective function of contrastive learning ensures that all the data items of a given label (benign or malicious) form a single cluster. We use soft nearest neighbor loss as the objective function for contrastive learning. Given a batch B of samples $\{(x_i, \hat{y}_i)\}$ where $\hat{y}_i$ is the (pseudo) label for x_i and a function $sim(\cdot,\cdot)$ that measures the similarity between two data items, the loss function is defined at temperature τ as

$$\mathcal{L}_{snn} = -\frac{1}{|B|}\sum_{i=1}^{|B|} \log \frac{\sum_{i\neq j, \hat{y}_i=\hat{y}_j} \exp(-sim(x_i,x_j)/\tau)}{\sum_{i\neq k} \exp(-sim(x_i,x_k)/\tau)} \tag{3}$$

Obtaining Pseudo Labels. The simplified geometry of the embedding space from contrastive learning allows us to use relatively simple and intuitive pseudo-labeling strategies. We do not desire to use the k-NN-based classifier as it results in overly complex and fragile decision boundaries. Our approach is based on two steps. First, given the labeled data D_T, we identify class *prototypes* – exemplars of data items that are representative of a given class. Second, for each item $x \in D_U$, we estimate the pseudo label using the nearest neighbor strategy. So if x is closer to the benign class prototype, then it is assigned a pseudo label as benign (and vice versa). We use cosine distance to measure the similarity between the embedding of the class prototype and that of x.

There are many methods to compute the class prototypes. A simple approach would be to compute the class centroid. Recall that we know the labels for each data item $x \in D_T$. So, we begin by partitioning D_T based on the class labels. In the case of a binary classification problem, we will have two partitions – one corresponding to benign items and another for malicious items. We compute the class centroid by computing the average feature vector of all the data items from that cluster. Of course, the centroid produced using the original feature vector could produce sub-optimal centroids as the data items for a single class could belong to distinct clusters in the feature space. A better alternative is to use the contrastive encoder that can produce better clusters. Given a partition (such as for benign items), we compute the embedding of each data item by feeding it to the encoder and then compute the centroid as the average of the embeddings. We can see that the embedding centroid is more robust and a better prototype.

Algorithm 3 provides the pseudocode. One additional detail that can be in the pseudocode is that we also use the learned contrastive encoder to train a classifier $\mathcal{C}$ with parameters θ using the labeled data from D_T. We use the standard cross entropy loss to measure its accuracy. This ensures that the learned embeddings are good for both pseudo-labeling and training a classifier. The whole process has to be iterated multiple times before the encoder produces good-quality embeddings. Initially, the encoder is randomly initialized and hence produces low-quality class prototypes resulting in poor pseudo labels. However, as the quality of the encoder improves, the accuracy of the pseudo-labeler also increases.

Algorithm 3 Contrastive Bias Mitigation

```
Input: $D_T$ and $D_U$
Randomly initialize parameters $\phi$ of contrastive encoder $h$
for epoch = 1 to max epoch do
    Compute class prototypes from $D_T$ using $h_\phi$
    Compute pseudo labels $\hat{y}_i$ for each $x_i \in D_U$ using their distance to the class prototypes
    Form positive and negative sample pairs using $\hat{y}_i$ and compute $\mathcal{L}_{snn}$
    Train classifier $\mathcal{C}$ on embeddings from $h(D_T)$ and compute cross-entropy loss $\mathcal{L}_{CE}$
    $\mathcal{L} = \mathcal{L}_{snn} + \mathcal{L}_{CE}$
    Back propagate and update parameters $\phi$ of $h$ and $\theta$ of $\mathcal{C}$
end for
```

7 Bias Mitigation using Cycle Consistency

In this section, we propose a novel algorithm for mitigating sampling bias. Our approach is based on the concept of cyclic consistency loss – where our goal is to train an encoder such that it learns an effective mapping between D_T and D_U. Once both D_T and D_U are mapped into the same representation space, then we can design accurate pseudo-labeling strategies.

Measuring Pseudo Label Accuracy. A key challenge in self-training is to get an estimate of how accurate the pseudo labels are. If the pseudo labels are quite accurate, then self-training works well in practice. If not, the training regimen often enters a doom loop where the quality continuously deteriorates. Hence, an important sub-problem is to get an indication of the quality of pseudo labels at any stage in the classifier training. Unfortunately, solving this problem is not trivial as we do not have any labels for D_U. In this section, we introduce a simple idea that can address this issue. We make two key changes. First, instead of using a single classifier for self-training, we use a pair of inter-related classifiers – one trained over D_T and one trained over D_U. Second, this two-step process provides a proxy for measuring the accuracy of pseudo labels. Intuitively, we use a cyclic approach where we train a classifier $\mathcal{C}_T$ on D_T and use it to obtain pseudo labels for D_U. In reverse, we train a classifier $\mathcal{C}_U$ on D_U and use it to obtain pseudo labels for D_T. However, since we have accurate labels for D_T, we can compare the pseudo labels with the actual labels. This allows us to indirectly evaluate the accuracy of the generic pseudo-labeling strategy. In the rest of the section, we flesh out the relevant details.

Common Setup. A key factor that is necessary for the success of the cyclic training approach is that of a shared encoder h_ϕ whose parameters are shared between $\mathcal{C}_T$ and $\mathcal{C}_U$. In other words, the inputs to both the classifiers are passed first to the encoder h_ϕ to obtain the embeddings. The predictions of both the classifiers are then performed over the embeddings. It is essential to have the shared encoder as it ensures that the latent embedding space for data items from both D_T and D_U is aligned resulting in better pseudo labeling.

Forward Step. We begin by training a source classifier $\mathcal{C}_T$ using D_T (on top of shared representation $h_\phi(D_T)$). Once the classifier is trained, we can then use it to obtain pseudo labels for each data item $x \in D_U$. For simplicity, we use the arg max approach where we select the class label produced by $\mathcal{C}_T$ with the highest confidence. In a typical self-training setup, we will select a subset of the most confident pseudo-labels and then use it to retrain the classifier in the next iteration. However, one could design a better algorithm if we have some indication of the accuracy of the generated pseudo labels. We make a simple observation. The data items $x \in D_U$ that are likely to have accurate pseudo labels are those that are most similar to data items from D_T. In other words, if an unseen data item is similar to one that is seen in the training data, then it is likely to have the right prediction (and hence accurate pseudo-label). As a corollary, this also means that once the shared encoder aligns the two distributions D_T and D_U, the accuracy of the pseudo labels becomes proportional to the accuracy of the classifier $\mathcal{C}_T$ on D_T. The contra-positive of this observation motivates us to design the reverse step. Suppose that we train a classifier $\mathcal{C}_U$ and use it to generate pseudo labels for D_T.

Using the same line of reasoning, we can argue that the data items from D_T that have accurate pseudo labels are those that are most similar to those from D_U. However, since we do have access to the actual labels of D_T, we can use it to estimate the level of alignment between D_T and D_U.

Reverse Step. Using the classifier $\mathcal{C}_T$, we first obtain the pseudo labels for all data items $x \in D_U$. Next, we train a classifier $\mathcal{C}_U$ using the pseudo labeled data D_U. Our goal is twofold. First, we can see that the accurate pseudo labels from D_U have the ability to produce accurate pseudo labels for D_T. Second, by forcing the pseudo labels of D_U (and thereby $\mathcal{C}_T$) to be more accurate for source domain, we can indirectly ensure that their accuracy on the target domain D_U is improved. In other words, by forcing the classifier $\mathcal{C}_T$ to work well on D_T, it iteratively improves upon the learned embeddings and thereby the learned pseudo labels. After a few iterations, this cyclic consistency constraint results in the alignment of D_T and D_U which results in more accurate pseudo labeling for both D_T and D_U. Finally, we can monitor the progress of the learning by measuring the accuracy of the pseudo labels for D_T. Algorithm 4 provides the pseudocode. An important thing to note is that we perform backpropagation to update the parameters for the shared encoder and the classifier $\mathcal{C}_U$ and not $\mathcal{C}_T$. In other words, our goal is to keep updating the parameters so that the classifier performs well on the source domain. Empirically, we found that this produces better results than improving $\mathcal{C}_T$.

Algorithm 4 Bias Mitigation with Cycle Consistency

```
Input: $D_T$ and $D_U$
Randomly initialize parameters $\phi$ of shared encoder $h$
for epoch = 1 to max epoch do
    Train $\mathcal{C}_T$ on $D_T$ and generate pseudo labels for $D_U$
    Train classifier $\mathcal{C}_U$ using $D_U$ and pseudo labels
    Apply $\mathcal{C}_U$ on $D_T$ and generate pseudo labels
    Estimate accuracy of pseudo labels using labeled data $Y_T$ from $D_T$
    Back propagate and update parameters of $h_\phi$ and $\mathcal{C}_U$ using $Y_T$
end for
```

8 Experiments

In this section, we describe the results of the extensive experiments that show the efficacy of our proposed algorithm for detecting and mitigating sampling bias.

8.1 Experimental Setup

All our experiments are conducted on widely used benchmark datasets from three different domains to demonstrate the efficacy and utility of our algorithms. Concretely, our goal is to show that our methods work well and produce non-trivial improvement in performance even in the case of well-studied datasets where such heuristics have been exhaustively studied.

Training and Deployment Datasets. We are provided with a training dataset D_T. Our goal is to train a classifier $\mathcal{C}$ that can work well in production D_{Test} by leveraging an *unlabeled* dataset D_U. In a realistic setting, the classifier $\mathcal{C}$ is applied to production data, and its performance is unknown as the production data is unlabeled. Typically, the domain expert might *inspect* some of the production data to get an estimate. However, by sampling theoretic bounds, the size of the inspected data to get a good estimate of the performance runs in the hundreds. In other words, a domain expert has to label hundreds of data items before obtaining a classifier performance estimate with high accuracy and low variance.

In this paper, we avoid this issue by using a pair of interrelated and *labeled* benchmark datasets. Generically, we represent a dataset of the form $D_1 - D_2$ where both datasets D_1 and D_2 belong to the same domain (such as Android malware or Microsoft PE malware). Naturally, the two datasets are quite different and have different data collection methodologies. Concretely, datasets D_1 and D_2 have different sampling biases and their data distributions are different from each other. Hence blindly using a classifier trained on D_1 to D_2 will provide poor performance. Dataset D_1 corresponds to the training dataset D_T and our goal is to train a classifier that performs well on D_2 that we use for testing. Recall that our approach requires an unlabeled dataset D_U that is similar to D_{test}. Hence, we partition D_2 into two subsets that corresponds to the unlabeled dataset D_U and testing datasets D_{test}. The feature engineering for D_1 and D_2 are identical.

Our experiments are conducted over widely used malware benchmark datasets from Android and Microsoft PE. These datasets are used in a number of recent research papers that apply ML for malware detection. A compendium of these papers can be found in [2, 14, 33, 39, 41, 42]. Each of these datasets has distinct data collection methodologies and thereby sampling biases. Hence, blindly using a classifier trained on D_1 will result in poor performance over D_2.

Android Malware Datasets. Our first group of datasets corresponds to malware detection in the Android domain. As before, this dataset consists of two parts – D_1 and D_2. For D_1, we use a subsample of AndroZoo [3] from the recently published Transcendent paper [8]. It has 232,848 benign and 26,387 malicious apps spanning 5 years from Jan 2014 through to Dec 2018. For dataset D_2, we use five different subsampling strategies (described shortly below) to randomly sample apps of the same size from AndroZoo during the same time period. We construct two variants of this dataset – TRANSCENDENT-ANDROZOO and ANDROZOO-TRANSCENDENT where the training and deployment datasets are swapped.

Microsoft PE Datasets. We use three widely used PE datasets and then appropriately pair them as D_1 and D_2 respectively.

The first dataset is Ember [5] which has 750K benign and 800K malicious samples. The second is the UCSB-Packed [1] dataset with 109K benign and 232K malicious samples. Our final dataset is BODMAS [61] which has 77K benign and 57K malicious samples. We construct two datasets EMBER-UCSB and EMBER-BODMAS by using Ember for D_T and UCSB/BODMAS for D_U and D_{test} respectively. Table 1 shows the details.

Performance Measures. Due to the unbalanced nature of the datasets, we use the F-score to measure the performance.

8.2 Detection of Sampling Bias

Dataset Variants. Each of the four datasets that we introduced in the previous subsection (TN-AZ, AZ-TN, EMB-UCSB, EMB-BODMAS) exhibit malignant sampling bias – wherein a classifier trained on D_T and applied directly on D_{test} produces sub-optimal performance. In order to showcase the versatility and robustness of our algorithms, we create additional variants for each of the four datasets. Concretely, we use *five* strategies to create 10,000 different variants of the datasets (with 2,000 variants for each strategy). Each of these strategies differs in how the base datasets D_1 and D_2 are sampled to form datasets D_T, D_U, and D_{test} respectively. The strategies are: (a) *adversarial* where the data items in D_U and D_{test} are selected to be most different from that in D_T; (b) *benign* where the data items are chosen in a benign and non-adversarial manner so that D_U and D_{test} are similar to each other; (c) *mixed* where the datasets from each of the aforementioned strategies are randomly mixed to produce a more complex partitioning of data; (d) *mixed-2* where the test set D_{test} consists of tuples from both D_1 and D_2 (to ensure that the classifier can perform well on data items in production that could be similar to D_1); (e) *mixed-3* where D_{test} consists of data distribution that is distinct from D_T and D_U. For example, D_{test} can include malware families that are not present in either D_T or D_U. We use the domain discrimination (DOMDISC) from Section 4.1 for *adversarial* and *benign* sampling strategies. Intuitively, we train a classifier to differentiate between D_U and D_{test}. We can see that tuples from D_U that were correctly classified form the source of benign samples. Similarly, tuples that were incorrectly classified or those classified correctly with low confidence can be treated as a source of adversarial samples.

For each of the dataset variants, we train two classifiers – one trained on D_T and another trained on D_U and evaluate their performance on D_{test}. If the difference in performance (e.g. F-score) is at least 5 percentage points, we conclude that malignant sampling bias exists. If not, we assume that the sampling bias is not severely malignant.

Algorithms Evaluated. We propose two algorithms to detect sampling bias– DOMDISC based on domain discrimination and KNN-DIST based on distance distribution of the k-th nearest neighbor. We are not aware of any prior algorithms from the cybersecurity community for the detection of sampling bias. Hence, we select 5 state-of-the-art algorithms from ML and related communities and adapt them to this problem. Concretely, the baseline algorithms are:

(a) Permutation method (PM) [9] is an exact statistical hypothesis testing method that compares two separate data samples. The null hypothesis is that the two samples are from the same underlying data distribution. We use the classifier F-score as the test statistic T. Permutation tests are one of the most powerful nonparametric tests that can distinguish data samples without making any distributional assumptions. Furthermore, a permutation test is known to exist for any test statistic and is simple to design.

(b) Cross Match method (CM) [45] is another powerful hypothesis testing method for verifying if two data samples are from the same underlying distribution. We create a bipartite graph where data items from D_T and D_U form the nodes where the edge between the two nodes measures their corresponding similarity. Next, we run a maximum bipartite matching algorithm that chooses a set of edges such that no two edges share a common node. The matching algorithm maximizes the overall similarity between matched pairs of data items. Then we use Rosenbaum's test [45] to find the distribution of edges where both the nodes are from D_T, D_U, or a combination of D_T and D_U.

(c) f-divergence (f-div) [44] is a function that measures the distance between two probability distributions P and Q. It generalizes other popular metrics such as KL-divergence, Hellinger distance, and total variation distance.

(d) Virtual-logit Matching (ViM) [58] and *(e) MaxLogit method* [24]. Our final two methods are based on state-of-the-art ML methods for detecting out-of-distribution (OOD) data items from tabular data. We apply the OOD method for each data item in D_U and if the proportion of OOD crosses a threshold then we conclude that sampling bias exists.

Each of the five baseline algorithms relies on a threshold to determine whether sampling bias exists. We chose an oracular approach where an omniscient oracle tells us the optimal threshold such as many of the 10K dataset variants are correctly classified. In contrast, we artificially handicap our algorithms by forcing them to use a fixed thresholding strategy. For DOMDISC, we assume that sampling bias exists if the accuracy of the logistic regression classifier is at least 60%. For KNN-DIST, we use the median absolute deviation metric that is known to be a strong and robust estimator.

Comparison against Baseline Algorithms. In our first set of experiments, we compare our two algorithms against the five baseline algorithms. As mentioned above, we created 10K different dataset variants using different subsampling strategies and categorized each of them based on whether it exhibits malignant sampling bias. Each of the algorithms evaluated used the chosen threshold to do the same categorization. Table 2

Dataset	Domain	D_T	#Benign	#Malware	D_U, D_{test}	#Benign	#Malware
TN-AZ	Android	Transcendent	232,848	26,387	AndroZoo	232,848	26,387
AZ-TN	Android	AndroZoo	232,848	26,387	Transcendent	232,848	26,387
EMB-UCSB	Microsoft PE	Ember	750,000	800,000	UCSB	109,030	232,415
EMB-BODMAS	Microsoft PE	Ember	750,000	800,000	BODMAS	77,142	57,293

Table 1: Dataset Characteristics.

shows the results of the F-score of the categorization for each of the algorithms. We can see that our two proposed algorithms achieve the highest F-score. The f-divergence-based method achieves the least F-score as it only measures divergence between the two data distributions without quantifying if the divergence is benign or malignant. Both the hypothesis testing-based methods (permutation tests and cross-match tests) provide good F-score. However, this performance is not representative of the real-world as identifying an appropriate threshold is a challenging problem in hypothesis testing. Finally, the two OOD-based methods ViM and MaxLogit work well. This shows that extending OOD-based methods for detecting sampling bias is a promising line of research.

	TN-AZ	AZ-TN	Emb-UCSB	Emb-BODMAS
DomDisc	0.97	0.98	0.99	0.97
kNN-Dist	0.99	0.98	0.99	0.98
PM	0.89	0.91	0.88	0.9
CM	0.91	0.86	0.9	0.86
f-Div	0.78	0.81	0.77	0.72
ViM	0.94	0.96	0.96	0.95
MaxLogit	0.92	0.93	0.96	0.91

Table 2: Accuracy of Bias Detection Algorithms for 5% threshold.

Impact of Subsampling Strategies. In the next set of experiments, we investigate how the various bias detection algorithms are impacted by the subsampling strategies – adversarial, benign, mixed, mixed-2 and mixed-3 respectively. Due to space limits, Table 3 shows the results for the TN-AZ dataset. The results for the other datasets show the same overall trend. Once again, our proposed algorithms provide robust results regardless of how the datasets were constructed. In fact, our algorithms achieve high F-score even in the case of the adversarial dataset construction strategy. The hypothesis testing-based methods perform poorly in the adversarial setting which is not surprising as they violate the underlying assumption of the two-sample hypothesis testing methods. The OOD-based methods ViM and MaxLogit work well in general and form a promising backup option. In the mixed-2 approach, D_{test} also consists of tuples from the same data distribution as D_T. This is an easier setting as our pseudo-labeling approaches can leverage information from D_T to accurately label these tuples. Hence, this produces results that are better than adversarial and mixed sampling strategies though not as much as that of benign sampling. In the mixed-3 setting, D_{test} (partially) consists of data points that are distinct from both D_T and D_U. This is again a beneficial setting as it is relatively easier to identify a distinct data distribution. Almost all of the baseline algorithms and our proposed algorithms achieve high accuracy in this setting.

	Adv.	Benign	Mxd	Mxd-2	Mxd-3
DomDisc	0.91	0.99	0.92	0.93	0.98
kNN-Dist	0.89	0.99	0.94	0.94	0.99
PM	0.68	0.92	0.77	0.8	0.89
CM	0.81	0.93	0.81	0.86	0.95
f-Div	0.66	0.86	0.77	0.81	0.89
ViM	0.8	0.95	0.82	0.88	0.96
MaxLogit	0.83	0.95	0.87	0.89	0.92

Table 3: Impact of subsampling strategies on the F-score of bias detection algorithms.

Impact of Downstream Classifier. Next, we investigate how the various bias detection algorithms are impacted based on the downstream classifier. Any cybersecurity setup often has a custom ML pipeline and a downstream classifier. For example, Drebin [7] used a linear SVM classifier for malware classification. It is important that all our proposed methods are robust to the downstream classifier. Otherwise, it would be necessary to design different algorithms that are cognizant of the ML setup. Table 4 shows the F-score of the algorithms for five different classifiers – SVM, random forest (RF), logistic regression (LogReg), fully connected DL model (DL), and simple transformer (Trans). We can see that both of our proposed algorithms and the baseline algorithms produce similar results regardless of the classifier. This is the ideal behavior as it allows us to use the same algorithm regardless of the ML pipeline setup.

Impact of Threshold for Bias Detection. By default, we use a threshold of 5% to determine the existence of sampling bias. This is a threshold that is large enough to avoid false positives but small enough to detect meaningful data distribution discrepancies. We conducted additional experiments over other threshold values to show that our method can address both small and large deviations. Table 5 shows the results. As expected, if the threshold is too small (such as 1%), then

	SVM	RF	LogReg	DL	Trans
DomDisc	0.96	0.97	0.97	0.98	0.98
kNN-Dist	0.95	0.95	0.96	0.96	0.96
PM	0.86	0.86	0.84	0.83	0.88
CM	0.89	0.88	0.86	0.83	0.87
f-Div	0.77	0.79	0.8	0.77	0.78
ViM	0.92	0.93	0.92	0.94	0.95
MaxLogit	0.92	0.91	0.91	0.92	0.93

Table 4: Impact of sampling bias detection based on downstream classifier

there is a large amount of false positives resulting in lowered performance. Our models achieve almost perfect performance for higher thresholds. The practitioner can choose a threshold that is appropriate to their situation.

	1%	2%	5%	10%
DomDisc	0.84	0.87	0.97	0.99
kNN-Dist	0.88	0.92	0.96	0.99

Table 5: Impact of threshold parameter on accuracy of bias detection

8.3 Mitigation of Sampling Bias

Algorithms Evaluated. In this paper, we proposed two algorithms – contrastive learning based (ConL-BM) and cycle consistency (CyC-BM) that use different techniques for learning effective representations that are then useful for obtaining pseudo-labels. We also compare these algorithms against 4 state-of-the-art baseline algorithms from unsupervised domain adaptation and self-training, respectively. Recall that the biggest challenge in using prior methods from ML is that they are based on augmentations that are not easy to design for tabular data from cybersecurity. Hence, we focus on SoTA methods that can work well for tabular data. They include:

(a) DANN [19] is a classic algorithm that works by identifying discriminative features that work well on source domains and that are also invariant to the shift between source and target domains. In the modern parlance, the algorithm can be reinterpreted as using a domain discriminator that uses adversarial learning to minimize the domain gap.

(b) SHOT [32] is a SoTA algorithm for performing data-free unsupervised domain adaptation using hypothesis transfer learning. It works by freezing the classifier module (i.e. hypothesis) and learning a target-specific features extraction module through self-supervised pseudo-labeling methods.

(c) VAT [35] is a novel regularization-based learning algorithm based on virtual adversarial loss. It works by exploiting the continuity assumption by making the classifier robust to the conditional label distribution around data items against simple adversarial perturbations. A key advantage is that VAT can be defined without label information and hence can be used to obtain pseudo-labels.

(d) FixMatch [50] uses a two-step process where the model first generates pseudo-labels on weakly augmented unlabeled data items that are then used to train a classifier that works well with strongly augmented data. We use the masking strategy with $p = 0.05$ and $p = 0.15$ from SIRAJ [54] to generate weak and strong augmentations.

Comparison against Baseline Algorithms. Our first set of experiments compares how our two proposed algorithms – ConL-BM and CyC-BM compare against the four state-of-the-art algorithms for minimizing sampling bias mitigation. We use the same setup as the bias detection experiments. For each of the benchmark datasets, we generated 10K dataset variants using different subsampling strategies. For each of the dataset variants, we train three classifiers – classifier $\mathcal{C}_T$ that is trained on D_T, classifier $\mathcal{C}_{TU}$ that is trained on both D_T and D_U (using our proposed and baseline algorithms) classifier $\mathcal{C}_U$ that is trained on D_U. We measure the difference in F-score between ($\mathcal{C}_U$ - $\mathcal{C}_T$) and ($\mathcal{C}_U$ - $\mathcal{C}_{TU}$). We can see that the former is an upper bound of the improvement in F-score that can be obtained by any algorithm.

We would like to note that the performance of each of the classifiers $\mathcal{C}_T$, $\mathcal{C}_{TU}$ and $\mathcal{C}_U$ are sufficiently high for practical deployment. For example, for AZ-TN dataset, $\mathcal{C}_U$ has an F1-score of 0.96 for the best classifier, while $\mathcal{C}_T$ achieves an F1-score of 0.83. Of course, this value is artificially low as TN has a different data distribution than AZ. In most practical scenarios, the difference between $\mathcal{C}_T$ and $\mathcal{C}_U$ will be small (such as 5%). However, our proposed algorithms were able to bridge most of the differences in the performance. Concretely, the classifier $\mathcal{C}_{TU}$ based on ConL-BM achieves an F1-score of 0.933 while CyC-BM achieves an F1-score of 0.937.

Table 6 shows the results of this experiment. We can see that the gap in F-score between a classifier trained only on D_T and only on D_U for TN-AZ dataset is 16.9. Hence, any classifier using unlabeled D_U will necessarily get a lower improvement. Concretely, our contrastive learning-based algorithm was able to achieve 12.3 out of the maximum 16.9 *without* using the labeled data information. We can see that our two proposed algorithms are able to cover most of the difference in the F-score gap for all the 4 benchmark datasets and their $10K$ subsampling variants. None of the unsupervised domain adaptation and self-training-based baselines are even able to match 50% of their F-score gap reduction. This is not surprising as these algorithms were designed for computer vision tasks and require non-trivial adaptation. It is our hope that more work will be done on tabular unsupervised domain adaptation so that we get stronger baselines.

Impact of Subsampling Strategies. In the next set of experiments, we investigate how the various bias mitigation algorithms are impacted by the subsampling strategies – ad-

	TN-AZ	AZ-TN	Emb-UCSB	Emb-BODMAS
max Δ	16.9	12.9	26.1	27.3
ConL-BM	12.3	10.2	19.1	22.3
CyC-BM	14.3	10.6	21.3	22.7
DANN	9.8	8.1	14.7	16.1
SHOT	6.5	6.2	10.1	11.3
VAT	4.4	4.1	8.8	7.9
FixMatch	7.5	6.6	11.3	13.4

Table 6: Comparison of Bias Mitigation Algorithms.

versarial, benign, mixed, mixed-2 and mixed-3 respectively. Due to space limits, Table 7 shows the results for the TN-AZ dataset. The results for the other datasets show the same overall trend. Once again, we can see that our proposed algorithms can successfully mitigate sampling bias regardless of the subsampling strategy used to generate the dataset variant. Interestingly, we achieve good performance in the case of the adversarial strategy even though our algorithms do not use any form of adversarial learning. This is a testament to our representation learning strategy that is class label aware. In the mixed-2 approach, D_{test} also consists of tuples from the same data distribution as D_T. This is an easier setting as our pseudo-labeling approaches can leverage information from D_T to accurately label these tuples. Hence, this produces results that are better than adversarial and mixed sampling strategies though not as much as that of benign sampling. In the mixed-3 sampling strategy, D_{test} partially includes tuples that are very distinct from D_T and D_U. As we can see from the results, almost all of the algorithms perform sub-optimally. This is to be expected as the test distribution diverges sharply from both D_T and D_U. In this very challenging setting, our algorithms outperform the competing algorithms. We can see that the baseline algorithms perform comparatively poorly for all the sub-sampling strategies. This is not surprising as they are often designed for a particular type of domain shift (such as adversarial for DANN). Hence, it is especially gratifying to see that our algorithm works well for different dataset construction strategies. We believe that it is important to design generic algorithms that can automatically work regardless of the type of domain gap.

	Adv.	Benign	Mxd	Mxd-2	Mxd-3
max Δ	24.9	7.9	19.7	11.4	22.7
ConL-BM	18.2	6.7	16.5	12.6	11.2
CyC-BM	19.1	7.1	17.1	9.9	12.4
DANN	11.2	5.4	13.2	8.4	7.2
SHOT	7.6	3.3	7.8	5.4	5.8
VAT	5.1	4.1	7.5	4.8	6.9
FixMatch	8.3	5.1	6.2	5.6	4.4

Table 7: Impact of subsampling strategies on Bias Mitigation

	SVM	RF	LogReg	DL	Trans.
max Δ	9.2	11.3	8.7	16.9	16.2
ConL-BM	8.8	10.2	6.4	12.1	11.9
CyC-BM	9.1	10.3	6.8	14.2	13.8
DANN	6.6	6.8	4.2	9.8	9.8
SHOT	5.1	5.6	3.9	6.6	6.4
VAT	4.4	4.7	3.8	4.3	4.1
FixMatch	6.1	6.3	5.9	7.9	7.7

Table 8: Impact of downstream classifier for Bias Mitigation

Impact of Downstream Classifier on Bias Mitigation. Next, we investigate how the various bias mitigation algorithms are impacted based on the downstream classifier. Recall from Section 8.2 that both our bias detection algorithms and the baselines were not impacted by the classifier. In contrast, the classifier has a significant impact on bias mitigation algorithms. For each of the dataset variants, we varied the classifier used to perform the malware/goodware classification on the TN-AZ dataset. Specifically, we tried five different classifiers – SVM, random forest (RF), logistic regression (LogReg), fully connected DL model (DL), and simple transformer (Trans). Table 8 shows the result of the experiments. We can see that the delta in F-score varies between the classifiers with simpler classifiers such as SVM and Logistic regression achieving a lower improvement compared to deep learning-based methods (DL and Transformers). Ensemble methods based on random forests are in between. This observation can be explained due to the difference in the computational capacity of the classifiers. SVM and Logistic regression are simple classifiers that mostly rely on the linear transformation of features to perform prediction while DL and transformer-based methods can create "deep" features that are more suitable for accurate prediction. Regardless, the gap in performance between the classifiers is not very large in our proposed algorithms. This is due to the fact that our algorithms produce an encoder with an appropriate embedding space where even simple linear classifiers can work accurately. In contrast, other baseline algorithms perform poorly (except for DANN which uses adversarial domain discrimination for feature alignment).

8.4 Generalizability of Bias Mitigation

Our next set of experiments is designed to demonstrate the generalizability of our proposed methods. So far, our experiments have been done on the malware domain. In this subsection, we conduct experiments on intrusion detection and domains. Due to space limits, we focus on bias mitigation experiments as they are more challenging and informative.

IDS Datasets. We demonstrate the efficacy of our approach by evaluating it across three common Intrusion Detection System (IDS) datasets, namely CICIDS2017 [49], ToN-IoT [36], and BoT-IoT [30]. These datasets provide labeled network flows encompassing diverse attack categories. The CICIDS2017 dataset is designed to resemble real-world scenarios, encompassing both benign activities and common attacks such as Brute Force, DoS, DDoS, Heartbleed, Web Attacks, Infiltration, and Botnet attacks. ToN-IoT features labeled network traffic sourced from typical network elements combined with IoT sensors, while BoT-IoT provides IoT network traffic covering a variety of attacks involving Botnets. For each dataset, we extract standard features [47] using the CICFlowMeter-V4 tool [31], which encompasses attributes including basic flow characteristics, statistical properties, temporal patterns, protocol-specific details, and payload-based information. We train a classifier to distinguish attack flows from legitimate traffic, without differentiating between specific attack types.

We construct three dataset pairs to evaluate our algorithms. The dataset IDS2017 is constructed by temporally splitting the CICIDS2017 dataset. We use the data from the first three days for training, the fourth day as unlabeled D_U, and the data from the fifth day for testing. The dataset IDS2017-BoT has the same setting as IDS2017 except that BoT-IoT [30] is used for testing. Finally, the dataset IDS2017-ToN has the same setting as IDS2017 except that ToN-IoT [36] is used for testing. We can see that IDS2017-BoT and IDS2017-ToN are challenging settings as we use IDS2017 for D_U even though they are sufficiently from the dataset used for testing.

	IDS2017	**IDS2017-BoT**	**IDS2017-ToN**
max Δ	4.2	12.4	14.6
ConL-BM	3.5	9.1	11.4
CyC-BM	3.8	10.2	12.3

Table 9: Performance of bias mitigation algorithms for intrusion detection

Table 9 shows the results. As expected, our methods perform well for all three dataset pairs. Since the domain discrepancy is the least for IDS2017, the maximum possible improvement in classifier performance is just 4.2 percentage points. Nevertheless, our methods are able to recover as much as 3.8 out of 4.2 that was lost due to domain discrepancy. The improvements are much larger for IDS2017-BoT and IDS2017-ToN as the distribution discrepancy between D_U and D_{test} is much higher.

Domain Datasets. We further analyze three distinct domain datasets, comprising two benign and one malicious, to assess the effectiveness of our approach. The first dataset consists of domains sourced from the Tranco top 100K list [40], representing widely used domains on the Internet. The second dataset is derived from the manually profiled organizations list from Crunchbase [16], with 100K domains randomly selected from a pool of 3 million excluding overlaps with the Tranco 1M list. Our third dataset, which is malicious, was collected from the daily VirusTotal data feed [57], comprising domains flagged as malicious by at least six scanners, observed on February 6, 2024. As for the features, we utilize passive DNS attributes from Farsight PDNS data [18], which shed light on the hosting environment, along with features extracted from registration records such as lifetime, number of name servers, and indicators of privacy protection. To train a classifier, we combine each benign dataset with the malicious one and train a binary classifier. Subsequently, we test the trained classifier on the other benign dataset to assess its performance.

	Tranco-CB	**CB-Tranco**
max Δ	5.3	5.1
ConL-BM	4.7	4.6
CyC-BM	5.1	4.8

Table 10: Performance of bias mitigation algorithms for domains

Table 10 shows the results. We can see that the data distribution discrepancy between the two pairs of datasets is comparable. In both cases, our proposed algorithms are able to recover most of the performance lost to the discrepancy. Once again, the iterative algorithm CYC-BM outperforms that simpler CONL-BM.

9 Discussion

In this section, we provide additional details about the various facets of our proposed approach.

Domain Adaptation vs Sampling Bias Detection/Mitigation. Our work is closely related to the unsupervised domain adaptation idea. However, instead of directly attacking this problem, we split this into two parts – bias detection and mitigation. This problem partitioning allows the practitioner to pick and choose the solutions based on their individualized setting. For example, a practitioner can use a bias detection component and if the bias is deemed small enough, use their existing set classifier without running the bias mitigation algorithms that are effective but expensive. The practitioner is also free to choose thresholds other than 5%. Alternatively, a conservative practitioner might always choose to run the bias mitigation algorithm that is guaranteed to help in the worst case (when D_T and D_{test} are different) and does not cause any performance regression in the best case (when D_T and D_{test} are similar). We believe that our proposed approach allows for many more degrees of freedom than directly solving it as a domain adaptation problem.

Accuracy-Performance Trade-offs. Each of our four algorithms is designed to achieve different trade-offs. For example, DOMDISC is designed to be very efficient. Even on a dataset with tens of millions of tuples, DOMDISC produces an output within a minute. However, KNN-DIST is less efficient but much more tunable and can produce more accurate results than DOMDISC. As mentioned in Section 8.1, we artificially handicapped all of our algorithms to use a fixed threshold (such as 5% for DomDisc and median absolute deviation for kNN-Dist). However, with some careful tuning, kNN-Dist will always outperform DomDisc at the cost of increased runtime. ConL-BM is a comparatively simpler algorithm than Cyc-BM where it learns a latent space using carefully chosen positive and negative tuple pairs. In contrast, Cyc-BM is an iterative algorithm where the latent space is learned in each of the forward and reverse steps. This iterative nature results in a much more sophisticated latent space as it focuses on a tougher task (getting good accuracy using pseudo labels) as against ConL-BM which only seeks to embed similar tuples closer together. In both cases, the slower algorithm takes 3-8x more time than the faster one. We have described all these algorithms so that the practitioner can choose the algorithm based on their specific setup.

Threshold Selection. The threshold parameter for bias detection can be considered as a proxy for the tolerance level of the cybersecurity application for data distribution discrepancies. Hence a low threshold (such as 1%) can be chosen when the classifier is used in a highly performance sensitive scenario where even a drop of 1% in accuracy is unacceptable. However, the trade-off is that it can inadvertently detect some false positives. A higher threshold (such as 5% or 10%) can be used in exploratory setting (such as evaluating diverse D_U or different algorithms for bias mitigation).

Limitations of Our Approach. Our proposed algorithms (or any ML classifier for that matter) will produce sub-optimal performance when D_{test} is very distinct from both D_T and D_U. Another common source of failure is the low accuracy of pseudo labels. Due to the various tricks that we employ (such as cyclic training and encoders to learn shared latent space), the pseudo labels typically have good accuracy. The performance of classifier C_U can be used as a proxy for the pseudo label accuracy. If the accuracy is low, then the domain expert has to be called upon to label a few tuples from D_U.

For both the bias mitigation algorithms, the accuracy of the pseudo labels is quite low in the first few epochs as the encoder has not learned to map both D_T and D_U into a shared latent space. However, after a few epochs, the accuracy of the pseudo labels starts increasing. In our experiments, we found that the pseudo labels have an accuracy between 70-80% on average and as much as 95% for the benign subset sampling strategy. The pseudo-labeling community has come up with a wide variety of confidence-based heuristics so that only the pseudo-labels that have a high likelihood of being correct are added to the pool of D_U^P. In other words, even though the classifiers trained on pseudo labels have only 70-80% accuracy, the subset of tuples D_P^U that are used to train C_U have an accuracy above 90%. Of course, it is possible that the pseudo-labeling strategy does not work for certain datasets. In that case, the domain expert has to be called upon to label a few tuples from D_U and repeat the whole process.

10 Related Work

As discussed in [11, 43], sampling bias is the most common pitfall in cybersecurity projects that use ML. Approximately 90% of cybersecurity research exhibits some degree of sampling bias. Despite its prevalence, this issue has not received extensive attention. Tesseract [38] considers spatial bias, which arises from variations in the ratio of goodware to malware, and temporal bias, caused by incorrect time splits in the training and testing sets. As a special case of sampling bias, where training and testing data distribution change over time, several studies address concept drift. Transcend [26] employs conformal evaluators that rely on the similarity of a new sample to a history of past samples. They define a non-conformity measure as a fitness assessment and use a credibility p-value to quantify how similar the testing sample is to the training samples that belong to the same class. Trancendent [8] introduces a general form of Transcend and proposes methods to define class-level thresholds. CADE [62] employs contrastive learning to detect drifting samples. Similarly, [13] proposes hierarchical contrastive learning for detection and uses pseudo-loss uncertainty to decide which samples to include in the active learning process.

DroidEvolver [60] utilizes a model pool, where each model maintains a subset of the training samples. A model is designated as aging if the new sample differs from its training samples, and the aging models are updated using active learning with the pseudo-labels generated by the non-aging models in the pool. DroidEvolver++ [27] observes the self-poisoning effect of pseudo-labels and uses only high-quality pseudo-labels through a confidence thresholding mechanism. Insomnia [6] also leverages pseudo-labels for updating its classifier through active learning. It selects samples using uncertainty sampling and generates pseudo-labels by identifying the nearest centroid. OWAD [23] employs a statistical approach and applies hypothesis testing to detect whether the model outputs exhibit a normality shift. It uses uncertainty sampling to select new samples for labeling.

11 Conclusion

In this paper, we initiated investigations into an understudied problem of sampling bias in cybersecurity. We focused on two key sub-problems – detection and mitigation. We proposed two different algorithms for the detection of sampling bias. We proposed two different algorithms for bias mitigation that learns effective embedding space wherein even simple pseudo-labeling strategies produce good results. Our extensive experiments over four datasets from three different domains shows that our proposed methods are effective and generalizable across cybersecurity.

Acknowledgments

We thank the anonymous reviewers for their helpful and valuable feedback. This work was partially supported by the Qatar National Research Fund under Grant ARG01-0531-230438.

References

[1] Hojjat Aghakhani, Fabio Gritti, Francesco Mecca, Martina Lindorfer, Stefano Ortolani, Davide Balzarotti, Giovanni Vigna, and Christopher Kruegel. When malware is packin'heat; limits of machine learning classifiers based on static analysis features. In *Network and Distributed Systems Security (NDSS) Symposium 2020*, 2020.

[2] Mostofa Ahsan, Kendall E Nygard, Rahul Gomes, Md Minhaz Chowdhury, Nafiz Rifat, and Jayden F Connolly. Cybersecurity threats and their mitigation approaches using machine learning—a review. *Journal of Cybersecurity and Privacy*, 2(3):527–555, 2022.

[3] Kevin Allix, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. Androzoo: Collecting millions of android apps for the research community. In *Proceedings of the 13th international conference on mining software repositories*, pages 468–471, 2016.

[4] Massih-Reza Amini, Vasilii Feofanov, Loic Pauletto, Emilie Devijver, and Yury Maximov. Self-training: A survey. *arXiv preprint arXiv:2202.12040*, 2022.

[5] Hyrum S Anderson and Phil Roth. Ember: an open dataset for training static pe malware machine learning models. *arXiv preprint arXiv:1804.04637*, 2018.

[6] Giuseppina Andresini, Feargus Pendlebury, Fabio Pierazzi, Corrado Loglisci, Annalisa Appice, and Lorenzo Cavallaro. Insomnia: Towards concept-drift robustness in network intrusion detection. In *Proceedings of the 14th ACM workshop on artificial intelligence and security*, pages 111–122, 2021.

[7] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. Drebin: Effective and explainable detection of android malware in your pocket. In *Ndss*, volume 14, pages 23–26, 2014.

[8] Federico Barbero, Feargus Pendlebury, Fabio Pierazzi, and Lorenzo Cavallaro. Transcending transcend: Revisiting malware classification in the presence of concept drift. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 805–823. IEEE, 2022.

[9] Kenneth J Berry, Janis E Johnston, and Paul W Mielke Jr. Permutation methods. *Wiley Interdisciplinary Reviews: Computational Statistics*, 3(6):527–542, 2011.

[10] Marcus Botacin, Hojjat Aghakhani, Stefano Ortolani, Christopher Kruegel, Giovanni Vigna, Daniela Oliveira, Paulo Lício De Geus, and André Grégio. One size does not fit all: A longitudinal analysis of brazilian financial malware. *ACM Transactions on Privacy and Security (TOPS)*, 24(2):1–31, 2021.

[11] Marcus Botacin, Fabricio Ceschin, Ruimin Sun, Daniela Oliveira, and André Grégio. Challenges and pitfalls in malware research. *Computers & Security*, 106:102287, 2021.

[12] Ke Chen and Shihai Wang. Semi-supervised learning via regularized boosting working on multiple semi-supervised assumptions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):129–143, 2010.

[13] Yizheng Chen, Zhoujie Ding, and David Wagner. Continuous learning for android malware detection. In *Proceedings of the 32nd USENIX Security Symposium*, 2023.

[14] Yizheng Chen, Zhoujie Ding, and David Wagner. Continuous learning for android malware detection. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 1127–1144, 2023.

[15] Corinna Cortes, Mehryar Mohri, Michael Riley, and Afshin Rostamizadeh. Sample selection bias correction theory. In *International conference on algorithmic learning theory*, pages 38–53. Springer, 2008.

[16] Crunchbase. Crunchbase. `http://www.crunchbase.com/`. Accessed: 08-02-2024.

[17] Mohammed Elbes, Samar Hendawi, Shadi AlZu'bi, Tarek Kanan, and Ala Mughaid. Unleashing the full potential of artificial intelligence and machine learning in cybersecurity vulnerability management. In *2023 International Conference on Information Technology (ICIT)*, pages 276–283. IEEE, 2023.

[18] Farsight Security, Inc. DNS Database. https://www.dnsdb.info/, 2022.

[19] Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. In *International conference on machine learning*, pages 1180–1189. PMLR, 2015.

[20] R Geetha and T Thilagam. A review on the effectiveness of machine learning and deep learning algorithms for cyber security. *Archives of Computational Methods in Engineering*, 28:2861–2879, 2021.

[21] Robert M Groves and Lars Lyberg. Total survey error: Past, present, and future. *Public opinion quarterly*, 74(5):849–879, 2010.

[22] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE computer society conference on computer vision and pattern recognition (CVPR'06)*, volume 2, pages 1735–1742. IEEE, 2006.

[23] Dongqi Han, Zhiliang Wang, Wenqi Chen, Kai Wang, Rui Yu, Su Wang, Han Zhang, Zhihua Wang, Minghui Jin, Jiahai Yang, et al. Anomaly detection in the open world: Normality shift detection, explanation, and adaptation. In *30th Annual Network and Distributed System Security Symposium (NDSS)*, 2023.

[24] Dan Hendrycks, Steven Basart, Mantas Mazeika, Andy Zou, Joe Kwon, Mohammadreza Mostajabi, Jacob Steinhardt, and Dawn Song. Scaling out-of-distribution detection for real-world settings. *arXiv preprint arXiv:1911.11132*, 2019.

[25] Anil K Jain, M Narasimha Murty, and Patrick J Flynn. Data clustering: a review. *ACM computing surveys (CSUR)*, 31(3):264–323, 1999.

[26] Roberto Jordaney, Kumar Sharad, Santanu K Dash, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro. Transcend: Detecting concept drift in malware classification models. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 625–642, 2017.

[27] Zeliang Kan, Feargus Pendlebury, Fabio Pierazzi, and Lorenzo Cavallaro. Investigating labelless drift adaptation for malware detection. In *Proceedings of the 14th ACM Workshop on Artificial Intelligence and Security*, pages 123–134, 2021.

[28] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. *Advances in neural information processing systems*, 33:18661–18673, 2020.

[29] Ilmun Kim, Aaditya Ramdas, Aarti Singh, and Larry Wasserman. Classification accuracy as a proxy for two-sample testing. *Annals of Statistics*, 49(1):411–434, 2021.

[30] Nickolaos Koroniotis, Nour Moustafa, Elena Sitnikova, and Benjamin Turnbull. Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset. *Future Generation Computer Systems*, 100:779–796, 2019.

[31] Arash Habibi Lashkari, Gerard Draper Gil, Mohammad Saiful Islam Mamun, and Ali A Ghorbani. Characterization of tor traffic using time based features. In *International Conference on Information Systems Security and Privacy*, volume 2, pages 253–262. SciTePress, 2017.

[32] Jian Liang, Dapeng Hu, and Jiashi Feng. Do we really need to access the source data? source hypothesis transfer for unsupervised domain adaptation. In *International conference on machine learning*, pages 6028–6039. PMLR, 2020.

[33] Kaijun Liu, Shengwei Xu, Guoai Xu, Miao Zhang, Dawei Sun, and Haifeng Liu. A review of android malware detection approaches based on machine learning. *IEEE access*, 8:124579–124607, 2020.

[34] Brad Miller, Alex Kantchelian, Michael Carl Tschantz, Sadia Afroz, Rekha Bachwani, Riyaz Faizullabhoy, Ling Huang, Vaishaal Shankar, Tony Wu, George Yiu, et al. Reviewer integration and performance measurement for malware detection. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 13th International Conference, DIMVA 2016, San Sebastián, Spain, July 7-8, 2016, Proceedings 13*, pages 122–141. Springer, 2016.

[35] Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, and Shin Ishii. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *IEEE transactions on pattern analysis and machine intelligence*, 41(8):1979–1993, 2018.

[36] Nour Moustafa. A new distributed architecture for evaluating ai-based security systems at the edge: Network ton_iot datasets. *Sustainable Cities and Society*, 72:102994, 2021.

[37] Fionn Murtagh and Pedro Contreras. Algorithms for hierarchical clustering: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(1):86–97, 2012.

[38] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, Lorenzo Cavallaro, et al. Tesseract: Eliminating experimental bias in malware classification across space and time. In *Proceedings of the 28th USENIX Security Symposium*, pages 729–746. USENIX Association, 2019.

[39] Fabio Pierazzi, Feargus Pendlebury, Jacopo Cortellazzi, and Lorenzo Cavallaro. Intriguing properties of adversarial ml attacks in the problem space. In *2020 IEEE symposium on security and privacy (SP)*, pages 1332–1349. IEEE, 2020.

[40] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński, and Wouter Joosen. Tranco: A research-oriented top sites ranking hardened against manipulation. *arXiv preprint arXiv:1806.01156*, 2018.

[41] Morteza Safaei Pour, Christelle Nader, Kurt Friday, and Elias Bou-Harb. A comprehensive survey of recent internet measurement techniques for cyber security. *Computers & Security*, 128:103123, 2023.

[42] Junyang Qiu, Jun Zhang, Wei Luo, Lei Pan, Surya Nepal, and Yang Xiang. A survey of android malware detection with deep neural models. *ACM Computing Surveys (CSUR)*, 53(6):1–36, 2020.

[43] E Quiring, F Pendlebury, A Warnecke, F Pierazzi, C Wressnegger, L Cavallaro, and K Rieck. Dos and don'ts of machine learning in computer security. In *31st USENIX Security Symposium (USENIX Security 22), USENIX Association, Boston, MA*, 2022.

[44] Alfréd Rényi. On measures of entropy and information. In *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics*, volume 4, pages 547–562. University of California Press, 1961.

[45] Paul R Rosenbaum. An exact distribution-free test comparing two multivariate distributions based on adjacency. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 67(4):515–530, 2005.

[46] Sinan Saraçli, Nurhan Doğan, and İsmet Doğan. Comparison of hierarchical cluster analysis methods by cophenetic correlation. *Journal of inequalities and Applications*, 2013(1):1–8, 2013.

[47] Mohanad Sarhan, Siamak Layeghy, and Marius Portmann. Evaluating standard feature sets towards increased generalisability and explainability of ml-based network intrusion detection. *Big Data Research*, 30:100359, 2022.

[48] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[49] Iman Sharafaldin, Arash Habibi Lashkari, Ali A Ghorbani, et al. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp*, 1:108–116, 2018.

[50] Kihyuk Sohn, David Berthelot, Nicholas Carlini, Zizhao Zhang, Han Zhang, Colin A Raffel, Ekin Dogus Cubuk, Alexey Kurakin, and Chun-Liang Li. Fixmatch: Simplifying semi-supervised learning with consistency and confidence. *Advances in neural information processing systems*, 33:596–608, 2020.

[51] Valmi D Sousa, Jaclene A Zauszniewski, and Carol M Musil. How to determine whether a convenience sample represents the population. *Applied Nursing Research*, 17(2):130–133, 2004.

[52] Yiyou Sun, Yifei Ming, Xiaojin Zhu, and Yixuan Li. Out-of-distribution detection with deep nearest neighbors. In *International Conference on Machine Learning*, pages 20827–20840. PMLR, 2022.

[53] Zhichuang Sun, Ruimin Sun, Long Lu, and Alan Mislove. Mind your weight (s): A large-scale study on insufficient machine learning model protection in mobile apps. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1955–1972, 2021.

[54] Saravanan Thirumuruganathan, Mohamed Nabeel, Euijin Choo, Issa Khalil, and Ting Yu. Siraj: a unified framework for aggregation of malicious entity detectors. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 507–521. IEEE, 2022.

[55] Robert Tibshirani, Guenther Walther, and Trevor Hastie. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(2):411–423, 2001.

[56] Rosanna Turrisi, Rémi Flamary, Alain Rakotomamonjy, and Massimiliano Pontil. Multi-source domain adaptation via weighted joint distributions optimal transport. In *Uncertainty in Artificial Intelligence*, pages 1970–1980. PMLR, 2022.

[57] VirusTotal. Free Online Virus, Malware and URL Scanner. https://www.virustotal.com/. Accessed: 04-11-2023.

[58] Haoqi Wang, Zhizhong Li, Litong Feng, and Wayne Zhang. Vim: Out-of-distribution with virtual-logit matching. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4921–4930, 2022.

[59] Qizhe Xie, Minh-Thang Luong, Eduard Hovy, and Quoc V Le. Self-training with noisy student improves imagenet classification. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10687–10698, 2020.

[60] Ke Xu, Yingjiu Li, Robert Deng, Kai Chen, and Jiayun Xu. Droidevolver: Self-evolving android malware detection system. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 47–62. IEEE, 2019.

[61] Limin Yang, Arridhana Ciptadi, Ihar Laziuk, Ali Ahmadzadeh, and Gang Wang. Bodmas: An open dataset for learning based temporal analysis of pe malware. In *4th Deep Learning and Security Workshop*, 2021.

[62] Limin Yang, Wenbo Guo, Qingying Hao, Arridhana Ciptadi, Ali Ahmadzadeh, Xinyu Xing, and Gang Wang. Cade: Detecting and explaining concept drift samples

for security applications. In *USENIX security symposium*, pages 2327–2344, 2021.

[63] Bianca Zadrozny. Learning and evaluating classifiers under sample selection bias. In *Proceedings of the twenty-first international conference on Machine learning*, page 114, 2004.

[64] Barret Zoph, Golnaz Ghiasi, Tsung-Yi Lin, Yin Cui, Hanxiao Liu, Ekin Dogus Cubuk, and Quoc Le. Rethinking pre-training and self-training. *Advances in neural information processing systems*, 33:3833–3845, 2020.

[65] Yang Zou, Zhiding Yu, Xiaofeng Liu, BVK Kumar, and Jinsong Wang. Confidence regularized self-training. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 5982–5991, 2019.