A Tiny, Client-Side Classifier

C. Meyers, A. P. MacSween, T. Löfstedt, and E. Elmroth

May 17, 2025

Abstract

The recent developments in machine learning have highlighted a conflict between online platforms and their users in terms of privacy. The importance of user privacy and the struggle for power over user data has been intensified as regulators and operators attempt to police the online platforms. As users have become increasingly aware of privacy issues, client-side data storage, management, and analysis have become a favoured approach to large-scale centralised machine learning. However, state-of-the-art machine learning methods require vast amounts of labelled user data, making them unsuitable for models that reside client-side and only have access to a single user's data. State-of-the-art methods are also computationally expensive, which degrades the user experience on compute-limited hardware and also reduces battery life. A recent alternative approach has proven remarkably successful in classification tasks across a wide variety of data—using a compression-based distance measure (called normalized compression distance) to measure the distance between generic objects in classical distance-based machine learning methods. In this work, we demonstrate that the normalized compression distance is actually not a metric; develop it for the wider context of kernel methods to allow modelling of complex data; and present techniques to improve the training time of models that use this distance measure. We show that the normalised compression distance works as well as and sometimes better than other metrics and kernels—without incurring additional computational costs and in spite of the lack of formal metric properties. The end results is a simple model with remarkable accuracy even when trained on a very small number of samples allowing for models that are small and effective enough to run entirely on a client device using only user-supplied data.

1 Introduction

Modern machine learning (ML) methods have demonstrated remarkable efficacy across many domains. However, they often have large numbers of parameters, and thus also require large numbers of samples to train on [1]. This aggregation of vast amounts of data creates numerous privacy, safety, and security threats [2] that are consequences of large-scale user data collection, for instance by online platform operators (see Section 1.1 for more details). We evaluate and extend prior work on compression-based distance measures by incorporating them into novel kernel methods, enabling efficient classification even with limited training samples. When using small training sets and small and efficient models, they can be trained entirely on a client device without sharing private user data with anyone—allowing the model builder to circumvent many weaknesses associated with state-of-the-art methods [2, 3, 1]. We demonstrate the efficacy of the proposed approach in the context of malware detection, network intrusion detection, and spam detection.

1.1 Threat Model

In the context of online platforms, data are collected from end-users using dubious amounts of consent [4] and aggregated at massive scales [1]. Such data collection on online platforms often creates privacy, safety, and security risks [5, 6]. One such privacy risk is the periodic attempts by regulators to weaken encryption standards [7] and create backdoors to user devices. Platform operators and governments discuss best-practices for scanning client devices for illegal or "offensive" content [2, 8], but existing proposals are no less risky to user privacy, safety, or security than weakening encryption. Privacy experts have denounced such approaches for numerous reasons: the ease of *extracting* private training data [9, 10, 11] or the model itself [10, 12, 13, 14, 15], the ability of a malicious user to induce false positives for other users when the model is trained on user data [*poisoning* attacks; 16, 17, 18, 19, 20, 21, 22, 23, 24], and the triviality of simply *evading* the detection mechanism [25, 26, 27, 28, 6, 5, 29, 27].

Current platform solutions often rely on large-scale ML methods that are trained on vast amounts of user data and federated across devices [8]—an approach that has been criticized by privacy experts [2]. Examples of security risks include attacks against ML systems that target a model during train-

ing [30], prediction [28, 29, 25], and deployment [31, 32]. Even when access to a model by an adversary is limited, it is possible to induce a misclassification [27], reverse engineer the model [33], determine the model weights [13], or infer the class-membership of new samples [34]. This raises profound questions for safety-critical systems [6] and legal questions about access and control of the underlying data [35, 36]. Additionally, it has been shown that finding prototypical meta samples from the training set of large-scale ML models is trivial [37, 6]. Even if the attacker only has access to a typical application programming interface (API), there are reliable ways to fool the model [27].

Many privacy experts warn of the potential for any hypothetical *centralized* content-censoring system not only because of the potential failures due to adversaries mentioned above, but also because of the potential these systems to be used for mass surveillance and censorship [2]. In short, distributed and centralised training paradigms are both inherently fragile to malicious users and dangerous for user privacy, even if the resulting model lives on the user device (rather than in the cloud).

1.2 Motivations

In contrast to many state-of-the-art methods, this work proposes a lightweight, client-side approach to content filtering that does not rely on large-scale data collection.

Recently, Jiang *et al.* [38] proposed a remarkably successful approach to "parameter free" classification, dubbed NCD-KNN, that exploits a compression-based distance measure, the *normalized compression distance* [NCD; 39], to classify objects using the k-nearest neighbours (KNN) method [40]. NCD-KNN is known to work well even when trained on small numbers of samples [41]. By building a model that is accurate on a small number of samples, we can fulfil the goal of training a ML model entirely on a client device using data generated by a single user. An additional goal of this work was to evaluate the efficacy of NCD in general and to extend it to kernel methods in particular.

While other research examined topics like image classification [42], molecular property classification [43], and text classification [44], the ability of NCD to classify datasets that contain strings, numeric values, and categorical data (heterogenous datasets) has remained unexplored.

Additionally, the original NCD work [39] included an error term that

is usually ignored in recent research [42, 43, 44, 38]. To the best of our knowledge, no research has addressed the problem of negative values for NCD. The original authors [39] asserted that NCD is always positive when using perfect compressors. However, this is clearly demonstrated to be false in Lemma 1 when using imperfect compressors (which is what we will have in practice).

Prior works about NCD have primarily focused on distance-based ML methods (*e.g.*, KNN), which limits both the class of methods that can be considered and the types of data that can be effectively analysed. A key motivation for this work was to extend the use of NCD beyond distance-based methods by developing a novel kernel-based formulation. This significantly broadens the applicability of NCD, making it suitable for a wider range of machine learning techniques where traditional distance-based approaches cannot be used.

1.3 Contributions

To use NCD, the model builder must choose a compression algorithm. While the effect of various compressors has been explored, in part, before [45], we expand the analysis by Cebrián *et al.* [45] to more recent compression algorithms and also offer additional run-time improvements over previous implementations [38].

The NCD-KNN method has shown very strong performance across several benchmarks, but prior implementations [38] are not appropriate for real-time settings due to unnecessary repeated computations. We therefore propose several run-time improvements and modifications, outlined in Section 3.1.

Further, we show that NCD is not a metric [42, 43, 44, 38], which means that applying ML methods blindly can lead to erroneous results (*e.g.*, by incorrectly ordering the nearest neighbours). In this work we demonstrate this non-metric behaviour in Lemma 1 and propose techniques to mitigate the effects of this behaviour in Section 3.1, effectively making the modified NCD "more like a metric".

Additionally, we expand the notion of NCD [42, 43, 44, 39, 38, 38] to kernels (Section 3.2) thus allowing for this method to be used with other models besides KNN. We thus extend NCD to reproducing kernel Hilbert spaces and hence more elaborate ML methods—allowing its use in a broader set of ML methods and to model more complex decision boundaries.

Hence, in this paper, we:

- Demonstrate that the normalized compression distance is not a metric (Lemma 1) and propose techniques to make it behave "more like a metric" (Section 3.1.2).
- Propose the use of NCD as a kernel and evaluate its efficacy.
- Develop classifiers (evaluations with KNN, logistic regression, and support vector machines (SVC)) that offers large run-time improvements over a reference implementation [38] (Section 3.1).
- Evaluate and empirically show the efficacy of the proposed NCD-based classifiers across multiple binary classification tasks (Section 4).

2 Background

In the sections below, we describe and define the NCD, outline the NCD-KNN method proposed by Jiang *et al.* [38], outline several other string metrics, and discuss the distance matrix and how to efficiently compute it,

2.1 Normalized Compression Distance

NCD has been demonstrated to be a *universal* measure of similarity between two objects [39]—where a value of 0 denotes equivalence and a value of 1 denotes complete dissimilarity. The NCD is defined as [39]

$$\mathrm{NCD}(x, x') = \frac{|\mathcal{C}(xx')| - \min\{|\mathcal{C}(x)|, |\mathcal{C}(x')|\}}{\max\{|\mathcal{C}(x)|, |\mathcal{C}(x')|\}} + \varepsilon, \tag{1}$$

where $|\mathcal{C}(z)|$ is the length of the compressed form of the data, z, using compression algorithm, $\mathcal{C}$, the notation xx' denotes the concatenation of strings x and x', and $\varepsilon \geq 0$ is an error term accounting for imperfect compression algorithms [39]. The error term is usually assumed to be small relative to the other term.

NCD requires a choice of compression algorithm, and we evaluated the `gzip` [46], `bz2` [47], and `brotli` [48] compressors. To distinguish between the use of these different compressors, a subscript is used such that $\mathrm{NCD}_{\mathrm{gzip}}$ denotes the use of the `gzip` compressor.

While NCD is often discussed as a measure of distance [42, 43, 44, 38, 39], it does in fact not adhere to the axioms of metric spaces. A function, d :

$X \times X \to \mathbb{R}$, associated with a set of points, X, where $\mathbb{R}$ denotes the set of real numbers, is said to be a metric if the following four axioms hold for all $x_1, x_2, x_3 \in X$ [49]:

$$\text{Zero Axiom:} \quad d(x_1, x_2) = 0 \iff x_1 = x_2 \tag{2}$$

$$\text{Non-negativity Axiom:} \quad d(x_1, x_2) \geq 0 \tag{3}$$

$$\text{Symmetry Axiom:} \quad d(x_1, x_2) = d(x_2, x_1) \tag{4}$$

$$\text{Triangle Inequality:} \quad d(x_1, x_3) \leq d(x_1, x_2) + d(x_2, x_3). \tag{5}$$

Much of the literature devoted to NCD has treated it as a proper metric [42, 43, 44, 38]. However, as we show now, it is not.

Lemma 1. *When using* `gzip`*,* `bz2`*, and* `brotli` *compressors, NCD does not adhere to the axioms in Equations 2–5, and is thus not a metric.*

Proof. We show in what follows that NCD fails to adhere to the axioms for metrics by counter-examples.

Zero axiom: The following counter-examples violate the zero axiom:

$$\text{NCD}_{\text{gzip}}(A, A) = 0.05, \quad \text{NCD}_{\text{bz2}}(B, G) = 0, \quad \text{and} \quad \text{NCD}_{\text{brotli}}(X, X) = 0.2.$$

Non-negativity axiom: The following counter-examples violate the non-negativity axiom:

$$\text{NCD}_{\text{gzip}}(AAAA, AAAA) = -0.04,$$

$$\text{NCD}_{\text{bz2}}(AABABAA, BAABAAB) = -0.03,$$

and

$$\text{NCD}_{\text{brotli}}(CCCCBBCCC, CBCCCBBCCC) = -0.08.$$

Symmetry axiom: The following counter-examples violate the symmetry axiom:

$$\text{NCD}_{\text{gzip}}(AA, BAA) = 0.13 \neq \text{NCD}_{\text{gzip}}(BAA, AA) = 0.04,$$

$$\text{NCD}_{\text{bz2}}(AA, AAB) = 0.11 \neq \text{NCD}_{\text{bz2}}(AAB, AA) = 0.00,$$

and

$$\text{NCD}_{\text{brotli}}(AAAAAAA, B) = 0.6 \neq \text{NCD}_{\text{brotli}}(B, AAAAAAA) = 0.7.$$

Triangle Inequality: The following counter-examples violate the triangle inequality:

$$\mathrm{NCD}_{\mathrm{gzip}}(AAA, A) > \mathrm{NCD}_{\mathrm{gzip}}(AAA, AAAA) + \mathrm{NCD}_{\mathrm{gzip}}(AAAA, A),$$

$$\mathrm{NCD}_{\mathrm{bz2}}(BC, AN) > \mathrm{NCD}_{\mathrm{bz2}}(BC, J) + \mathrm{NCD}_{\mathrm{bz2}}(J, AN),$$

and if

$$x_1 = CAAAACAA, \quad x_2 = CAC, \quad \text{and} \quad x_3 = CCACCCACCC,$$

then

$$\mathrm{NCD}_{\mathrm{bz2}}(x_1, x_3) > \mathrm{NCD}_{\mathrm{bz2}}(x_1, x_2) + \mathrm{NCD}_{\mathrm{bz2}}(x_2, x_3).$$

□

2.2 Other String Metrics

To model datasets that comprise strings, several existing measures of distance between strings are routinely used [50]. To evaluate the relative performance of the NCD metric, we compared it to several other common measures of string distance:

- *Levenshtein* is the "edit distance" or minimum number of single-character edits to transform one string into another [51].
- *Lev Ratio* is the Levenshtein distance divided by the total length of the longer string [50].
- *Hamming* is the number of character positions where two strings differ.
- *Ham Ratio* is the number of character positions where two strings differ, divided by the length of the longer string.

2.3 Calculating the distance matrix

In what follows, the pairwise distances between two sets of samples (denoted X and X') is collected in a a *distance matrix*, D. When computing the value of $\mathrm{NCD}(x, x')$, it is necessary to calculate the values of $\mathcal{C}(x)$ and $\mathcal{C}(x')$. To classify a sample, Jiang *et al.* [38] iterated over all elements in the sets X and X', where X would be a set of samples with known labels and X' one

with unknown labels, such that for each $x_i \in X$ the compression $\mathcal{C}(x_i)$ was computed repeatedly for each $x'_j \in X'$. Because computational time scales linearly with the size of both X and X', the run time is thus $\mathcal{O}(|X| \cdot |X'|)$ for both compute and memory, where $|X|$ is the cardinality of the set X. Clearly, if the number of samples in X and X' are large, then run-time becomes a concern. We propose some modifications to how the pairwise distances are computed and which pairwise distances are computed to reduce the computational costs and also make NCD behave "more like a metric", which is outlined in Section 3.1.

3 Methods

This section outlines proposed modifications to NCD and Jiang's NCD-KNN method [38] before outlining the data and experiments used to verify the efficacy of said modifications.

3.1 Proposed Modifications to NCD

Jiang *et al.*'s implementation of the NCD-KNN method [38] becomes inefficient because it includes many repeated computations. To minimize run-time, we first propose a simple improvement in Section 3.1.1. Second, we propose several modifications that symmetrise the distance matrix, intended to ensure adherence to the symmetry axiom (Equation 4). Then, in an effort to improve the adherence to the zero axiom (Equation 2) we propose a modification for the special case of $x = x'$ in Section 3.1.3. Finally, we outline the proposed way to use NCD as a kernel in Section 3.2.

3.1.1 Pre-computing the Compression vector

When computing the value of $\text{NCD}(x, x')$, it is necessary to calculate the values of $\mathcal{C}(x)$ and $\mathcal{C}(x')$ as in Equation 1. For each $x \in X$, the $\mathcal{C}(x)$ would be computed repeatedly when computing the pairwise NCD distances between $x \in X$ and the elements in X'. Instead of recomputing the $\mathcal{C}(x)$ repeatedly, it is much more efficient to pre-compute the compressed versions of each element in X and X' only once. Since compressing the input samples is the most costly part of this computation, this saves substantial run-time as demonstrated in Section 4.

3.1.2 Symmetrisation

We propose three new ways to induce "more" adherence to the axioms in Equations 2–5 and compare their efficacy and run-time in Section 4.

For the purposes of the experiments below, the method proposed by Jiang *et al.* [38] is denoted "Vanilla" and computes the pairwise distances between two sets by naively computing every element of a distance matrix using the NCD function in Equation 1.

The second method, proposed here as a modification to NCD, assumes symmetry by only computing the lower triangular part of the distance matrix and then reflecting those values about the diagonal, instead of computing the entire distance matrix; this method is denoted "Assumed." Hence, in a distance matrix, D, the "Assumed" method computes the lower triangular part of D and then let

$$D_{i,j} = D_{j,i}, \tag{6}$$

which effectively halves the computational cost of computing the distance matrix.

In the third method, proposed here, symmetry is *enforced* by sorting the inputs of NCD alphanumerically before computing the distance between them. This approach thus ensures symmetry during prediction as well as training. This method is denoted "Enforced", and also effectively halves the computational cost of computing the distance matrix since again $D_{i,j} = D_{j,i}$.

The fourth method, also proposed here, computes the *average* value of $\mathrm{NCD}(x, x')$ and $\mathrm{NCD}(x', x)$. The average of $\mathrm{NCD}(x, x')$ and $\mathrm{NCD}(x', x)$ is

$$\overline{\mathrm{NCD}}(x, x') = \frac{\mathrm{NCD}(x, x') + \mathrm{NCD}(x', x)}{2}, \tag{7}$$

which can be simplified to

$$\overline{\mathrm{NCD}}(x, y) = \frac{\frac{\mathcal{C}(xx')+\mathcal{C}(x'x)}{2} - \min[\mathcal{C}(x), \mathcal{C}(x')]}{\max[\mathcal{C}(x), \mathcal{C}(x')]} + \varepsilon, \tag{8}$$

which clearly leads to $D_{i,j} = D_{j,i}$. The simplification in Equation 8 thus only includes one additional compression, increasing the computational cost by roughly 20% instead of doubling the computational cost as in Equation 7. This method is denoted "Averaged".

3.1.3 Zero-axiom check

A simple means to ensure that the output of NCD is zero when the two inputs are equal, we propose to check for this case before any distances are computed and return zero if the inputs are equal. We denote this modification the "Zero-axiom check". The Zero-axiom check was performed with the "Assumed" and "Enforced" methods, but not the "Averaged" method, assuming that the error associated with calculating $\mathrm{NCD}(x, x')$ would cancel out the error associated with $\mathrm{NCD}(x, x')$.

3.2 Kernelisation

We propose to use NCD to construct an approximate kernel, allowing NCD to be used with a much larger set of ML methods than as a distance. For this purpose, a kernel is defined as a function, $k : X \times X \to \mathbb{R}$, such that

$$k(x, x') := \langle \phi(x), \phi(x') \rangle \tag{9}$$

for all $x, x' \in X$, where $\phi : X \to Y$ is a feature function (a function extracting features from its inputs), and $\langle \cdot, \cdot \rangle$ denotes an inner product in the feature space, Y. The i-th row and j-th column of a kernel matrix, K, is

$$K_{ij} = k(x_i, x_j).$$

The radial basis function (RBF) kernel [40], also known as the Gaussian kernel (when the distance is the Euclidean distance) is defined as

$$k(x, x') = \exp\left(-\frac{d(x, x')^2}{\lambda}\right), \tag{10}$$

where λ is a tunable parameter (denoted a *length scale*) that controls how quickly the kernel function decreases as a function of the distance between points, *i.e.*, determines the influence of individual points on neighbouring points. We thus propose to use NCD as the distance function, d, in the kernel in Equation 10. The RBF kernel is particularly effective as it is known to be a universal function approximator [52].

The Hamming kernel [53], based on the Hamming distance between two strings or binary vectors, is defined as

$$k(x, x') = 1 - \lambda \frac{d_H(x, x')}{\max(|x|, |x'|)}, \tag{11}$$

where $d_H(x, x')$ denotes the Hamming distance, $\max(|x|, |x'|)$ denotes the length of the longer string, and λ is a tunable parameter that controls the sensitivity of the kernel to differences between input vectors. Smaller values of λ cause the kernel to decay more rapidly as the number of differing positions increases, thereby emphasizing exact or near-exact matches. We propose to use this kernel in settings where inputs are strings or binary representations, as it naturally captures similarity through positional agreement. Like the RBF kernel, the Hamming kernel belongs to the class of positive-definite kernels and has been shown to be effective at classification tasks [54].

Euclidean distances can be computed in the feature space by using a kernel. We see that

$$\begin{aligned} d(x, x') &= \|\phi(x) - \phi(x')\|_2^2 \\ &= \langle \phi(x) - \phi(x'), \phi(x) - \phi(x') \rangle \\ &= \langle \phi(x), \phi(x) \rangle + \langle \phi(x'), \phi(x') \rangle - 2\langle \phi(x), \phi(x') \rangle \\ &= k(x, x) + k(x', x') - 2k(x, x'), \end{aligned}$$

and denote this distance as the *kernel distance*. For the RBF kernel, when the symmetry and the zero axioms hold, we have that $k(x, x) = k(x', x') = 1$, and the kernel distance can be computed efficiently as

$$d_k(x, x') = 2 - 2k(x, x'). \tag{12}$$

This formulation was used to extend NCD-KNN to kernels, *i.e.*, the kernel distance in Equation 12 was used together with the RBF kernel in Equation 10, as well as with the Hamming kernel in Equation 11. Logistic regression and SVCs were trained on the kernel matrices formed from Equations 10 & 11.

3.3 Data

Several open datasets were used to evaluate the efficacy of NCD in the context of heterogeneous tabular and text data.

We used the KDD-NSL data, which is a log of system process data for both regular users (denoted benign) and malicious software (denoted adversarial) [55]. It includes 6072 samples and 41 features that encapsulate the behaviour of both benign software and malware. KDD-NSL includes software

protocol, system error rate, whether the process has root privileges, and the number of files accessed by the process.

We also used the DDoS IoT dataset [56], which includes information collected from network packet headers of adversarial and benign users across many types of DDoS attacks. Specific features include source IP address, source port, destination IP address, destination port, and network protocol among a total of 90 features across more than 40 million samples, collected from both benign users and malicious traffic.

We used the Truthseeker dataset [57], which includes 134 thousand messages from Twitter users with a label provided by the data distributors, and a label that encodes whether or not a given user was a suspected bot.

Finally, we used the SMS Spam dataset [58] which includes SMS messages and a label indicating whether or not a message is spam across 5575 samples.

For several of the datasets, malicious examples were rare compared to the number of benign examples. To address the class imbalances, each dataset was under-sampled [59] using the `imblearn` package [60] to reduce bias towards the majority class and to ensure metrics like accuracy are meaningful. For each dataset, model, symmetrisation method, and distance metric, 1000 samples from each dataset were used to conduct five-fold cross validation, yielding five disjoint validation sets of 200 samples each. Accuracy, distance matrix computation time, model training times, and prediction times were recorded for each of the five cross-validation folds. In an effort to represent both text and numerical data as strings, the rows of each numerical dataset were extracted as lists in `Python` and then those lists were cast directly to strings for the DDoS, KDD-NSL, and SMS Spam datasets.

3.4 Experiments

We evaluated the proposed methodology using the described datasets, models, symmetrisation methods ("Vanilla", "Assumed", "Enforced", and "Averaged"), and metrics (NCD_{gzip}, NCD_{bz2}, and NCD_{brotli}, Levenshtein distance, a normalised Levenshtein distance (labelled Lev Ratio), Hamming distance, and a normalised Hamming distance (labelled Ham Ratio)). After generating the 5-fold cross validation sets for each dataset, the distance matrices for each symmetrisation method and metric were computed, as outlined in Sections 2.3 and 3.1.2. Additionally, kernel matrices were computed as described in Section 3.2. The classifiers used were KNN, logistic regression, and SVC, as implemented in `scikit-learn` [61]. For KNN, we both used NCD

directly and used the kernel distance from Equation 12 computed using both the RBF and Hamming kernels. For (kernel) logistic regression and SVC, the RBF kernel in Equation 10 and the Hamming kernel in Equation 11 were used with the NCD distance.

Each model was tuned using the hyper-parameters outlined in the following. For NCD, all metrics (NCD, Hamming distance, Levenshtein distance, etc) were used to compute kernel matrices as per Equations 10 and 12. In addition, the Hamming kernel from Equation 11 was used as a baseline [53]. Both kernels have a hyper-parameter, λ, that must be tuned; λ was was evaluated in powers of 10 in the range $[10^{-3}, 10^{3}]$.

KNN requires the model builder to specify the number of nearest neighbours. In our experiments, $k \in \{1, 3, 5, 7, 11\}$, as odd numbers means there were no ties (for our binary classification task). In logistic regression, an ℓ_2 penalty was used as well as a configuration without any penalty. The coefficient of the penalty was set to powers of 10 in the range $[10^{-3}, 10^{3}]$. The `SAGA` solver [62] was used for logistic regression, with a tolerance of 10^{-4}. The penalty term in the SVC was varied in the range $[10^{-3}, 10^{3}]$ for each power of ten.

To find the most appropriate set of hyper-parameters, each of the dataset-model-symmetrisation-metric combinations enumerated above were evaluated using a grid search and 5-fold cross-validation. Then, the best-fit model was chosen for each dataset-model-symmetrisation-metric configuration by finding the configuration(s) with the highest mean cross-validation accuracy and choosing the model with the smallest standard deviation in the case ties. After fitting each model to each dataset, symmetrisation method, measure of distance, and model-dependent hyper-parameters, the best-fit models were repeatedly trained on $m \in \{10, 20, 35, 60, 100, 200, 500, 1000\}$ samples and evaluated against the 200 withheld samples during cross-validation to evaluate the ability of the model to generalise even when trained on a small number of samples.

In addition to the classification experiments above, the distance matrices calculated from those datasets were exhaustively checked for adherence to the axioms in Equations 2–4 and the percentage of violations for each axiom was recorded. In addition, 100k randomly selected 3-tuples from each distance matrix were sampled to compute the percentage of samples that violate Equation 5. An additional synthetic dataset comprised of 100k 3-tuples was generated from short, alphabetic strings using the uppercase English alphabet and a maximum string size of 144 characters (the length of a "tweet") for

each metric and symmetrisation method and used to calculate the probability of axiom violations.

4 Results and Discussion

In this section, the results from the aforementioned experiments are discussed.

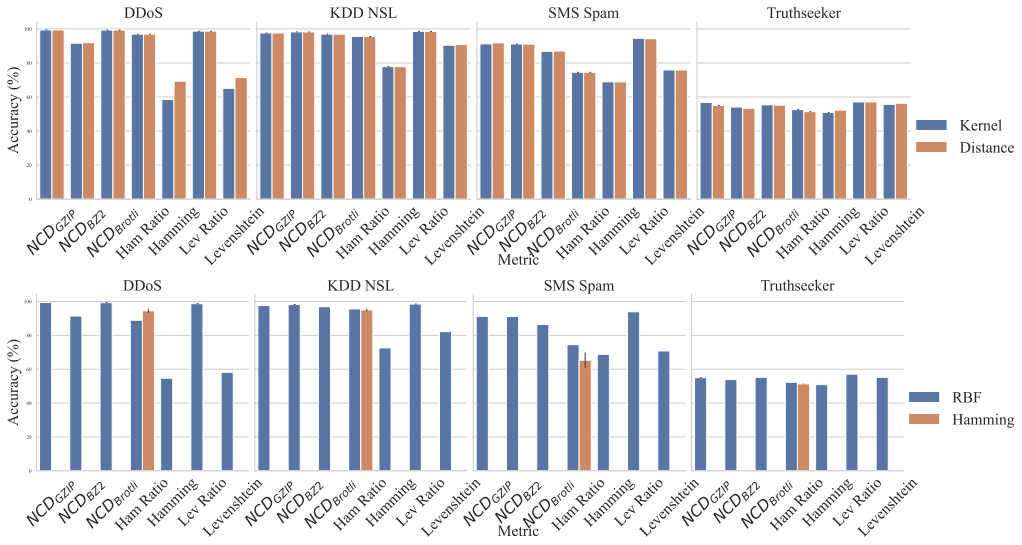

Figure 1: The 5-fold accuracy across each dataset (columns), distance metric (first-axis), whether or not the distance or kernel matrix was used (top plot, colour), kernel (top plot, colour), and symmetrisation method (bottom plot, colour). For both plots, the bars represent the mean accuracy and the error bars represent 95% confidence intervals.

Figure 1 depicts the mean accuracy of the best-fit model for each dataset-metric-model-kernel combination. The error bars reflect the 95% confidence interval of the mean accuracy, computed across five cross-validated folds. The top of Figure 1 compares accuracy of the proposed kernelised KNN to the distance-based KNN proposed by Jiang *et al.* [38] for a variety of string metrics (first axis). The bottom of Figure 1 compares the RBF kernel (Equation 10) to the Hamming kernel (Equation 11) for all three kernelised ML models. It is clear that the kernel method is consistent with the distance method (top of Figure 1), apart from the un-normalised Levenshtein and

Hamming distances, wherein the distance method is superior for the DDoS dataset. Additionally, the RBF kernel has accuracy that is consistent with the Hamming kernel (bottom of Figure 1) and at least one version of NCD outperforms the Hamming ratio in every dataset. Therefore, it is clear that the proposed kernelised classification model is as accurate at the distance-based KNN [38] (top of Figure 1) and can also outperform the Hamming kernel (bottom of Figure 1).

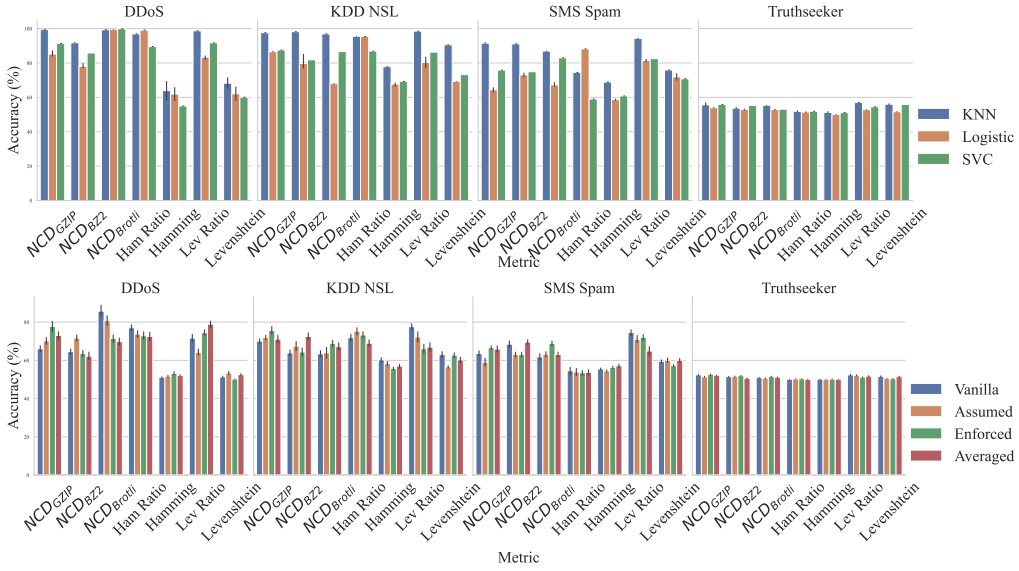

Figure 2: The accuracy across each dataset (columns), distance metric (first-axis), model (top plot, colour), and symmetrisation method (bottom plot, colour). For both plots, the bars represent the mean accuracy and the error bars represent 95% confidence intervals. Only KNN was used in the top plot as it is the only tested model that is meant to use distance matrices and the results depict the 5-fold accuracies for the best-fit model of each model-dataset-metric combination before (orange) and after kernelisation (blue).

Figure 2 shows the classifier performance across each dataset and string metric (including NCD, using various compressors) for both the kernel and distance matrices (top row), all models (middle row) and all symmetrisation methods (bottow row). The top sub-figure groups the results by whether or not the distance or kernel matrix was used, the middle sub-figure groups the results by model type using colour, and the bottom figure groups the results by symmetrisation method. From these results it is clear that KNN

is the most consistently accurate (middle of Figure 2) for NCD and that the distance matrix method tends to out-perform the kernel method (top of Figure 2).

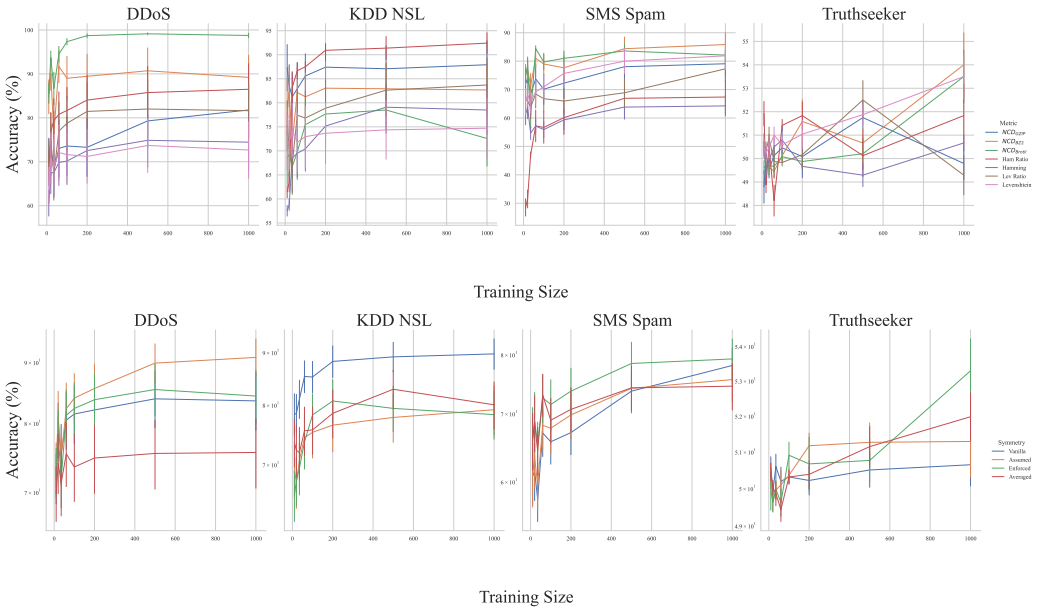

Figure 3: Accuracy across varying training sample sizes, computed on 200 samples withheld during cross-validation using the best-fit model for each model–metric–symmetrisation configuration. *Top:* Accuracy is grouped by evaluation metric (colour), showing how different metrics behave across datasets (columns) and training sizes, averaged over models and symmetrisation methods. *Bottom:* Accuracy is grouped by symmetrisation method (colour), illustrating how different symmetrisation choices impact performance across datasets, averaged over models and metrics. Each line represents the mean accuracy, and error bars indicate the 95% confidence interval of the mean.

Figure 3 depicts the accuracy of the best-fit models as the number of training samples is varied (first-axis). The top of Figure 3 groups the results by metric and the bottom groups the results by symmetrisation method, using colour to differentiate the groups. For the DDoS dataset, it is clear from the top subplot of Figure 3 that NCD improves the accuracy when a small number of samples are used to train the model compared to other string metrics. However, results are more mixed for other datasets. Additionally,

we can see that non-Vanilla symmetrisation methods tend to outperform the Vanilla variety on DDoS, SMS-Spam, and Truthseeker, although the confidence intervals overlap somewhat.

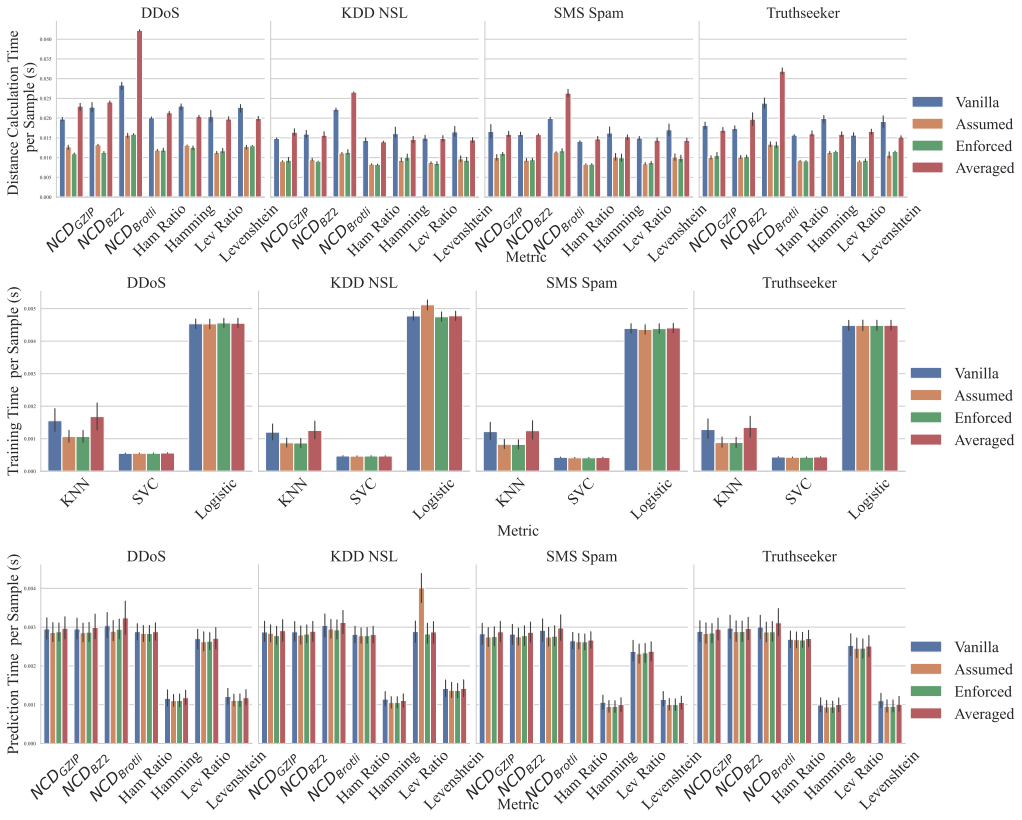

Figure 4: Performance times for various stages of the model evaluation process, computed across different metrics, datasets, and methods. *Top:* Distance matrix calculation time per sample averaged over all dataset-method-metric combinations. *Middle:* Training time per sample, measured after computing the distance matrix, averaged over all dataset-method-metric-model combinations. *Bottom:* Prediction time per sample, also after distance matrix computation, showing the time required for predictions averaged over all dataset-method-metric-model combinations. For all three sub-plots, the colour reflects the symmetrisation method. The firs-axis shows the metric in the top and bottom plots, but the middle plot depicts the training time as it varies more by model than metric. The top of each bar reflects the mean of the measured time and the error bars are 95% confidence intervals.

Figure 4 shows the run-times associated with calculating the distance between two samples (top), training the model on a single sample (middle), or inferring the classification of a new sample (bottom). The importance of the symmetrisation methods is to reduce the run-time. It is clear from Figure 4 that choice of symmetrisation method has a substantial effect on run-time, with the "Assumed" and "Enforced" symmetrisation methods taking roughly half as long per sample as the "Vanilla" version and the "Averaged" version taking slightly longer. During training (assuming the distance or kernel-matrix is pre-computed), the variance between models is far large than the variance induced by the symmetrisation methods (see middle row of Figure 4). During prediction, the differences between symmetrisation methods are much less pronounced, owing to the smaller number of samples—instead, the variance in prediction times is predominated by the run-time of the distance function as the variance within a given metric is much smaller than the variance between them (see the bottom row of Figure 4).

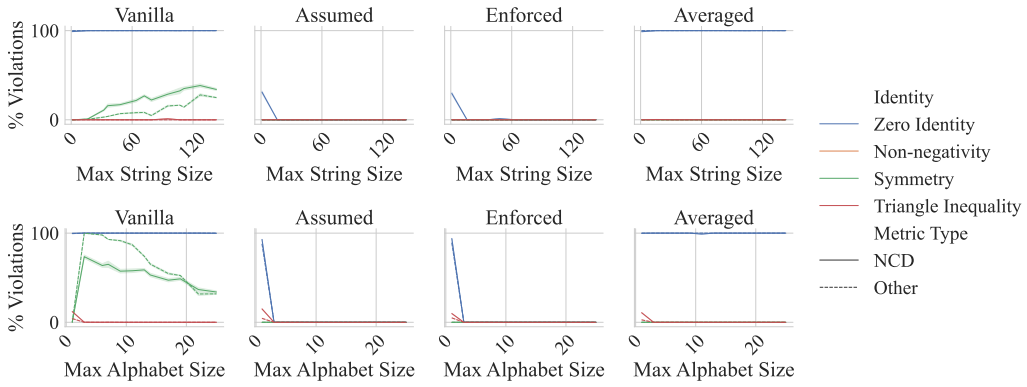

Figure 5: Percentage of examples found that violate the axioms outlined in Equations 2–5 using the vanilla symmetrisation method (top column), assumed symmetry (second column), enforced symmetry (third column), and averaged (bottom column) methods on 100 thousand random strings generated from the standard English alphabet. Unless otherwise specified by the first axis in an individual plot, *Max String Size* was 144 characters (the length of a "tweet") and the *Alphabet Size* was 26. Each line marker corresponds to a different axiom as outlined in Equations 2–5 and each colour corresponds to either NCD (Equation 1) or some other string metric (Section 2.2).

Figures 5–6 display how the symmetrisation methods influence to what

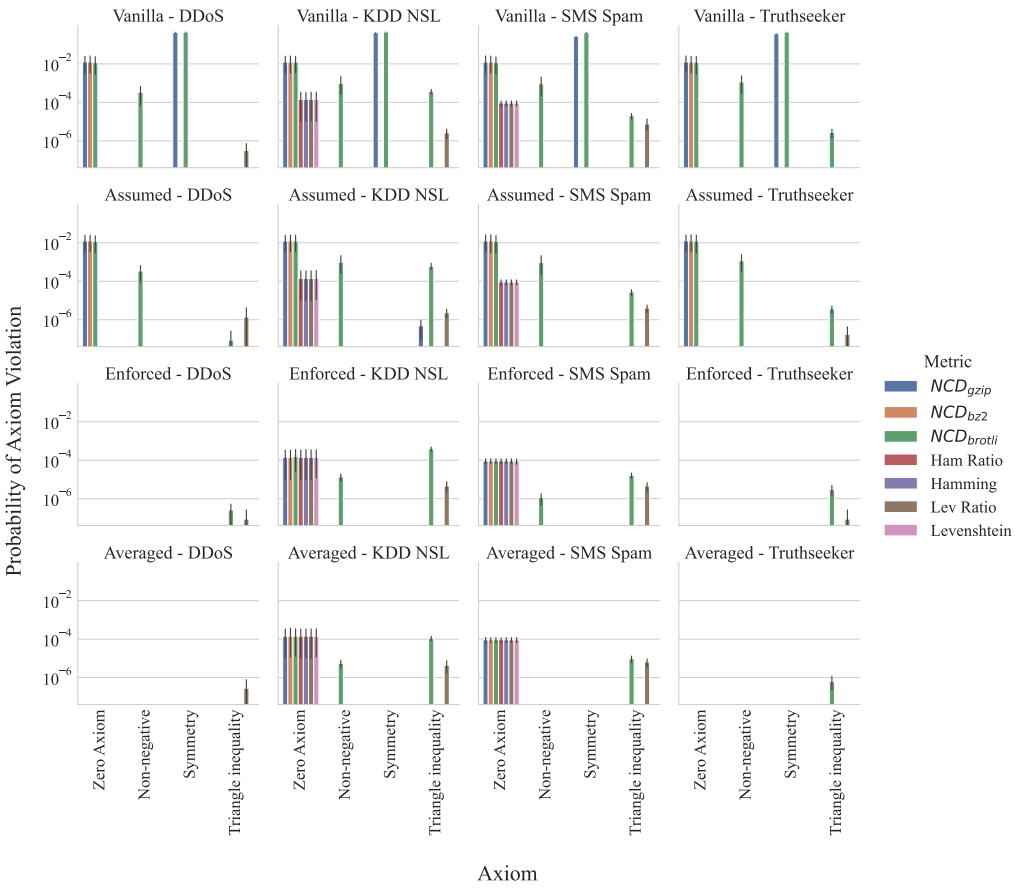

Figure 6: Percentage of examples found that violate the assumptions outlined in Equations 2–5 using the vanilla (top row), assumed symmetry (second row), enforced (third row), and averaged (bottom row) methods on the training matrices for each of the outlined datasets. Each column is dedicated to a dataset and each model is given a column. The first axis displays which of the axioms is violated and the colour indicates which symmetrisation method was used. Since evaluating all possible distance 3-tuples would be computationally infeasible for even hundreds of samples, three samples were sampled 100 thousand times such that no 3-tuple was repeated.

degree a given metric adheres to the axioms outlined in Equations 2–5 and show that they enforce adherence to the symmetric and zero-axiom as expected. It is clear that the proposed modifications make NCD behave more

like a metric and more consistently adhere to the axioms in Equations 2- 5. This is true on randomly constructed strings (Figure 5) as well as strings collected from the aforementioned datasets (Figure 6). It is clear that this does not correspond to a decrease in accuracy (see Figure 2) and that these modifications can significantly improve run-time (see Figure 4).

5 Limitations

While the methods presented in this work demonstrate strong performance across various tasks and datasets, several limitations should be acknowledged.

To further improve run-time performance, compression algorithms optimised for graphics processing units (GPUs) have been developed [63]. These are likely to outperform the CPU-based implementations used in this paper when applied to large-scale datasets. Incorporating such GPU-accelerated compressors could significantly reduce processing time and improve scalability.

The model tuning parameters used in this study were limited in scope. While the selected configurations were sufficient to demonstrate key insights, a broader hyperparameter search may reveal additional trade-offs or improvements in accuracy and efficiency.

This work focused on a specific set of compression algorithms. Although other compressors exist–including those optimised for specific data types [63, 64, 65]—-they were considered out of scope for this study. Future work could explore the impact of alternative compressors on both accuracy and run-time.

The preprocessing step used for heterogeneous tabular data was intentionally kept simple—each row was cast to a Python list then cast again as a string. While effective for our purposes (see Figure 1), this approach is crude and may obscure structural or semantic relationships within the data by including extraneous characters like commas and brackets (used to delineate and denote a list in Python, respectively). As always, the best implementation will be determined by the data and its schema. More sophisticated encoding methods—such as schema-aware parsing or embedding techniques—could improve performance, particularly on more complex or high-dimensional datasets.

6 Conclusion

Overall, we see that NCD is at least as accurate as other string metrics (top of Figure 2), despite not being a true metric (Lemma 1). Furthermore, we see that the proposed symmetrisation methods are quite effective—sometimes even outperforming the "Vanilla" method (bottom of Figure 2). Kernelised NCD is effective even when only a small number of samples are used to train the model, making it ideal for lightweight, client-side deployments (top of Figure 3) and our extensions (Section 3.1) do not significantly change this result (bottom of Figure 3). NCD can reach more than 95% accuracy on even tens of samples (Figure 3). It is clear from Figure 4 that the "Assumed" and "Enforced" symmetrisation methods proposed in this work are superior to those found in the literature by decreasing the run-time without penalising accuracy (bottom of Figure 2). In some cases, the "Averaged" symmetrisation method offers superior accuracy over the other methods (Figure 2, bottom) while inducing a marginal cost of only a few milliseconds (Figure 4, top). In other cases, "Enforced" and "Assumed" offer both superior accuracies (Figure 2, bottom) and run-times (Figure 4, top and bottom).

The proposed model is a real-time, client-side classification method that can be trained on a very small number of samples—potentially collected from only a single user. This reduces the attack surface to only adversaries that have access to data that users generally consider private (*e.g.*, the contents of a message). That is, the attack surface is reduced and can be unique to each user, session, or device by using data from each user in isolation and training a model on that user's device. Client-side models would only need to share a single bit (the classification label in a binary classification context) with the platform operator—allowing those operators to serve user-specific contents without exfiltrating private data from their entire user base. However, not training a model on large amounts of collected user data means that such hypothetical client-side methods are at risk of not performing at the level of state-of-the-art large-scale methods. The end result is a model with a substantially reduced attack surface that is nevertheless accurate, but also simple, generally applicable, and very fast.

References

[1] R. Desislavov, F. Martínez-Plumed, J. Hernández-Orallo, Compute and energy consumption trends in deep learning inference, arXiv preprint arXiv:2109.05472 (2021).

[2] H. Abelson, R. Anderson, S. M. Bellovin, J. Benaloh, M. Blaze, J. Callas, W. Diffie, S. Landau, P. G. Neumann, R. L. Rivest, J. I. Schiller, B. Schneier, V. Teague, C. Troncoso, Bugs in our pockets: the risks of client-side scanning, Journal of Cybersecurity 10 (1) (Jan. 2024).

[3] Goldman Sachs Research, AI is poised to drive 160% increase in data center power demand, Goldman Sachs (May 2024).
URL `https://www.goldmansachs.com/insights/articles/AI-poised-to-drive-160-increase-in-power-demand`

[4] M. Nouwens, I. Liccardi, M. Veale, D. Karger, L. Kagal, Dark patterns after the GDPR: Scraping consent pop-ups and demonstrating their influence, in: Proceedings of the 2020 CHI conference on human factors in computing systems, 2020.

[5] A. Chakraborty, M. Alam, V. Dey, A. Chattopadhyay, D. Mukhopadhyay, Adversarial attacks and defences: A survey, arXiv:1810.00069 [cs, stat] (2018).

[6] C. Meyers, T. Löfstedt, E. Elmroth, Safety-critical computer vision: An empirical survey of adversarial evasion attacks and defenses on computer vision systems, Artificial Intelligence Review (2023).

[7] Amnesty International, Encryption—a matter of human rights (2016).
URL `https://www.amnesty.nl/content/uploads/2016/03/160322_encryption_-_a_matter_of_human_rights_-_def.pdf?x12112`

[8] Apple Inc., CSAM detection: A technical summary (2021).
URL `https://www.apple.com/child-safety/pdf/CSAM_Detection_Technical_Summary.pdf`

[9] C. A. Choquette-Choo, F. Tramer, N. Carlini, N. Papernot, Label-only membership inference attacks, in: International conference on machine learning, PMLR, 2021, pp. 1964–1974.

[10] M. Fredrikson, S. Jha, T. Ristenpart, Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures, in: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security - CCS '15, ACM Press, Denver, Colorado, USA, 2015.

[11] Z. Li, Y. Zhang, Membership leakage in label-only exposures, in: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, 2021, pp. 880–895.

[12] T. Orekondy, B. Schiele, M. Fritz, Knockoff nets: Stealing functionality of black-box models, in: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2019, pp. 4954–4963.

[13] M. Jagielski, N. Carlini, D. Berthelot, A. Kurakin, N. Papernot, High accuracy and high fidelity extraction of neural networks, in: 29th USENIX security symposium (USENIX Security 20), 2020, pp. 1345–1362.

[14] J. R. Correia-Silva, R. F. Berriel, C. Badue, A. F. De Souza, T. Oliveira-Santos, Copycat cnn: Stealing knowledge by persuading confession with random non-labeled data, in: 2018 International joint conference on neural networks (IJCNN), IEEE, 2018, pp. 1–8.

[15] R. Shokri, M. Stronati, C. Song, V. Shmatikov, Membership inference attacks against machine learning models, in: 2017 IEEE Symposium on Security and Privacy (SP), IEEE, 2017, pp. 3–18.

[16] A. Rawat, K. Levacher, M. Sinn, The devil is in the gan: backdoor attacks and defenses in deep generative models, in: European Symposium on Research in Computer Security, Springer, 2022, pp. 776–783.

[17] R. Shokri, et al., Bypassing backdoor detection algorithms in deep learning, in: 2020 IEEE European Symposium on Security and Privacy (EuroS&P), IEEE, 2020, pp. 175–183.

[18] T. Gu, B. Dolan-Gavitt, S. Garg, Badnets: Identifying vulnerabilities in the machine learning model supply chain, arXiv preprint arXiv:1708.06733 (2017).

[19] A. Saha, A. Subramanya, H. Pirsiavash, Hidden trigger backdoor attacks, in: Proceedings of the AAAI conference on artificial intelligence, Vol. 34, 2020, pp. 11957–11965.

[20] H. Aghakhani, D. Meng, Y.-X. Wang, C. Kruegel, G. Vigna, Bullseye polytope: A scalable clean-label poisoning attack with improved transferability, in: 2021 IEEE European symposium on security and privacy (EuroS&P), IEEE, 2021, pp. 159–178.

[21] A. Turner, D. Tsipras, A. Madry, Clean-label backdoor attacks (2018).

[22] A. Shafahi, W. R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, T. Goldstein, Poison frogs! targeted clean-label poisoning attacks on neural networks, Advances in neural information processing systems 31 (2018).

[23] J. Geiping, L. Fowl, W. R. Huang, W. Czaja, G. Taylor, M. Moeller, T. Goldstein, Witches' brew: Industrial scale data poisoning via gradient matching, arXiv preprint arXiv:2009.02276 (2020).

[24] H. Souri, L. Fowl, R. Chellappa, M. Goldblum, T. Goldstein, Sleeper agent: Scalable hidden trigger backdoors for neural networks trained from scratch, Advances in Neural Information Processing Systems 35 (2022) 19165–19178.

[25] N. Carlini, D. Wagner, Towards evaluating the robustness of neural networks, in: 2017 IEEE symposium on security and privacy (sp), Ieee, 2017, pp. 39–57.

[26] E. Dohmatob, Generalized No Free Lunch Theorem for Adversarial Robustness, in: Proceedings of the 36th International Conference on Machine Learning, Vol. 97 of PMLR, 2019.

[27] J. Chen, M. I. Jordan, M. J. Wainwright, HopSkipJumpAttack: A query-efficient decision-based attack, in: IEEE symposium on security and privacy (sp), IEEE, 2020, pp. 1277–1294.

[28] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, F. Roli, Evasion attacks against machine learning at test time, 2013, pp. 387–402.

[29] S.-M. Moosavi-Dezfooli, A. Fawzi, P. Frossard, Deepfool: a simple and accurate method to fool deep neural networks, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 2574–2582.

[30] B. Biggio, B. Nelson, P. Laskov, Poisoning Attacks against Support Vector Machines, 2013.

[31] A. Aljuhani, Machine learning approaches for combating distributed denial of service attacks in modern networking environments, IEEE Access 9 (2021).

[32] P. M. Santos, B. Manoj, M. Sadeghi, E. G. Larsson, Universal adversarial attacks on neural networks for power allocation in a massive mimo system, IEEE Wireless Communications Letters 11 (1) (2021) 67–71.

[33] M. Jagielski, N. Carlini, D. Berthelot, A. Kurakin, N. Papernot, High accuracy and high fidelity extraction of neural networks, in: 29th USENIX security symposium (USENIX Security 20), 2020, pp. 1345–1362.

[34] J. W. Bentley, D. Gibney, G. Hoppenworth, S. K. Jha, Quantifying membership inference vulnerability via generalization gap and other model metrics, arXiv preprint arXiv:2009.05669 (2020).

[35] L. Mitrou, Data protection, artificial intelligence and cognitive services: is the general data protection regulation (gdpr)'artificial intelligence-proof'?, Artificial Intelligence and Cognitive Services: Is the General Data Protection Regulation (GDPR)'Artificial Intelligence-Proof (2018).

[36] M. Marks, C. E. Haupt, Ai chatbots, health privacy, and challenges to hipaa compliance, Jama (2023).

[37] C. Meyers, M. R. S. Sedghpour, T. Löfstedt, E. Elmroth, A training rate and survival heuristic for inference and robustness evaluation (trashfire), arXiv preprint: arXiv:2401.13751 [cs.LG] (2024).

[38] Z. Jiang, M. Y. Yang, M. Tsirlin, R. Tang, J. Lin, Less is more: Parameter-free text classification with gzip, arXiv preprint arXiv:2212.09410 (2022).

[39] M. Li, X. Chen, X. Li, B. Ma, P. Vitanyi, The similarity metric, IEEE Transactions on Information Theory 50 (12) (2004).

[40] S. Shalev-Shwartz, S. Ben-David, Understanding machine learning: From theory to algorithms, Cambridge university press, Cambridge, UK., 2014.

[41] M. Scilipoti, M. Fuster, R. Ramele, A strong inductive bias: Gzip for binary image classification, arXiv preprint arXiv:2401.07392 (2024).

[42] J. Opitz, Gzip versus bag-of-words for text classification with knn, arXiv preprint arXiv:2307.15002 (2023).

[43] J. Weinreich, D. Probst, Parameter-free molecular classification and regression with gzip (2023).

[44] K. Nishida, R. Banno, K. Fujimura, T. Hoshide, Tweet classification by data compression, in: Proceedings of the 2011 international workshop on DETecting and Exploiting Cultural diversiTy on the social web, 2011, pp. 29–34.

[45] M. Cebrián, M. Alfonseca, A. Ortega, Common pitfalls using the normalized compression distance: What to watch out for in a compressor (2005).

[46] Free Software Foundation, GZIP: Gnu zip, https://www.gnu.org/software/gzip/manual/gzip.html (2009–2023).

[47] muraroa.demon.co.uk, bz2 (Jul. 1998).
URL `https://web.archive.org/web/19980704181204/http://www.muraroa.demon.co.uk/`

[48] Google Inc., Github - google/brotli: Brotli compression format.
URL `https://github.com/google/brotli`

[49] D. Burago, A course in metric geometry, American Mathematical Society (2001).

[50] rapidfuzz, Levenshtein, https://rapidfuzzrap.github.io/Levenshtein/ (2021).

[51] G. Navarro, A guided tour to approximate string matching, ACM computing surveys (CSUR) 33 (1) (2001) 31–88.

[52] B. Hammer, K. Gersmann, A note on the universal approximation capability of support vector machines, neural processing letters 17 (2003) 43–53.

[53] K. T. Phelps, M. LeVan, Kernels of nonlinear hamming codes, Designs, Codes and Cryptography 6 (3) (1995) 247–257.

[54] M. Norouzi, D. J. Fleet, R. R. Salakhutdinov, Hamming distance metric learning, Advances in neural information processing systems 25 (2012).

[55] G. Mohi-ud din, NSL-KDD, IEEE Dataport (2018).

[56] E. C. P. Neto, S. Dadkhah, R. Ferreira, A. Zohourian, R. Lu, A. A. Ghorbani, Ciciot2023: A real-time dataset and benchmark for large-scale attacks in iot environment, Sensors 23 (13) (2023) 5941.

[57] M. Khalil, M. Azzeh, Fake news detection models using the largest social media ground-truth dataset (truthseeker), International Journal of Speech Technology (2024) 1–16.

[58] T. Almeida, J. Hidalgo, SMS Spam Collection, UCI Machine Learning Repository (2011).

[59] S. Yen, Y. Lee, Under-sampling approaches for improving prediction of the minority class in an imbalanced dataset, Lecture notes in control and information sciences 344 (2006) 731.

[60] G. Lemaître, F. Nogueira, C. K. Aridas, Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning, Journal of Machine Learning Research 18 (17) (2017).

[61] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.

[62] A. Defazio, F. Bach, S. Lacoste-Julien, Saga: A fast incremental gradient method with support for non-strongly convex composite objectives, Advances in neural information processing systems 27 (2014).

[63] R. A. Patel, Y. Zhang, J. Mak, A. Davidson, J. D. Owens, Parallel lossless data compression on the GPU, IEEE, 2012.

[64] K. Brandenburg, Mp3 and aac explained, in: Audio Engineering Society Conference: 17th International Conference: High-Quality Audio Coding, Audio Engineering Society, 1999.

[65] V. Sze, M. Budagavi, G. J. Sullivan, High efficiency video coding (HEVC), Integrated circuit and systems, algorithms and architectures 39 (2014) 40.