Tiny, Hardware-Independent, Compression-based Classification

C. Meyers[1,2], A. P. MacSween, E. Elmroth[2], and T. Löfstedt[2]

[1]Institute for Experiential AI, Northeastern University, Boston, United States
[2]Department of Computer Science, Umeå University, Umeå, Sweden
E-MAIL: c.meyers@cs.umu.se

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 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 normalised compression distance) to measure the distance between generic objects in classical distance-based machine learning methods. In this work, we demonstrate that the normalised 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 demonstrate that the normalised compression distance works as well as and sometimes better than other metrics and kernels—while requiring only marginally more 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.

Keywords:

Machine Learning; Trustworthy Machine Learning; Privacy-Preserving Machine Learning; Client-Side Learning; Edge Learning; Normalised Compression Distance

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 proposing to incorporate them into 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. In addition, we show that our proposed method reduces the run-time relative to the baseline by a factor of approximately 50% while simultaneously improving the accuracy significantly.

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 [2]. Privacy experts have

denounced such approaches for numerous reasons: the ease of *extracting* private training data [9] or the model itself [10, 11], the ability of a malicious user to induce false positives for other users when the model is trained on user data [*poisoning* attacks; 12, 13, 14], and the triviality of simply *evading* the detection mechanism [15, 16, 17, 5, 18].

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 criticised by privacy experts [2]. While, in some cases, personally identifiable information can be removed, applications like spam detection IP address, user name, and message contents are precisely the data we need to classify spam and shut down botnets. Beyond the inherent privacy concerns, examples of security risks include attacks against ML systems that target a model during training [19], prediction [15], and deployment [20]. Even when access to a model by an adversary is limited, it is possible to induce a misclassification [17], reverse engineer the model [21], determine the model weights [22], or infer the class-membership of new samples [23]. This raises profound questions for safety-critical systems [24] and legal questions about access and control of the underlying data [25]. Even if the attacker only has access to a typical application programming interface (API), there are reliable ways to fool the model [17].

Many privacy experts warn of the potential for any hypothetical *centralised* 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 light-weight, client-side approach to content filtering that does not rely on large-scale data collection.

Recently, Jiang *et al.* [26] proposed a remarkably successful approach to "parameter free" classification, dubbed NCD-KNN, that exploits a compression-based distance measure, the *normalised compression distance* [NCD; 27], to classify objects using the k-nearest neighbours (KNN) method [28]. NCD-KNN is known to work well even when trained on small numbers of samples [29]. 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 [30], molecular property classification [31], and text classification [32], 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 [27] included an error term that is usually ignored in recent research [30, 31, 32, 26].

The original authors [27] 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) [33].

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, in part, been explored before [33], we expand the analysis by Cebrián *et al.* [33] to more recent compression algorithms and also offer additional run-time improvements over previous implementations [26].

The NCD-KNN method has shown very strong performance across several benchmarks, but prior implementations [26] are not appropriate for real-time settings due to redundant computations. We therefore propose several modifications in Section 3.1.

Further, we show that NCD is not a metric [30, 31, 32, 26, 33], 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 [30, 31, 32, 27, 26] 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.

2 Background

In the sections below, we define and describe the NCD, outline the NCD-KNN method proposed by Jiang *et al.* [26], outline several other string metrics, and discuss the NCD distance matrix and how to efficiently compute it.

2.1 Normalised Compression Distance

NCD has been demonstrated to be a *universal* measure of similarity between two objects [27]—where a value of 0 denotes equivalence and a value of 1 denotes complete dissimilarity. The NCD is defined as [27]

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

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 [27]. The error term is usually assumed to be small relative to the other term.

NCD requires a choice of compression algorithm, and here we evaluated the `gzip` [34], `bz2` [35], and `brotli` [36] compressors. To distinguish between the use of these different compressors, a subscript is used such that NCD_{gzip} denotes the use of the `gzip` compressor.

While NCD is often discussed as a measure of distance [30, 31, 32, 26, 27], 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 $x_1, x_2, x_3 \in X$ [37]:

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

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

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

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

Much of the literature devoted to NCD has treated it as a proper metric [30, 31, 32, 26]. However, as we show now, it is not, which can lead to erroneous classifications or violated assumptions [38].

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.

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

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

□

2.2 Other String Metrics

To model datasets that comprise strings, several existing measures of distance between strings are routinely used [39]. 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 [40]. *Hamming* is the number of character positions where two strings differ. *Hamming Ratio* is the number of character positions where two strings differ, divided by the length of the longer string (denoted "Ratio" in the figures).

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 $\text{NCD}(x, x')$, it is necessary to calculate the values of $\mathcal{C}(x)$ and $\mathcal{C}(x')$. To classify a sample, Jiang *et al.* [26] 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 Jiang's NCD-KNN method [26] and the proposed modifications to NCD 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 [26] becomes inefficient because it includes many repeated computations. To minimise run-time, we first propose a simple modification to pre-compute the compressed length of all input strings and caching them. 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. In addition, because $\text{NCD}(x, x')$ can behave strangely when $x = x'$, we also propose a check for this special case and return 0, thus complying with the zero-axiom.

For the purposes of the experiments below, the method proposed by Jiang *et al.* [26] 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. Larger modifications to the "Vanilla" method discussed by Jiang *et al.* [26] are proposed in the following subsections. We propose several modifications that symmetrise the distance matrix, intended to ensure adherence to the symmetry axiom (Equation 4), as discussed in Section 3.1.1. Finally, we outline the proposed way to use NCD as a kernel in Section 3.2.

3.1.1 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.

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 $\text{NCD}(x, x')$ and $\text{NCD}(x', x)$. The average of $\text{NCD}(x, x')$ and $\text{NCD}(x', x)$ is

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

which can be simplified to

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

which clearly leads to $D_{i,j} = D_{j,i}$. The simplification in Equation 8 thus only includes one additional compression and only requires the lower-triangular distance matrix to be computed. Hence, the computational cost is 66.67% of the cost of the Vanilla method, and not a doubling of the computational cost as Equation 7 suggests. This method is denoted "Average".

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

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

The radial basis function (RBF) kernel [28], 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 [41].

The Hamming kernel [42], 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 [43].

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 and Hamming kernels, when the symmetry and the zero axioms hold, we known 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 regressors and SVCs were trained on the kernel matrices formed from Equations 10 & 11. The KNN models were trained using a matrix calculated with Equation 12 for both the Hamming and RBF kernels.

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) [44]. It includes 6,072 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 [45], 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 [46], 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 [47] which includes SMS messages and a label indicating whether or not a message is spam across 5,575 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 [48] using the `imblearn` package [49] to reduce bias towards the majority class and to ensure metrics like accuracy are meaningful.

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. This crude conversion was intentionally adopted as a baseline representation, allowing us to evaluate model performance without introducing additional preprocessing heuristics or feature engineering that might confound comparisons across disparate datasets.

3.4 Experiments

We evaluated the proposed methodology using the described datasets, models, symmetrisation methods ("Vanilla", "Assumed", "Enforced", and "Average"), and metrics (NCD_{gzip}, NCD_{bz2}, and $\text{NCD}_{\text{brotli}}$, Levenshtein distance, Hamming distance, and a normalised Hamming distance (labelled "Ratio")). After generating the 5-fold cross-validation sets for each dataset-metric-symmetrisation combination, the distance matrices for each dataset-metric-symmetrisation combination method were computed and cached, as outlined in Sections 2.3 and 3.1. 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` [50].

Each model was tuned using the hyper-parameters outlined in the following. Both the RBF and Hamming kernels have a hyper-parameter, λ, that had to 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. We evaluated $k \in \{1, 3, 5, 7, 11\}$, as odd numbers means there were no ties (for the 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 [51] 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 on the training data. However, the accuracies and times reported below are calculated on 200 *test* samples withheld from the training and cross-validation processes for the best-fit model for each dataset-model-symmetrisation-metric combination.

All experiments were run on an Apple M4 Pro with 12 cores rather than a data-center CPU so that timing measurements reflect (an admittedly high-end) client device.

4 Results and Discussion

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

Figure 1 depicts the test accuracy across all dataset-metric-model-kernel combinations. The left and centre subplots of Figure 1 depict the test accuracies for each best-fit dataset-metric-model-kernel-symmetrisation combination, and the error bars indicate the 95% confidence interval. The left subplot of Figure 1 clearly indicates that kernel methods (blue) tend to outperform distance-based methods when using NCD with various compressors (orange). Furthermore, the left subplot shows that that kernelised NCD is often significantly better than string metrics or their kernelised counterparts. The centre subplot of the same figure clearly shows that for all metrics besides "Ratio" (the only metric for which the Hamming kernel is defined), that the RBF kernel significantly outperforms the baseline Hamming kernel. Additionally, the centre subplot shows that NCD is significantly more accurate than any of the baseline string metrics since compressors encode more semantic and string frequency data that metrics alone. The right subplot validates the performance of the proposed method by depicting the test accuracies for the best-fit model using kernelised NCD, clearly indicating that kernelized NCD models are often more accurate than NCD-KNN.

Figure 2 shows the classifier performance across each dataset and string metric in terms of run-time (left) and accuracy (right), with the error bar indicating the 95% performance for all best-fit dataset-metric-model-kernel-symmetrisation combinations. It is clear from the right subplot of Figure 2 that choice of symmetrisation method has a substantial effect on run-time, with the "Assumed", "Enforced", and "Average" symmetrisation methods taking roughly half as long per sample as the "Vanilla" version. It is clear from the left subplot of Figure 2 that the "Assumed", "Enforced", and "Average" symmetrisation methods proposed in this work are superior to those found in the literature by decreasing the run-time without significantly penalising accuracy (Figure 2, right).

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 [52]. 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. This work focused on a specific set of compression algorithms. Although other compressors exist—including those optimised for specific data types [53, 54]—they were considered out of scope for this study. Likewise, the data 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 white spaces, commas, and brackets (used to delineate and denote a list in Python, respectively). More sophisticated encoding methods—such as schema-aware parsing or embedding techniques—could improve performance, particularly on more complex or high-dimensional datasets, in particular for the "Truthseeker" dataset [55].

6 Conclusion

Overall, we see that NCD is at least as accurate as other string metrics (left subplot of Figure 1), despite not being a true metric (Lemma 1). Furthermore, we see that the proposed sym-

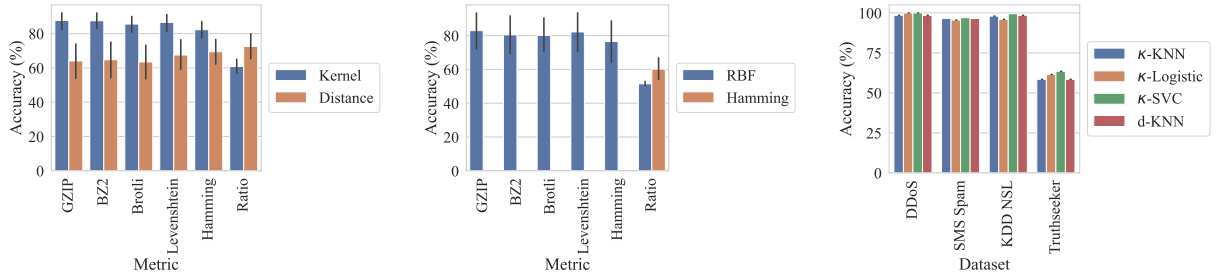

FIGURE 1. This figure depicts the accuracy broken down by string metric (left and centre plots) as well as by dataset and model (right plot). The left and centre plots depict the test accuracy for all best-fit dataset-model-symmetrisation-metric combinations with the error bar indicating the 95% interval. The left plot compares the performance between models that use kernel methods (blue) and models that use distance alone (orange, KNN only). The centre plot compares the RBF kernel (blue) to the Hamming kernel (orange)—the latter is only defined for the "Ratio" metric. In contrast, the right plot depicts test accuracy on *only* the best NCD-kernelised model (denoted with a "κ-" prefix) and the best distance-based KNN proposed by Jiang *et. al.* [26] (denoted with a "d-" prefix).

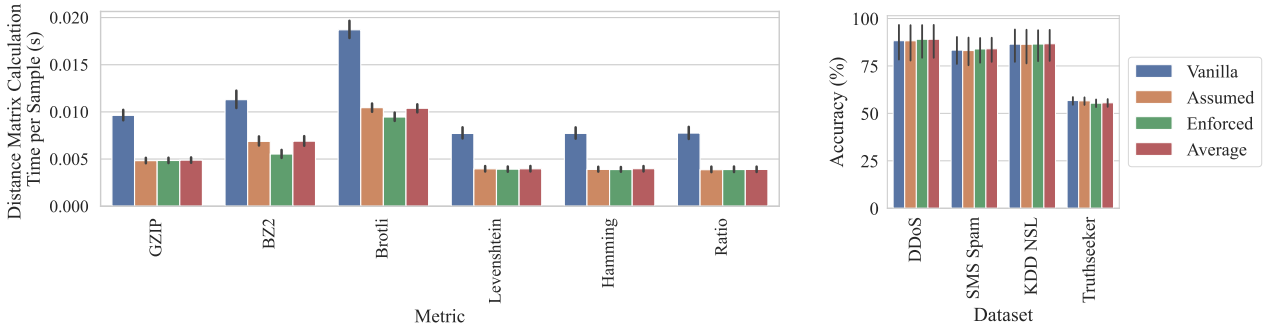

FIGURE 2. The distance calculation time (left) and accuracy (right) across each dataset (columns), distance metric (first-axis, left plot), dataset (first-axis, right plot), model (left plot, colour), and symmetrisation method (right plot, colour). For both plots, the bars represent the mean and the error bars represent 95% confidence intervals and colour represents the symmetrisation method. The left plot was calculated using 1000 training samples for each metric-algorithm combination. The right plot was calculated using the test accuracies and 95% confidence interval for the best fit NCD-kernelised model, calculated for each of the symmetrisation-data-model-compressor combination.

metrisation methods are quite effective—sometimes even outperforming the "Vanilla" method with respect to accuracy(right subplot of Figure 2). Additional run-time improvements are significant and obvious (left subplot of Figure 2).

Furthermore, because NCD is known to work well on a small number of samples [27], each model can be unique to each user, session, or device by using data from each user in isolation. The proposed model is a real-time, client-side classification method that can be trained quickly—potentially on data collected from only a single user and stored only on their device. The proposed client-side classifier limits the attack surface to only adversaries that have access to data that users generally keep pri-

vate (*e.g.*, the contents of a message, system data, or network traffic). The end result is a model with a marginal attack surface that is nevertheless accurate, simple, generally applicable, and fast enough to be trained entirely on a generic client device.

References

[1] R. Desislavov, F. Martínez-Plumed, J. Hernández-Orallo, Trends in AI inference energy consumption: Beyond the performance-vs-parameter laws of deep learning, Sustainable Computing: Informatics and Systems 38 (2023).

[2] European Commission, Proposal for a regulation of the european parliament and of the council laying down rules to prevent and combat child sexual abuse, https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=COM:2022:209:FIN (2022).

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

[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, Association for Computing Machinery, 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, Massively parallel evasion attacks and the pitfalls of adversarial retraining, EAI Endorsed Transactions on Internet of Things 10 (2024).

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

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

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

[10] 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.

[11] 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), 2018.

[12] 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, 2022.

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

[14] 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).

[15] N. Carlini, D. Wagner, Towards evaluating the robustness of neural networks, in: 2017 IEEE symposium on security and privacy (SP), 2017.

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

[17] J. Chen, M. I. Jordan, M. J. Wainwright, HopSkipJumpAttack: A query-efficient decision-based attack, in: IEEE symposium on security and privacy (SP), 2020.

[18] C. Meyers, M. R. S. Sedghpour, T. Löfstedt, E. Elmroth, A training rate and survival heuristic for inference and robustness evaluation (trashfire), in: 2024 International Conference on Machine Learning and Cybernetics (ICMLC), 2024.

[19] B. Biggio, B. Nelson, P. Laskov, Poisoning attacks against support vector machines, arXiv preprint: arXiv:1206.6389 [cs, stat] (2013).

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

[21] 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.

[22] 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.

[23] 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).

[24] C. Meyers, T. Löfstedt, E. Elmroth, Safety-critical computer vision: An empirical survey of adversarial evasion attacks and defenses on computer vision systems, Springer Artificial Intelligence Review 56 (2023).

[25] M. Marks, C. E. Haupt, AI chatbots, health privacy, and challenges to HIPAA compliance, JAMA 330 (4) (2023).

[26] 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).

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

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

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

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

[31] J. Weinreich, D. Probst, Parameter-free molecular classification and regression with Gzip, chemRxiv, DOI: https://doi.org/10.26434/chemrxiv-2023-v1s2s-v2 (2023).

[32] 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.

[33] M. Cebrián, M. Alfonseca, A. Ortega, Common pitfalls using the normalized compression distance: What to watch out for in a compressor, Communications in Information and Systems 5 (4) (2005).

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

[35] J. Seward, The bzip2 home page, https://web.archive.org/web/19980704181204/http://www.muraroa.demon.co.uk/ (1998).

[36] Google Inc., google/brotli: Brotli compression format, https://github.com/google/brotli.

[37] D. Burago, Y. Burago, S. Ivanov, A course in metric geometry, American Mathematical Society, 2001.

[38] S. Fadel, S. Ghoniemy, M. Abdallah, H. A. Sorra, A. Ashour, A. Ansary, Investigating the effect of different kernel functions on the performance of svm for recognizing arabic characters, Int. J. Adv. Comput. Sci. Appl 7 (1) (2016) 446–450.

[39] M. Bachmann, Levenshtein, https://rapidfuzz.github.io/Levenshtein (2021).

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

[41] B. Hammer, K. Gersmann, A note on the universal approximation capability of support vector machines, Neural Processing Letters 17 (2003).

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

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

[44] G. Mohi-ud din, NSL-KDD, IEEE Dataport, https://ieee-dataport.org/documents/nsl-kdd (2018).

[45] 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).

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

[47] T. Almeida, J. Hidalgo, SMS Spam Collection, UCI Machine Learning Repository, https://archive.ics.uci.edu/dataset/228/sms+spam+collection (2011).

[48] S.-J. Yen, Y.-S. Lee, Under-sampling approaches for improving prediction of the minority class in an imbalanced dataset, in: D. Huang, K. Li, G. W. Irwin (Eds.), Intelligent Control and Automation, Vol. 344, Springer, Berlin, 2006.

[49] 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).

[50] 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).

[51] 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).

[52] R. A. Patel, Y. Zhang, J. Mak, A. Davidson, J. D. Owens, Parallel lossless data compression on the GPU, in: 2012 Innovative Parallel Computing (InPar), 2012.

[53] K. Brandenburg, MP3 and AAC explained, in: Audio Engineering Society Conference: 17th International Conference: High-Quality Audio Coding, 1999.

[54] V. Sze, M. Budagavi, G. J. Sullivan (Eds.), High efficiency video coding (HEVC): Algorithms and Architectures, Vol. 39 of Integrated Circuits and Systems (ICIR), Springer, Cham, 2014.

[55] R. S. Arslan, D. Çelik, Prediction of fake news on x: An analysis of llms and machine learning methods on truthseeker, in: 2025 7th International Congress on Human-Computer Interaction, Optimization and Robotic Applications (ICHORA), IEEE, 2025, pp. 1–5.