Multi-GPU processing of unstructured data for machine learning

Joel Ratsaby* Alexander Timashkov
Electrical and Electronics Engineering Department
Ariel University
Ariel, ISRAEL
Email: ratsaby@ariel.ac.il*

***Abstract*— We introduce a method for processing unstructured data for machine learning based on an LZ-complexity string distance. Computing the LZ-complexity is inherently a serial data compression process; hence, we introduce a string distance computed by a parallel algorithm that utilizes multiple GPU devices to process unstructured data, which typically exists in large quantities. We use this algorithm to compute a distance matrix representation of the unstructured data that standard learning algorithms can use to learn. Our approach eliminates the need for human-based feature definition or extraction. Except for some simple data reformatting done manually, our proposed approach operates on the original raw data and is fully automatic. The parallel computation of the distance matrix is efficient. It obtains a speed-up factor of 528 in computing the distance matrix between every possible pair of 16 strings of length 1M bytes. We show that for learning time-series classification, relative to the ubiquitous TFIDF data representation, the distance-matrix representation yields a higher learning accuracy for most of a broad set of learning algorithms. Thus, the parallel algorithm can be helpful in efficiently and accurately learning from unstructured data.**

***Index Terms*—LZ-complexity, string distance, multi-GPU, CUDA**

I. Overview

The majority of real-world data is unstructured; that is, it does not consist of pre-defined attributes or a fixed number of features and can come as a mix of different forms of data, such as text, general character-based, or any binary sequences of varying lengths. Data mining is based on machine learning algorithms, which require structured data with a fixed number of pre-defined features. For instance, in learning the classification of textual data, the standard and ubiquitous approach is to represent text as finite-dimensional numerical vectors based on the TFIDF word representation [1], which requires a pre-defined finite dictionary of possible words or tokens based on word stems. More generally, when dealing with raw data such as binary streams of complex time-dependent multi-sensor measurements, there is no well-defined dictionary of possible 'words' or pre-defined patterns. Thus, to learn such unstructured data, one needs a more general data representation that can preferably be computed automatically without human expert intervention to reduce processing time and cost.

In this paper, we propose a parallel algorithm for automatically processing unstructured data for machine learning via any standard learning algorithm. Our approach to learning unstructured data is as follows: given raw data that consists of general character or binary sequences that may be of different lengths, we compute a distance matrix of all possible pairs of data instances where the distance is based on an information-theoretic notion of the LZ-complexity of a string. The distance matrix forms a new representation of the original data in which an instance is represented as a numerical vector of dimensionality equal to the number of instances in the data set hence it can be large. To tackle this, we introduce a highly scalable and efficient parallel algorithm to compute this matrix and thereby convert any unstructured data that consists of instances of general sequences into a structured representation that any standard machine learning algorithm can learn. While LZ-complexity is a concept that lies at the heart of universal data compression algorithms, the significance of the current paper is in the area of machine learning rather than data compression.

The parallel algorithm scales well with multiple GPUs and achieves high speed-up factors, for instance, for a 16×16 matrix of $1M$ strings, we obtain a speed factor of 528 relative to the serial algorithm. We compare the generalization accuracy of learning classification of unstructured time-series data between the proposed LZ-distance matrix representation and the ubiquitous TFIDF data representation. The LZD data representation yields significantly higher machine-learning classification accuracies than when the data is represented by an alternative ubiquitous data representation based on TFIDF vectors.

II. Introduction

Lempel-Ziv complexity of a finite string [2] is a complexity measure inspired by the universal compression algorithms of Lempel Ziv [3], [4]. Computing the LZ complexity of a string (a sequence of bytes) is inherently a serial process. The algorithm traverses the string one byte at a time in a serial manner, and based on the bytes seen up to the present, it tries to reproduce a maximal length substring from the remaining yet unseen part of the string. It then adds this substring to the seen part and continues iteratively until all of the string has been reproduced.

LZ-complexity has recently been used in pattern recognition and classification through the introduction of a string distance function which is used for sequence analysis in bioinformatics [5]. A main advantage of this string distance is that it applies to any data type, character or byte, and it can measure the dissimilarity between a pair of strings of different lengths, which is very useful for dealing with unstructured data, that is, data which is not based on a fixed dimensionality and predefined attributes. For instance, in [6], [7], a universal image distance (UID) based on the LZ complexity is used for learning gray-scale image classification by subdividing an image into multiple segments of pixels that are represented as short strings of bytes. However, for a string of length n, computing LZ-complexity in the standard serial manner requires $O(n^2)$ time, which is impractical for machine learning. We note that the computational complexity of the family of compression algorithms, such as LZ77, LZSS, etc., which are based on the idea of the LZ-complexity is only $O(n)$ because they use a fixed length sliding-window over the string. In contrast, this cannot be done in the algorithm for computing LZ-complexity since, in general, it would not yield the true value of the LZ-complexity.

In [8], a parallel algorithm is introduced to compute the LZ-complexity of strings restricted to a single GPU block of 1,024 parallel execution threads and is limited to specific data consisting of gray-scale images. In contrast, the current paper achieves significantly higher speed-up factors for any type of unstructured data. It builds on the work of [9] who introduces a parallel algorithm for computing the LZ-complexity of arbitrarily long finite strings (limited only by the memory size of the GPU) of *any* type, character, or byte, and can execute any number of blocks in parallel, limited only by the hardware; for instance, on a DGX workstation it utilizes 160 blocks of 1,024 threads in parallel. It is reported there that this algorithm's parallel execution efficiency is 98.95%. This high efficiency enables us in the current paper to compute an LZ-complexity based string-distance (described in Section V as Algorithm LZD) in a reasonable time for long strings. For instance, it takes 30 seconds to compute 16 pairwise distances (each of which requires 3 LZ-complexity computations) between pairs of strings of length 1M byte.

In the current paper, our approach to learning unstructured data is as follows: given a data set of general sequences, character-based or binary, of possibly different lengths, an LZD matrix of distances between all possible pairs of instances is computed. This distance matrix forms a new representation of the original data in which an instance is newly represented as a numerical vector of dimensionality equal to the number of instances in the data set.

Unlike the ubiquitous TFIDF representation of textual data [1] used in natural language data mining and machine learning which requires *a priori* information about the data, e.g., a dictionary of words (or tokens/stems), the LZD representation enables to learn from unstructured data, including but not limited to text, with no *a priori* knowledge about the underlying data where any data instance is regarded just as a sequence of bytes of any finite length that may differ from one instance to another. Our parallel algorithm for computing LZD is a significant advantage for machine learning and data mining, as the majority of real-world data is unstructured and requires computational acceleration in data analytics [10].

The algorithm utilizes as many GPUs as available on a machine and scales well with problem size. For experiments, we use an nVIDIA DGX station with four V100 GPUs. The speed-up in computing an LZD matrix of 16 strings of length 1M bytes is 528.

The LZ distance can be applied not only to textual data, where strings have a delimiting symbol that separates words but to any strings of contiguous bytes that are not necessarily separated into words, namely, any binary sequences representing arbitrary sequential data. Thus, Algorithm LZD enables efficient and automatic transformation of general unstructured data into a representation that can be used by standard machine learning algorithms.

III. LZ-complexity

As introduced by [2], the Lempel-Ziv complexity of a finite string S is proportional to the minimal number of substrings that are necessary to produce S via a simple copy operation. For instance, the string $S = ababcabcabcbaa$ can be constructed from the five sub-strings, a, b, abc, $abcabcb$, aa and therefore its LZ-complexity equals 5. The standard serial algorithm for computing the LZ-complexity of a string of length n takes $O(n^2)$ time. The definition of this complexity [2], [5] is as follows: let S,Q and R be strings of bytes that are defined over the alphabet $\mathcal{A}$. Denote by $l(S)$ the length of S, and $S(i)$ denotes the i^{th} element of S. We denote by $S(i, j)$ the sub-string of S, which consists of bytes of S between position i and j (inclusive). An extension $R = SQ$ of S is reproducible from S (denoted as $S \rightarrow R$) if there exists an integer $p \leq l(S)$ such that $Q(k) = R(p + k - 1)$ for $k = 1, \dots, l(Q)$. For example, $aacgt \rightarrow aacgtcgtcg$ with $p = 3$ and $aacgt \rightarrow aacgtac$ with $p = 2$. R is obtained from S (the seed) by first copying all of S and then copying in a serial manner $l(Q)$ elements starting at the p^{th} location of S in order to obtain the Q part of R.

A string S is *producible* from its prefix $S(1, j)$ (denoted $S(1, j) \Rightarrow S$), if $S(1, j) \rightarrow S(1, l(S) - 1)$. For example, $aacgt \Rightarrow aacgtac$ and $aacgt \Rightarrow aacgtacc$ both with pointers $p = 2$. The production adds an extra 'different' byte at the end of the copying process, which is not permitted in reproduction.

Any string S can be built using a *production process* where at its i^{th} step we have the production $S(1, h_{i-1}) \Rightarrow S(1, h_i)$ where h_i is the location of a byte at the i^{th} step. (Note that $S(1, 0) \Rightarrow S(1, 1)$).

An m-step production process of S results in parsing of S in which $H(S) = S(1, h_1) \cdot S(h_1 + 1, h_2) \cdots S(h_{m-1} + 1, h_m)$ is called the *history* of S and $H_i(S) = S(h_{i-1} + 1, h_i)$ is called the i^{th} component of $H(S)$. For example, for $S = aacgtacc$ we have $H(S) = a \cdot ac \cdot g \cdot t \cdot acc$ as the history of S.

If $S(1, h_i)$ is not reproducible from $S(1, h_{i-1})$ then the component $H_i(S)$ is called *exhaustive* meaning that the copying process cannot be continued, and the component should be halted with a single byte *innovation*. Every string S has a unique exhaustive history [2].

Let us denote by $c_H(S)$ the number of components in a history of S. The LZ complexity of S is defined as $c(S) = \min\{c_H(S)\}$ where the minimum is over all histories of S. It can be shown that $c(S) = c_E(S)$ where $c_E(S)$ is the number of components in the exhaustive history of S.

A graphical processing unit (GPU) is a multi-processor electronic chip specialized for matrix and vector operations for fast 3D graphics. GPUs have been recently gaining popularity also for non-graphical processing due to their highly parallel computing architecture. This architecture typically consists of thousands of execution cores that operate concurrently on several gigabytes of global memory. We use CUDA as the Application Program Interface and parallel execution run time. A kernel is a program that is executed by every GPU thread in parallel. It is launched by a program on the host using CUDA command. When the host code launches a kernel, the CUDA run-time system generates a grid of threads that are organized into a two-level hierarchy. Each grid is organized as an array of thread blocks, which is referred to as a block. All blocks of a grid are of the same size, and each block has up to 1024 threads.

We start by presenting the serial LZ-complexity algorithm in the next section.

IV. Serial algorithm

The standard LZ-complexity algorithm [2], [5] is executed serially and is displayed as Algorithm 1. As it scans the input string S, it searches using Procedure 2 for a maximal length word that can be produced from the current position based on any byte in the history, then adds it as a new component and augments the history iteratively until it reaches the end of S. The number of components produced is the LZ-complexity of S.

While using data structures, such as hashtables, it is possible to make the search in Procedure 2 more efficient, we use the basic version of the serial LZ-complexity algorithm, which serves as a fair comparison against the parallel algorithm, which is introduced in the next section. (The parallel algorithm also employs basic arrays, as data structure optimization is beyond the scope of the current paper.)

V. Distance

Typically, machine learning algorithms operate on the assumption that data is well-structured and that it is possible to formalize quantitative features of the data that can be encoded by numerical variables organized as finite-dimensional feature vectors. Learning with unstructured data, for instance, multimedia data, textual data, or time-series data, requires the ability to learn data without pre-defined features. Learning from such featureless data is a major challenge in machine learning research, and one of the more promising approaches is measuring dissimilarity between data instances [11]. The dissimilarity between a pair of data points can be represented mathematically by the distance function's value. The LZ-complexity of a string may be used to define a distance function that measures the dissimilarity of two finite strings of possibly different lengths. Such strings may represent different types of unstructured data, for instance, bioinformatic sequences [5], images [8], [12], [13], and time-series data [14].

In [5], several string-distances based on the LZ-complexity are described. For a pair of strings X, Y, let us denote by $c(X)$, $c(Y)$, and $c(XY)$ the LZ-complexity of X, Y and their concatenation XY, respectively. In the current paper, we use the following distance,

$$d(X,Y) := \frac{c(XY) - \min\{c(X), c(Y)\}}{\max\{c(X), c(Y)\}}, \tag{1}$$

which is used in [6] for learning classification and clustering of strings that represent images. It is a normalized distance, taking a value in the unit interval, because

$$\begin{aligned} c(XY) - \min\{c(X), c(Y)\} &\leq c(X) + c(Y) - \min\{c(X), c(Y)\} \\ &= \max\{c(X), c(Y)\} \end{aligned}$$

and it is not a metric since a value 0 implies that the two strings are close but not necessarily identical.

Algorithm 3 uses the parallel LZ-complexity algorithm of [9] to compute the distance (1) between two strings. In the next section, we report on the computational execution times of Algorithm 3 and in Section VII, we use it to pre-process featureless data, specifically time series, and then apply standard machine learning algorithms to learn time-series classification.

VI. Computational efficiency

In this section, we evaluate the speed-up factor of Algorithm 3 which is implemented in C with the CUDA API [15] on an nVIDIA DGX station with Intel Xeon E5-2698 v4 2.2 GHz that has 20 cores (40 threads) and four V100 GPUs, each with 32G bytes of global memory and up to 96Kb of shared memory.

As mentioned above, Algorithm 3 uses the parallel LZ complexity algorithm of [9], whose speed-up factor for computing the LZ complexity of strings is estimated to grow with respect to string length n at a rate of $n^{2/3}$ at least for all $n \leq$ 6M bytes.

In this section, we compare the computational time of an $N \times N$ matrix whose entries are LZ distances of pairs of N strings, each of length $n =$ 1M bytes, using a single CPU core that evaluates the LZ distance using the serial Algorithm 1 versus the parallel Algorithm 3 using four GPUs. In both the serial and the parallel versions, the computation is divided into two stages: in the first stage, the LZ-complexity of each of the N strings is computed. In the second stage, the LZ-complexity of each of the N^2 possible concatenations of a pair of strings, is computed. In the first stage, each of the four GPUs computes the LZ-complexity of $N/4$ strings of length 1M bytes (from the set of N strings), and in the second stage, the task of computing the LZ-complexity of each of the N^2 strings of length 2M bytes is divided equally amongst the four

Algorithm 1
SerialLZ(S)

```
1: Input: $S := \{S(0), S(1), \ldots, S(n-1)\}$ // string $S$
2: $p := 1$, $hs := 0$, $he := 0$, $c := 1$ // Initialize present $p$ to $S(1)$, history start $hs$ and history end $he$ to $S(0)$, and LZ-complexity $c$
3: while $p < n$ do
4:   $\{found, max\} :=$ Search$(S, hs, he, p, n)$
5:   if $found = TRUE$ then
6:     // word found is $\{S(p), \ldots, S(p + max - 1)\}$, make it a component, add to history
7:     $he := he + max$
8:     $c := c + 1$ // increment number of components
9:     $p := he + 1$ // update present position to after the new word
10:  else
11:    // no word found, make $S(p)$ be new component, add it to history
12:    $he := he + 1$
13:    $c := c + 1$ // increment number of components
14:    $p := p + 1$ // update the present position to $S(p + 1)$
15:  end if
16: end while
17: Output: $c$ // LZ-complexity of $S$
```

Procedure 2
Search(S, hs, he, p, n)

```
1: // $S$ is input string of length $n$, history start and end positions $hs$, $he$, and present position $p$
2: $found := FALSE$, $max := 0$
3: for $hs \leq i \leq he$ do
4:   // Search for starting point $i$ in history that can reproduce maximal component starting at $S(p)$
5:   if $S(i) = S(p)$ then
6:     $found := TRUE$ // found starting point
7:     $j := i$
8:     $k := p$
9:     // create a word starting at $S(p)$
10:    while $j < n$ do
11:      $k := k + 1$
12:      $j := j + 1$
13:      if $S(j) \neq S(k)$ then
14:        // end of word
15:        if $j - i + 1 > max$ then
16:          // word's length is the new max
17:          $max := j - i + 1$
18:          break// from the while loop
19:        end if
20:      end if
21:    end while
22:  end if
23: end for
24: return $\{found, max\}$
```

Algorithm 3
LZD($S1, S2$)

```
1: Input: strings $S1$, $S2$
2: $S = S1 \cdot S2$ // Concatenated string
3: $c1 = ParallelLZ(S1)$ // Use algorithm of [9]
4: $c2 = ParallelLZ(S2)$
5: $c = ParallelLZ(S)$
6: $dist = (c - \min\{c1, c2\}) / \max\{c1, c2\}$
7: Output: $dist$ // LZ-complexity distance $d(S1, S2)$
```

GPUs. At the end of the computation, a matrix of all pairwise distances is obtained.

Table I displays the execution time results and speed-up factors. For instance, if $N = 4$, the speed-up factor is 419 for four GPUs versus a single CPU core. By Amdahl's Law, taking the number of parallel execution units to be the number of threads, which is four times $160K$, this amounts to a reduction in the percentage of serial execution time down to approximately 0.24%. Therefore, the percentage of time spent in executing parallel code is 99.76%. For the range $N = 4$, 8, 12, 16 this parallel utilization percentage ranges between 99.76% and 99.81%.

VII. Machine learning

In this section, we apply Algorithm 3 to learn classification of unstructured data. Given a data set of instances, each of which is a string, for example, each is a text document; we define an *LZD matrix* to be a matrix whose ij^{th} entry is the distance (1) between the i^{th} and j^{th} strings, as computed by Algorithm 3. As mentioned above, one of the benefits of this distance is that it can measure dissimilarity between two strings of different lengths, which is very useful for unstructured data, in particular, for time series.

We use an LZD distance matrix as an alternative representation of time-series data sets. These data sets consist of a variety of numerical multidimensional time series ranging from 3 to

TABLE I
LZD NxN DISTANCE MATRIX COMPUTATION TIMES TS, TP, FOR SERIAL AND PARALLEL ALGORITHMS, RESPECTIVELY.

N=4	TS	TP
Stage 1 execution time		
Total (4 $1M$-strings)	16.31 min	3 sec
Average per string	4.08 min	2.75 sec
Stage 2 execution time		
Total (16 $2M$-strings)	3.22 hr	27 sec
Average per string	12.07 min	1.69 sec
Total time: Stages 1 + 2	**3.49 hr**	**30 sec**
Speed Factor: 419		

N=8	TS	TP
Stage 1 execution time		
Total (8 $1M$-strings)	32.57 min	6 sec
Average per string	4.07 min	3 sec
Stage 2 execution time		
Total (64 $2M$-strings)	14.28 hr	1.7 min
Average per string	13.39 min	1.59 sec
Stages 1 + 2	**14.82 hr**	**1.8 min**
Speed Factor: 494		

N=12	TS	TP
Stage 1 execution time		
Total (12 $1M$-strings)	49.09 min	9 sec
Average per string	4.09 min	2.5 sec
Stage 2 execution time		
Total (144 $2M$-strings)	33.33 hr	3.77 min
Average per string	13.88 min	1.57 sec
Stages 1 + 2	**34.15 hr**	**3.92 min**
Speed Factor: 523		

N=16	TS	TP
Stage 1 execution time		
Total (16 $1M$-strings)	1.09 hr	12 sec
Average per string	4.07 min	2.63 sec
Stage 2 execution time		
Total (256 $2M$-strings)	59.98 hr	6.73 min
Average per string	14.06 min	1.58 sec
Stages 1 + 2	**61.07 hr**	**6.93 min**
Speed Factor: 528		

approximately 50 dimensions (each dimension corresponds to a numerical measurement of some sensors). This numerical time-series data is considered to be featureless because a numerical value at an instant of time does not constitute a feature value of an instance for machine learning (an instance needs to consist of the behavior along a whole time interval). We run several algorithms to learn time-series classification and compare the classification accuracy between our proposed LZD-matrix representation of the data and the standard ubiquitous Term Frequency Inverse Document Frequency (TFIDF) [1] data representation.

To start, we describe a simple preprocessing stage to convert a time series data into a symbolic representation. We first take the raw numerical time series data, normalize and discretize it such that each normalized numerical value at any time instant is transformed into one of ten characters, A, B, ..., J. So with n numerical attributes, we obtain a word of length n for each time instant in the time series. We insert a blank character between every word (while it is not needed for the LZD matrix representation, we still do this as it is needed for the TFIDF representation of the data). This way, we obtain a sequence of words such that each word is of length equal to the dimensionality n of the time series.

Next, we partition the sequence into subsequences by breaking it into words that are delimited by a space character. This is unnecessary for the LZD matrix representation since the LZ-complexity is well-defined for any data sequence, even for binary or character sequences with no word-delimiting characters. We do this solely for the TFIDF representation, which needs sequences of words (or word stems). Each subsequence is then considered as a single learning instance (since every standard learning algorithm trains on a set of instances).

This gives a symbolic representation of the original numerical time series data where now a learning instance is an ‘artificial document’ that consists of ‘artificial words’ in the A to J alphabet. We experiment with several variants, where in each variant, we choose a different value for the length of an instance. There is a trade-off: the more instances, the shorter the sequence in each instance. Standard learning algorithms treat instances as independent, so ideally, an instance should not be too short to capture a sufficient amount of the time-dependency structure in the original time-series data. And it should not be too long to have sufficient size for training data. This human intervention is only needed because we compare the LZD to the TFIDF representation. In operational mode (non-test mode), the LZD algorithm is fully automatic and can be applied to the original data or its character sequence representation; no artificial words need to be defined, and no human fine-tuning is needed.

Once the original time-series data is transformed into a symbolic representation of artificial documents, a simple script generates the LZD matrix representation and the TFIDF representation of the original time series.

Our experiments are organized as follows: we have four data sets from the UCI data-set learning repository [16]; see the Appendix for reproducibility information. We create several variants from each one. The data sets are labeled by their acronyms as follows: DSDD is “Dataset for Sensorless Drive Diagnostics”, which consists of measurements of electric current drive signals with intact and defective components. There are 11 different categories that describe the condition of a drive. LDPA is “Localization Data for Person Activity Data Set“ and UIWA is “User Identification From Walking Activity“, both of which consist of measurements made by sensors on a human body. LDPA has 11 classification categories, which indicate the type of activity that the person did at any particular time instance and UIWA has 22 classification categories. GSATM is “Gas sensor array temperature modulation”, which consists of gas sensor measurements for detecting various chemicals. We used 12 of the 13 different time series as data.

Table II displays the size of the data sets after the preprocessing stage that converts a time series into an artificial document. For each variant we write N x M x K where N is the number of instances (artificial documents), M is

TABLE II
DIFFERENT VARIANTS OF THE FOUR DATA SETS, DSDD, LDPA, GSATM, AND UIWA, WHERE FOR EACH VARIANT, WE HAVE N X M X K, WHERE N IS THE NUMBER OF INSTANCES, M IS THE NUMBER OF WORDS PER INSTANCE, AND K IS THE NUMBER OF BYTES PER WORD.

Dataset	Variant 1	Variant 2	Variant 3
DSDD	33 x 1,773 x 48	99 x 591 x 48	297 x 197 x 48
LDPA	829 x 200 x 3	556 x 300 x 3	334 x 500 x 3
GSATM	708 x 5,000 x 19	360 x 10,000 x 19	240 x 15,000 x 19
UIWA	756 x 200 x 3	509 x 300 x 3	309 x 500 x 3

the number of words per instance, and K is the number of characters per word (where the bytes are from the alphabet A to J). Note that data sets LDPA and UIWA consist of three-dimensional numerical time series based on positioning values in 3D space; hence the transformed data consists of three-letter words as indicated by the value of K on the second and fourth rows of Table II.

Figure 1 shows an example of two instances from Variant 1 of the LDPA data set; each is a row of 200 three-letter words. Each variant is stored as a folder with files, one per class category and each file consists of all the instances from that category.

For each variant of every data set, we compute an LZD matrix corresponding to the distances of every pair of instances in that variant. We use Algorithm 3, which as mentioned above, uses the parallel LZ-complexity algorithm of [9], to compute the distance (1) between every pair of instances and store each instance as a row in a file such that the i^{th} row is a numerical vector whose j^{th} entry is the distance (1) between the i^{th} and j^{th} instances. We refer to this file as the LZD matrix representation of a variant of a data set. For example, for Variant 1 of LDPA, there are 829 rows in the file, where the i^{th} row has 829 numerical values that correspond to the distances between the i^{th} instance and all 829 instances, followed by the class category of the i^{th} instance. This file is then used as data for all the learning algorithms that we test. To compute the LZD matrices for DSDD, it takes a few minutes for Variant 1, about 15 minutes for Variant 2, and about 40 minutes for Variant 3. For LDPA and UIWA, the approximate times are: 4 hours for Variant 1, 2 hours for Variant 2 and 40 minutes for Variant 3. For GSATM, it takes approximately 18 hours to compute the LZD matrix of Variant 1, 6 hours for Variant 2, and 4 hours for Variant 3.

In general, for larger data sizes, in order to reduce the LZD computation time one can choose to compute the distance between every instance in the data with every instance in a randomly chosen subset of the data, which yields a smaller non-square LZD matrix.

The TFIDF data representation for each variant of every data set is obtained as follows: we create a single file that consists of all instances in every artificial document of a variant. This file contains the instances of all class categories. We then apply WEKA's [17] filter *StringToWordVector*, which converts strings of words into the TFIDF vector representation.

The above constitutes all of the pre-processing that is needed to transform time-series data into the two representations, the LZD matrix and the TFIDF representations.

TABLE III
LEGEND OF THE LEARNING PERFORMANCE RESULTS

	Variant	Data representation
(a)	1	LZD
(b)	1	TFIDF
(c)	2	LZD
(d)	2	TFIDF
(e)	3	LZD
(f)	3	TFIDF

TABLE IV
LEARNING CLASSIFICATION RESULTS FOR DATA-SET DSDD

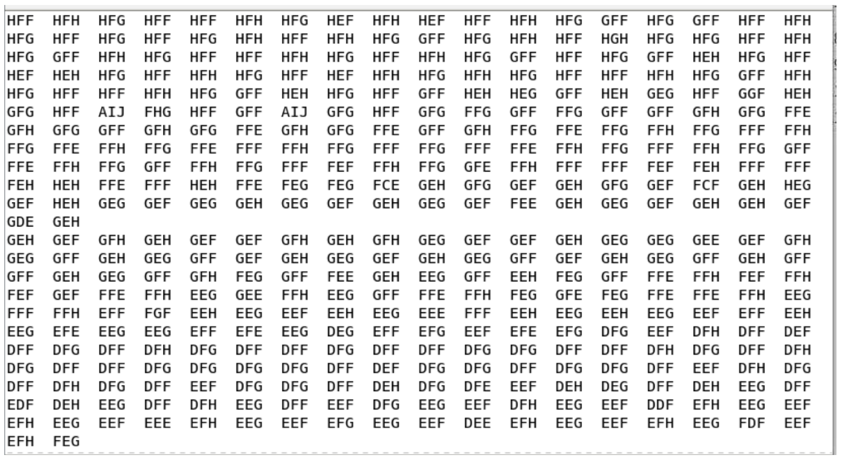
	a	b	c	d	e	f
a	-	0 (0)	18 (2)	7 (1)	29 (21)	18 (6)
b	27 (19)	-	30 (26)	27 (14)	30 (30)	27 (26)
c	12 (0)	0 (0)	-	2 (2)	28 (28)	14 (6)
d	23 (3)	2 (0)	28 (18)	-	30 (28)	28 (15)
e	1 (0)	0 (0)	2 (1)	0 (0)	-	4 (3)
f	12 (3)	3 (1)	16 (6)	2 (1)	26 (24)	-

Next, we describe the learning problems. Each variant of every data set defines a classification learning problem. We use WEKA as a machine-learning platform. We choose a range of algorithms [18] that includes: k-NN based on Euclidean distance between two instances, with different values for k in the range $k = 1, 3, 5, 7, 10$ and with three possible weight functions (used for weighing each instance), standard (no weight), inverse (1/Euclidean distance), $1-$Euclidean distance. The other algorithms are Naive-Bayes, C5.0 classification tree learning, multi-layer perceptrons (MLP), SVM with two different kinds of kernels, and three kinds of calibrations. For assessing the learning accuracy, in each learning problem, we do a 10-fold cross validation.

In total, 30 algorithms are used on each of the three variants of every one of the four data sets. An experiment is defined as a single machine learning algorithm that is executed on a single variant of a data set.

Tables IV, V, VI and VII display the learning performances on each of the variants of every data set, respectively. The legend is displayed in Table III. When looking at any of the tables, the $(i, j)^{th}$ entry represents the number of algorithms (out of the 30 in total) for which the variant specified in column j outperformed the variant specified in row i. We compare pairs (i, j) of LZD variant $j \in \{1, 2, 3\}$ against variant $i \in \{1, 2, 3\}$ of TFIDF, by picking column (a) and row (b), or, column (c) and row (d), or, column (e) and row (f).

Let us explain the results displayed in Table IV. Looking at row (b) of column (a), we see that out of the 30 machine learning algorithms executed on Variant 1 of DSDD, the LZD data representation resulted in 19 statistically significant wins over the TFIDF data representation (the number in parenthesis indicates this), and an additional 8 wins (not statistically significant). That is, 27 out of the 30 algorithms resulted in a higher learning accuracy when trained with the LZD



```
HFF  HFH  HFG  HFF  HFF  HFH  HFG  HEF  HFH  HEF  HFF  HFH  HFG  GFF  HFG  GFF  HFF  HFH
HFG  HFF  HFG  HFF  HFG  HFH  HFF  HFH  HFG  GFF  HFG  HFH  HFF  HGH  HFG  HFG  HFF  HFH
HFG  GFF  HFH  HFG  HFF  HFF  HFH  HFG  HFF  HFH  HFG  GFF  HFF  HFG  GFF  HEH  HFG  HFF
HEF  HEH  HFG  HFF  HFH  HFG  HFF  HEF  HFH  HFG  HFH  HFG  HFF  HFF  HFH  HFG  GFF  HFH
HFG  HFF  HFF  HFH  HFG  GFF  HEH  HFG  HFF  GFF  HEH  HEG  GFF  HEH  GEG  HFF  GGF  HEH
GFG  HFF  AIJ  FHG  HFF  GFF  AIJ  GFG  HFF  GFG  FFG  GFF  FFG  GFF  GFF  GFH  GFG  FFE
GFH  GFG  GFF  GFH  GFG  FFE  GFH  GFG  FFE  GFF  GFH  FFG  FFE  FFG  FFH  FFG  FFF  FFH
FFG  FFE  FFH  FFG  FFE  FFF  FFH  FFG  FFF  FFG  FFF  FFE  FFH  FFG  FFF  FFH  FFG  GFF
FFE  FFH  FFG  GFF  FFH  FFG  FFF  FEF  FFH  FFG  GFE  FFH  FFF  FFF  FEF  FEH  FFF  FFF
FEH  HEH  FFE  FFF  HEH  FFE  FEG  FEG  FCE  GEH  GFG  GEF  GEH  GFG  GEF  FCF  GEH  HEG
GEF  HEH  GEG  GEF  GEG  GEH  GEG  GEF  GEH  GEG  GEF  FEE  GEH  GEG  GEF  GEH  GEH  GEF
GDE  GEH
GEH  GEF  GFH  GEH  GEF  GEF  GFH  GEH  GFH  GEG  GEF  GEF  GEH  GEG  GEG  GEE  GEF  GFH
GEG  GFF  GEH  GEG  GFF  GEF  GEH  GEG  GEF  GEH  GEG  GFF  GEF  GEH  GEG  GFF  GEH  GFF
GFF  GEH  GEG  GFF  GFH  FEG  GFF  FEE  GEH  EEG  GFF  EEH  FEG  GFF  FFE  FFH  FEF  FFH
FEF  GEF  FFE  FFH  EEG  GEE  FFH  EEG  GFF  FFE  FFH  FEG  GFE  FEG  FFE  FFE  FFH  EEG
FFF  FFH  EFF  FGF  EEH  EEG  EEF  EEH  EEG  EEE  FFF  EEH  EEG  EEH  EEG  EEF  EFF  EEH
EEG  EFE  EEG  EEG  EFF  EFE  EEG  DEG  EFF  EFG  EEF  EFE  EFG  DFG  EEF  DFH  DFF  DEF
DFF  DFG  DFF  DFH  DFG  DFF  DFF  DFG  DFF  DFF  DFG  DFG  DFF  DFF  DFH  DFG  DFF  DFH
DFG  DFF  DFF  DFG  DFG  DFG  DFG  DFF  DEF  DFG  DFG  DFF  DFG  DFG  DFF  EEF  DFH  DFG
DFF  DFH  DFG  DFF  EEF  DFG  DFG  DFF  DEH  DFG  DFE  EEF  DEH  DEG  DFF  DEH  EEG  DFF
EDF  DEH  EEG  DFF  DFH  EEG  DFF  EEF  DFG  EEG  EEF  DFH  EEG  EEF  DDF  EFH  EEG  EEF
EFH  EEG  EEF  EEE  EFH  EEG  EEF  EFG  EEG  EEF  DEE  EFH  EEG  EEF  EFH  EEG  FDF  EEF
EFH  FEG
```

Fig. 1. An example of two instances in Variant 1 of LDPA. Each instance is an artificial document with 200 words each of length 3.

TABLE V
LEARNING CLASSIFICATION RESULTS FOR DATA-SET LDPA

	a	b	c	d	e	f
a	-	12 (9)	24 (7)	15 (12)	27 (24)	15 (12)
b	18 (15)	-	21 (16)	27 (9)	26 (21)	27 (24)
c	6 (3)	9 (6)	-	12 (10)	27 (9)	15 (12)
d	15 (15)	3 (3)	18 (15)	-	21 (19)	27 (2)
e	3 (3)	4 (4)	3 (0)	9 (5)	-	12 (10)
f	15 (1)	3 (3)	15 (15)	3 (3)	18 (15)	-

TABLE VI
LEARNING CLASSIFICATION RESULTS FOR DATA-SET GSATM

	a	b	c	d	e	f
a	-	3 (3)	12 (3)	3 (3)	15 (3)	3 (3)
b	27 (27)	-	27 (27)	6 (1)	27 (27)	10 (1)
c	18 (0)	3 (3)	-	3 (3)	21 (2)	3 (3)
d	27 (27)	24 (2)	27 (27)	-	27 (27)	17 (0)
e	15 (0)	3 (3)	9 (0)	3 (3)	-	3 (3)
f	27 (27)	20 (4)	27 (27)	13 (0)	27 (27)	-

TABLE VII
LEARNING CLASSIFICATION RESULTS FOR DATA-SET UIWA

	a	b	c	d	e	f
a	-	0 (0)	11 (0)	0 (0)	6 (0)	3 (0)
b	15 (9)	-	15 (3)	9 (0)	15 (0)	5 (0)
c	4 (0)	0 (0)	-	0 (0)	1 (0)	0 (0)
d	15 (0)	6 (0)	15 (8)	-	15 (0)	3 (0)
e	9 (0)	0 (0)	14 (0)	0 (0)	-	0 (0)
f	12 (4)	10 (2)	15 (5)	12 (0)	15 (6)	-

data representation compared to the TFIDF representation. Looking at row (d) of column (c), we see that out of the 30 machine learning algorithms executed on Variant 2 of DSDD, the LZD data representation resulted in 18 statistically significant wins over the TFIDF data representation and an additional 10 wins (not statistically significant). That is, 28 out of the 30 algorithms resulted in a higher learning accuracy when trained with the LZD data representation compared to the TFIDF representation. Looking at row (f) of column (e) we see that out of the 30 machine learning algorithms executed on Variant 3 of DSDD, the LZD data representation resulted in 24 statistically significant wins over the TFIDF data representation and an additional 2 wins (not statistically significant). That is, 26 out of the 30 algorithms resulted in a higher learning accuracy when trained with the LZD data representation compared to the TFIDF representation. So in summary, the LZD data representation is much better than the TFIDF representation as it improves the learning accuracy for almost all 30 algorithms.

We see the same success for the LZD representation in Tables V, VI as almost all of the 30 algorithms consistently learn better based on the LZD data representation than when compared to the TFIDF representation.

On the learning problem defined by the UIWA data set, we observe that the LZD data representation improved only the 15 algorithms (of the k-NN type) relative to the TFIDF. The results are displayed in Table VII. As can be seen from column (a) and row (b), column (c) and row (d), column (e), and row (f), the LZD data representation always outperforms the TFIDF representation.

VIII. CONCLUSIONS

We introduce Algorithm LZD for computing in parallel a string distance based on the LZ-complexity of a pair of arbitrarily long finite strings, limited only by the amount of global memory on a GPU and the number of GPUs. One of the main advantages of this distance is that it can measure dissimilarity between a pair of strings of different lengths, which is useful for machine learning from unstructured data. We introduce a data representation based on Algorithm LZD, which transforms featureless data into a distance-matrix representation that can be learned by standard machine learning algorithms. This LZD-matrix representation of the data is easily and automatically computed from the original unstructured data and hence eliminates the need for pre-defining or extracting machine learning attributes. The speed-up factor for computing an NxN distance matrix for N instances reaches 528 for $N = 16$ and strings of length 1M bytes.

As an example of unstructured featureless data, we run Algorithm LZD on several time-series data sets, including data from electric drive diagnostics for defective components, data sets with measurements of sensors on a human body (used to classify a person's activity), and a data set of gas sensor measurements for the detection of chemicals. The LZD data representation yields significantly higher machine-learning classification accuracies than when the data is represented by an alternative ubiquitous data representation based on TFIDF vectors. Therefore, we conclude that the proposed parallel algorithm can be useful to efficiently learn from unstructured data.

As regards future research, possible directions include learning from unstructured binary data, e.g., audio, video files, and other character-based data, evaluating the LZD-data representation for unsupervised learning, e.g., clustering from unstructured data, and comparing it to other methods of learning from unstructured data. A possible extension of Algorithm 3 is to reuse the result of the first stage in the second stage. With some added algorithm complexity and additional memory allocation, one could store the components (and not just the number of components) of the LZ-complexity of a string x in the first stage and use this as a starting point for computing the LZ-complexity of its concatenation with another string, namely, xy, where y is any of the other $N-1$ strings. We expect this to give a significant computational speedup in the second stage.

Acknowledgments

This work was supported by the Ministry of Science & Technology, ISRAEL.

References

[1] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Addison-Wesley, 1999.

[2] J. Ziv and A. Lempel, "On the complexity of finite sequences," *IEEE Transactions on Information Theory*, vol. 22, no. 3, pp. 75–81, 1976.

[3] ——, "A universal algorithm for sequential data compression," *IEEE Transactions on Information Theory*, vol. 23, no. 3, pp. 337–343, 1977.

[4] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Transactions on Information Theory*, vol. 24, no. 5, pp. 530–536, 1978.

[5] K. Sayood and H. H. Otu, "A new sequence distance measure for phylogenetic tree construction," *Bioinformatics*, vol. 19, no. 16, pp. 2122–2130, 2003.

[6] U. Chester and J. Ratsaby, "Universal distance measure for images," in *Proceedings of the 27th IEEE Convention of Electrical Electronics Engineers in Israel (IEEEI'12)*, Eilat, Israel, November 14-17, 2012, pp. 1–4.

[7] ——, "Machine learning for image classification and clustering using a universal distance measure," in *Proceedings of the 6th International Conference on Similarity Search and Applications (SISAP'13)*, ser. Springer Lecture Notes in Computer Science, N. Brisaboa, O. Pedreira, and P. Zezula, Eds., vol. 8199, 2013, pp. 59–72.

[8] A. Belousov and J. Ratsaby, "Massively parallel computations of the LZ-complexity of strings,," in *Proc. of the 28th IEEE Convention of Electrical and Electronics Engineers in Israel (IEEEI'14)*, Eilat, Dec. 3-5 2014, pp. pp. 1–5.

[9] J. Ratsaby and A. Timashkov, "Accelerating the LZ-complexity algorithm," in *Proc. of IEEE 29th International Conference on Parallel and Distributed Systems (ICPADS)*, 2023, pp. 1–8.

[10] K. Bhargavi and S. B. Babu, "Accelerating the big data analytics by GPU-based machine learning: A survey," in *Proceedings of International Symposium on Sensor Networks, Systems and Security*, N. S. Rao, R. R. Brooks, and C. Q. Wu, Eds. Cham: Springer International Publishing, 2018, pp. 63–83.

[11] R. P. W. Duin, E. Pekalska, and M. Loog, "Non-euclidean dissimilarities: Causes, embedding and informativeness," in *Similarity-Based Pattern Analysis and Recognition. Advances in Computer Vision and Pattern Recognition*, M. Pelillo, Ed. London: Springer, 2013.

[12] A. Belousov and J. Ratsaby, "A parallel distributed processing algorithm for image feature extraction," in *Advances in Intelligent Data Analysis XIV - 14th International Symposium, IDA 2015, Saint-Etienne, France, October 22 - 24, 2015. Proceedings*, ser. Lecture Notes in Computer Science, vol. 9385. Springer, 2015.

[13] ——, "A parallel computation algorithm for image feature extraction," *Journal of Advances in Applied and Computational Mathematics*, vol. 6, pp. 1–18, 2019.

[14] H. A. Dau, A. J. Bagnall, K. Kamgar, C. M. Yeh, Y. Zhu, S. Gharghabi, C. A. Ratanamahatana, and E. J. Keogh, "The UCR time series archive," *CoRR*, vol. abs/1810.07758, 2018. [Online]. Available: http://arxiv.org/abs/1810.07758

[15] D. B. Kirk and W. W. Hwu, *Programming Massively Parallel Processors: A Hands-on Approach*, 3rd ed. San Francisco, CA, USA: Elsevier, 2017.

[16] UCI, "Uci machine learning repository," https://archive.ics.uci.edu/ml/index.php.

[17] E. Frank, M. A. Hall, and I. Witten, *The WEKA Workbench, Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques"*, 4th ed. Morgan Kaufmann, 2016.

[18] I. H. Witten, E. Frank, and M. A. Hall, *Data Mining: Practical Machine Learning Tools and Techniques*, 4th ed. San Francisco, CA, USA: Elsevier, 2017.

Appendix

For reproducibility, all the datasets used in Section VII are publicly available at https://archive.ics.uci.edu/ in the following folders:

Dataset	Path
DSDD	/dataset/325/dataset+for+sensorless+drive+diagnosis
LDPA	/dataset/196/localization+data+for+person+activity
UIWA	/dataset/286/user+identification+from+walking+activity
GSATM	/dataset/487/gas+sensor+array+temperature+modulation

We used WEKA for all our machine learning experiments, publicly available at https://waikato.github.io/weka-wiki/downloading_weka/. The ML algorithms mentioned in Section VII were run using the default hyper-parameters in WEKA. The preprocessing stage that converts a time series into artificial documents is easy to implement. The code that implements Algorithm 3, together with the library that implements the parallel LZ-complexity algorithm, will be made available upon request.