Compression and Data Mining

Dan A. Simovici, Ping Chen, Tong Wang, and Dan Pletea
Univ. of Massachusetts Boston, Boston, USA
Email: {dsim, dpletea}@cs.umb.edu; Ping.Chen@umb.edu; tongwang0001@gmail.com

Abstract —Data compression plays an important role in data mining in assessing the minability of data and a modality of evaluating similarities between complex objects. We discuss various mining applications ranging from compressibility of strings of symbols and of languages, graph compressibility, compression of market basket data. Also, we examine the role of compression in computing similarity in text corpora and we propose a novel approach for assessing the quality of text summarization.

Index Terms—Compression ratio, Thue-Morse sequence, lossless compression, stemming, lemmatizing

I. Introduction

We intend to show that data compression is an essential tool for data mining that can be used both to assess data minability and also, as a mining tool itself. Indeed, as observed by Heikki Mannila [1], data mining itself can be regarded as a form of data compression since the goal of the data mining is to "compress data by finding some structure in it".

Two broad classes of compression algorithms exist: *lossy compression*, that reduces significantly data but does not allow the full inverse transformation, from compressed data to the original data, and *lossless compression*, that achieves data reduction and can be completely reversed. In this paper we focus on using lossless compression in data mining.

Compression can be used as a tool to evaluate the potential of a data set of producing interesting results in a data mining process. The basic idea that data that contains patterns that occur with a certain regularity will be compressed more efficiently compared to data that has no such characteristics. Thus, a pre-processing phase of the mining process should allow to decide whether a data set is worth mining, or compare the interestingness of applying mining algorithms to several data sets.

Since compression is generally inexpensive (and certainly less expensive than mining algorithms), and compression methods are well-studied and understood, pre-mining using compression will help data mining analysts to focus their efforts on mining resources that can provide a highest payout without an exorbitant cost.

The role of compression developing parameter-free data mining algorithms in anomaly detection, classification and clustering was examined in [2]. Further advances in this direction were developed in [3], [4] and [5]. A Kolmogorov complexity-based dissimilarity was successfully used to texture matching problems in [6] which have a broad spectrum of applications in areas like bioinformatics, natural languages, and music. Compression algorithms are used in the actual mining process to handle data mining explorations that return huge sets of results by extracting those results that actually are representative of the data set (see, for example [7], [8]).

Our goal is to show that compression can be used for assessing the interestingness of applying an actual data mining process, and to use compression as a tool for evaluating similarity between complex objects.

The lossless compression algorithm mostly used in this paper is the LZW (Lempel-Ziv-Welch) algorithm, introduced in 1984 by T. Welch in [9]. The algorithm does not need to check all the data before starting the compression and the performance is based on the number of the repetitions and the lengths of the strings and the ratio of 0s/1s or true/false at the bit level. There are several versions of the LZW algorithm. Popular programs (such as Winzip or the zip function of MATLAB) use variations of the LZW compression. These algorithms work both at the bit level and at the character level.

After examining compressibility of binary strings in Section II we explore several experimental settings that provide strong empirical evidence of the correlation between compression ratio and the existence of hidden patterns in data. Section III discusses the compressibility of sequences of symbols produced by various generative mechanisms. Section IV is dedicated to using compression in text mining. Finally, in Section V, we examine the compressibility of files that contain market basket data sets. This paper is an extension of our contribution [10].

II. Patterns in Strings and Compression

An *alphabet* is a finite and non-empty set whose elements are referred to as *symbols*. Let A^* be the set of sequences on the alphabet *A*. We refer to these sequences as *words* or *strings*. The length of a string w is denoted by $|w|$. The null string on *A* is denoted by λ and we define A^+ as $A^+ = A^* - \{\lambda\}$. The subsets of A_ are referred to as languages over *A*.

If $w \in A^*$ can be written as $w = utv$, where $u, v \in A^*$ and t $t \in A^+$, we say that the pair (t, m) is an *occurrence* of *t* in w, where m is the length of u.

Manuscript received April 10, 2015; revised Septermber 15, 2015.
doi:10.12720/jcm.10.9.677-684

The number of occurrences of a string x in a string w is denoted by $n_x(w)$. Clearly, we have $\sum\{n_a(w) \mid a \in A\} = |w|$ for any symbol $a \in A$. The *prevalence* of x in w is the number $f_x(w) = \frac{n_x(w)\cdot|x|}{|w|}$ which gives the ratio of the characters contained in the occurrences of t relative to the total number of characters in the string.

The result of applying a compression algorithm *C* to a string $w \in A^*$ is denoted by *C*(*w*) and the *compression ratio* is the number

$$\mathsf{CR}_C(w) = \frac{|C(w)|}{|w|}$$

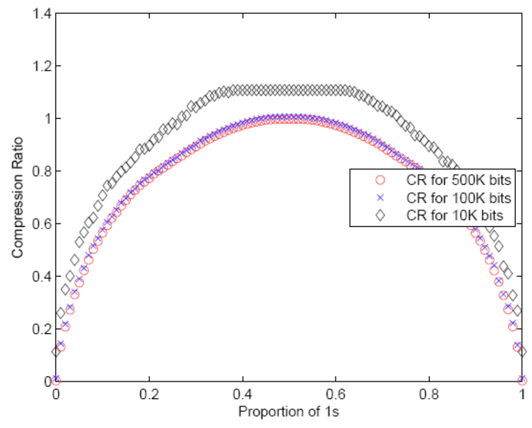


Fig. 1. Baseline CR_{jZIP} behavior

We shall use the binary alphabet $B = \{0,1\}$ and the LZW algorithm, the compression algorithm of the package java.util.zip, or the zip function of MATLAB.

We generated random strings of bits (0s and 1s) and computed the compression ratio for strings with a variety of symbol distributions. A string w that contains only 0s (or only 1s) achieves a very good compression ratio of $\mathsf{CR}_{jZIP}(w) = 0.012$ for 100*K* bits and $\mathsf{CR}_{jZIP} = 0.003$ for 500*K* bits, where $jZIP$ denotes the compression algorithm from the package java.util.zip. Fig. 1 shows, as expected, that the worst compression ratio is achieved when 0s and 1s occur with equal frequencies.

For strings of small length (less than 10^4 bits) the compression ratio may exceed 1 because of the overhead introduced by the algorithm. However, when the size of the random string exceeds 10^6 bits this phenomenon disappears and the compression ratio depends only on the prevalence of the bits and is relatively independent on the size of the file. Thus, in Fig. 1, the curves that correspond to files of size 100K bits and 500*K* bits overlap. We refer to the compression ratio of a random string w that contains $n_0(w)$ zeros and $n_1(w)$ ones as the *baseline compression ratio* for this distribution of bits.

We created a series of binary strings $\varphi t,m$ which have a minimum guaranteed number m of occurrences of patterns $t \in \{0,1\}^k$, where $0 \leqslant m \leqslant 100$. The compression baselines for files containing the patterns 01, 001,0010, and 00010 are shown in Table I.

TABLE I: BASELINE COMPRESSION RATIO FOR FILES CONTAINING A MINIMUM GUARANTEED NUMBER OF PATTERNS

Pattern	Proportion of 1s	Baseline
01	50%	1.007
001	33%	0.934
0010	25%	0.844
00010	20%	0.779

Specifically, we created 101 files $\varphi 001,m$ for the pattern 001, each containing 100K bits and we generated similar series for $t \in \{01, 0010, 00010\}$. In the case of the 001 pattern the baseline is established at 0.934, and after the prevalence exceeds 20% the compression ratio drops dramatically. Results of the experiment for 001 are shown in Table II. In Fig. 2 we show that similar results hold for all patterns mentioned above.

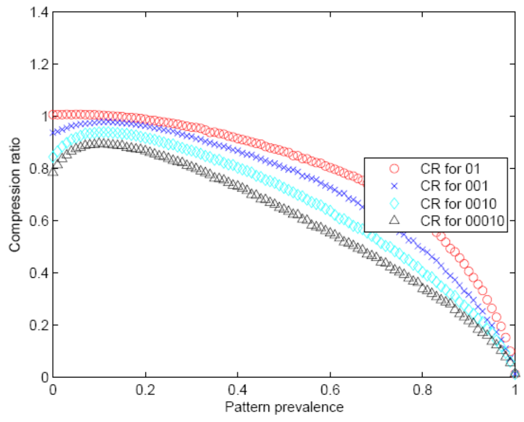


Fig. 2. Dependency of compression ratio on pattern prevalence

TABLE II: PATTERN '001' PREVALENCE VERSUS THE COMPRESSION RATIO CR_{jZIP}

Prevalence of '001' pattern	CR_{jZIP}
0%	0.93
10%	0.97
20%	0.96
30%	0.92
40%	0.86
50%	0.80
60%	0.72
70%	0.62
80%	0.48
90%	0.31
95%	0.19
100%	0.01

III. COMPRESSIBILITY OF LANGUAGES AND SEQUENCES

Sequences or sets of sequences of symbols are often subjected to data mining processes and identifying those sequences that contain interesting patterns before the actual mining process may be computationally significant.

We begin by examining the well-known sequence called the Thue-Morse sequence [11] that has many applications ranging from crystal physics [12], counter synchronization [13], metrology [14], [15], and chess playing [16], as well as in game theory, fractals and turtle graphics, chaotic dynamical systems, etc.

This sequence contains patterns but not repetitions.

Definition 3.1: Let $n \in \mathbb{N}$ be a natural number. The *Thue-Morse sequence* $\mathbf{s}_n = s_0 s_1 \cdots s_n$ is a word over the alphabet $\{0,1\}$ defined as:

$$s_i = \begin{cases} 1 & \text{if } i \text{ has an odd number of 1s} \\ & \text{in its binary representation} \\ 0 & \text{otherwise,} \end{cases}$$

for $0 \leqslant i \leqslant n$.

TABLE III: THE COMPRESSION RATIO $\mathrm{CR}_{jZIP}(\mathbf{s}_{2^k})$ FOR THUE-MORSE SEQUENCES

k	$\lvert seq_{2^k} \rvert$	$\mathrm{CR}_{jZIP}(seq_{2^k})$
5	32	34
8	256	4.625
10	1024	1.226
12	4096	0.328
14	16384	0.0932
15	32768	0.0542
16	65536	0.0322
17	131072	0.0208
18	262144	0.0151
19	524288	0.012
20	1048576	0.010
21	2097152	0.010
22	4194304	0.009

For example, we have

$$\mathbf{s}_{16} = (0,1,1,0,1,0,0,1,1,0,0,1,0,1,1,0)$$

It is clear that if $m, n \in \mathbb{N}$ and $m \leqslant n$, $\mathbf{s}_m$ is a prefix of $\mathbf{s}_n$. Thus, the successive Thue-Morse sequences define an infinite sequence.

An equivalent method for defining the Thue-Morse sequence is by starting with 0 and concatenating the complement of the sequence obtained so far. This procedure yields 0, then 01, 0110, 01101001, and so on. It is known (see [17], for example) that the Thue-Morse sequence is a cube-free sequence, that is, the sequence does not contain substrings of the form www.

We generated the Thue-Morse sequences and stored this sequence of 0s and 1s at the bit level. By using the zip compression utility from the java.util.zip package the compression ratios shown in Table III were obtained.

For small values of *k*, the sequence is incompressible due to the overhead produced by the compression process. As Table III and Fig. 3 show, for *k* big enough ($2^k \geqslant 2000$) the sequence becomes compressible and the compression ratio reaches a low value (of less than 1%) for Thue-Morse sequences longer than 4; 000; 000 characters. Since the Thue-Morse sequence $\mathbf{s}_{2^k}$ has equal number of 0s and 1s for any value of k and its compression ratio is well below the baseline compression ratio established for sequences of bits in Section II, we can conclude that even in the absence of repetitions, compression can be used for the detections of patterns.

In a series of experiments involving generative grammars we examined the compressibility of language fragments generated by these grammars. A generative grammar, or in short, a *grammar* is defined as a 4-tuple $G = (A_N, A_T, S, P)$, where A_N and A_T are non-empty, finite and disjoint sets referred to as the non-terminal and the terminal alphabet, respectively, $S \in A_N$ is the *initial symbol of the grammar G*, and *P* is a finite set of pairs of the form (α, β), where $\alpha \in (A_N \cup A_T)^+$ and $\beta \in (A_N \cup A_T)^*$. A pair $(\alpha, \beta) \in P$ is a production of the grammar *G*. Productions are used for rewriting words over $A_N \cup A_T$. Namely, if $\gamma, \delta \in (A_N \cup A_T)^*$, $\gamma = \gamma_1 \alpha \gamma_2$, and $\delta = \gamma_1 \beta \gamma_2$ for some production $(\alpha, \beta) \in P$, we write $\gamma \underset{G}{\Rightarrow} \delta$. The reflexive and transitive closure of the binary relation $\underset{G}{\Rightarrow}$ is denoted by "$\overset{*}{\underset{G}{\Rightarrow}}$". The language generated by *G* is the set $L(G) = \{x \in A_T^* \mid S \overset{*}{\underset{G}{\Rightarrow}} \}$.

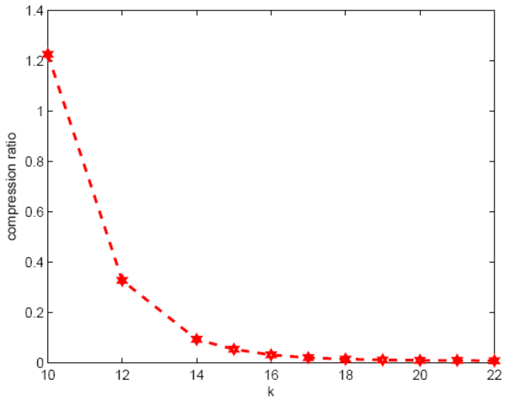


Fig. 3. Compression ratio behavior of thue-morse sequence

Grammars are used as generative devices that produce languages over their terminal alphabet. Chomsky's hierarchy (see [18] or [19]) defines four classes of grammars based on the complexity of their productions. In turn, these classes of grammars, define a strict hierarchy of classes of languages $\mathcal{L}_3 \subset \mathcal{L}_2 \subset \mathcal{L}_1 \subset \mathcal{L}_0$, where $\mathcal{L}_3$ is the class of regular languages, $\mathcal{L}_2$ is the class of context-free languages, $\mathcal{L}_1$ is the class of context-sensitive languages, and $\mathcal{L}_0$ is the class of recursively enumerable languages. It is worth noting that the classes $\mathcal{L}_3$ and $\mathcal{L}_2$ collapse on languages over one-symbol alphabet. In other words, if *L* is a language over an one-symbol alphabet, then $L \in \mathcal{L}_2$ implies $L \in \mathcal{L}_3$.

We evaluate the compressibility of a language *L* over an alphabet *A* by considering the increasing sequence of finite languages $\mathcal{S}(L) = (L_0, L_1, \ldots, L_n, \ldots)$, where L_n consists of the first *n* words of *L* in lexicographic order, computing the compression ratios $CR_{jZIP}(L_n)$, and examining the dependency of this ratio on *n*.

We examine comparatively the compressibility of the languages $L_1 = \{ww \mid w \in \{0,1\}^*\}$ (a context-sensitive language) versus the compressibility of a similar language $L_2 = \{ww^R \mid w \in \{0,1\}^*\}$ (a context-free language) which has a simpler structure. Here, the word w^R is the *reversal* of the word w and is defined as $\lambda^R = \lambda$ and $(a_{i_1} \cdots a_{i_n})^R = a_{i_n} \cdots a_{i_1}$.

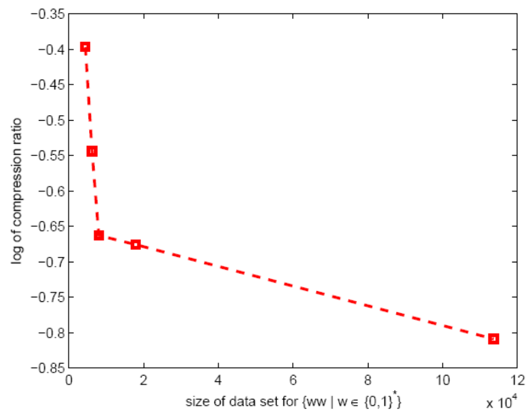


Fig. 4. Compression ratio behavior of the language L_1

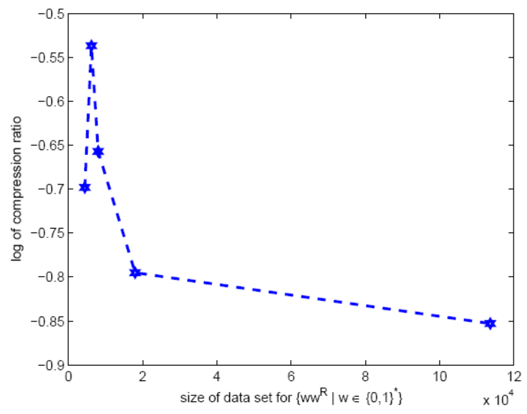


Fig. 5. Compression ratio behavior of the language L_2

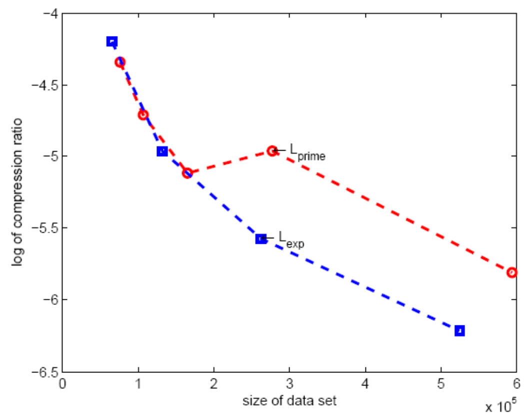


Fig. 6. Compression ratios of languages L_{exp} and L_{prime}

The results presented in Fig. 4 and Fig. 5 show that L2, the less complex language has a better (lower) compression ratio, and therefore, higher compressibility.

Similar results are obtained when comparing the compressibility of the context-sensitive languages L_{exp} and L_{prime} over the one-symbol alphabet $\{a\}$ defined by

$$\begin{aligned} L_{exp} &= \{a^{2^n} \mid n \in \mathbb{N}\}, \\ L_{prime} &= \{a^p \mid p \text{ is a prime number}\} \end{aligned}$$

The reference [18] (see Chapter 1, section 2) contains specific grammars developed for both languages. Namely, the grammar for L_{exp} has 6 productions, while the second grammar that generates L_{prime} has 42 productions. As expected, experiments summarized in Fig. 6 show that the L_{exp} is more compressible than L_{prime} which has a rather complex generating process.

These results suggest that the compressibility of languages is related to the complexity of the generative process that produce them. This will be the object of further investigations.

IV. Compression and Text-Mining

Compression offers a simple but efficient tool for text mining, in general [20]-[24] and, in particular, for biology and medicine [25] by suggesting a simple document dissimilarity computation and, also, a tool for evaluating the quality of summarization efforts.

We began with a series of experiments intended to show that compression is useful in document classification. We used java.util.zip package to compress text files.

For a document *x* let *C*(*x*) be the size of the compressed document. This allows us to define a dissimilarity between documents that belong to a corpus $\mathcal{K}$ as

$$d(x,y) = \frac{C(xy)}{C(x)+C(y)}$$

for $x, y \in \mathcal{K}$. Note that $d(x,x) = \frac{C(xx)}{2C(x)}$ is at least $\frac{1}{2}$ because any compression algorithm will take advantage of the fact that the two halves of *xx* are identical. Further, the more similar the documents are the closer the value of *d*(*x*, *y*) is to 0.5.

Given two corpora $\mathcal{K}$ and $\mathcal{K}'$ such that $|\mathcal{K}| = |\mathcal{K}'| = N$ we will define the similarity matrix between $\mathcal{K}$ and $\mathcal{K}'$ as

$$d(\mathcal{K},\mathcal{K}')_{ij} = d(x_i, y_j)$$

for $1 \leqslant i, j \leqslant N$.

All documents used in experiments were downloaded from *PubMed* and we retained only the title and abstract of each document. There are totally four groups of documents and each group has 15 documents. Two groups (ML1 and ML2) are on machine learning topics; the topic of the remaining two groups (ADHD1 and ADHD2) is the attention deficit hyperactivity disorder.

Various forms of preprocessing of the documents are considered; in each case we computed the dissimilarity matrices of various corpora.

Table IV shows the result of the first four experiments and contains the number of pairs that occur in the intervals specified in the header of the table. For example, in the first row of experiment 1, 42 means there are 42 values in the compression distance matrix *d*(ADHD1;ADHD2) in the interval [0.85, 0.865). The cumulative counts are shown in parentheses. cumulative total from the beginning interval.

TABLE IV: FIRST FOUR EXPERIMENTS. IN THE FIRST EXPERIMENT WE USED THE ORIGINAL DOCUMENT; IN THE SECOND, WE REMOVED THE STOP WORDS ON A SHORT LIST; IN THE THIRD EXPERIMENT, STOP WORDS FROM A LONG LIST WERE REMOVED; FINALLY, IN THE FOURTH EXPERIMENT WE REMOVED STOP WORDS FROM ALONG LIST, NUMBERS, AND PUNCTUATION.

Exp.	Matrix	Mean	$[0.5, 0.8)$	$[0.8, 0.85)$	$[0.85, 0.865)$	$[0.865, 0.88)$	$[0.88, 0.9]$	$\geqslant 0.9$
1	d(ADHD1, ADHD2)	0.87484	0	25	42(67)	69(136)	67	22
	d(ADHD2, ML1)	0.89179	0	0	7(7)	42(49)	109	67
	d(ADHD2, ML2)	0.89067	0	0	9(9)	48(57)	104	64
	d(ML1, ML2)	0.88615	0	0	6(6)	56(62)	135	28
2	d(ADHD1, ADHD2)	0.86382	0	52	57(109)	67(176)	35	14
	d(ADHD2, ML1)	0.88192	0	5	36(41)	63(104)	80	41
	d(ADHD2, ML2)	0.88072	0	6	35(41)	73(114)	79	32
	d(ML1, ML2)	0.87591	0	4	38(42)	95(137)	80	8
3	d(ADHD1, ADHD2)	0.85869	3	68	67(138)	50(188)	25	12
	d(ADHD2, ML1)	0.87743	0	13	51(64)	63(127)	65	33
	d(ADHD2, ML2)	0.87588	0	19	44(63)	66(129)	69	27
	d(ML1, ML2)	0.87025	0	16	57(73)	100(173)	49	3
4	d(ADHD1, ADHD2)	0.84192	10	137	42(189)	20(209)	13	3
	d(ADHD2, ML1)	0.86408	0	54	69(123)	54(177)	35	13
	d(ADHD2, ML2)	0.86146	0	66	69(135)	56(191)	23	11
	d(ML1, ML2)	0.85632	0	65	104(169)	43(212)	13	0

TABLE V: THREE EXPERIMENTS WITH DOCUMENTS THAT ONLY CONTAIN LETTERS. IN THE FIFTH EXPERIMENT EACH DOCUMENT WAS SORTED; IN THE SIXTH EXPERIMENTS WORDS WERE LEMMATIZED AND THEN SORTED; IN THE SEVENTH EXPERIMENTS WORDS WERE STEMMED AND THEN SORTED.

Exp.	Matrix	Mean	$[0.7, 0.75)$	$[0.75, 0.7750$	$[0.775, 0.8)$	$[0.8, 0.825)$	$[0.825, 0.85)$	≥ 0.85
5	d(ADHD1, ADHD2)	0.80363	4	26	65(95)	89(184)	28(212)	13
	d(ADHD2, ML1)	0.83627	0	0	2(2)	60(62)	108(170)	55
	d(ADHD2, ML2)	0.83383	0	0	4(4)	68(72)	115(187)	38
	d(ML1, ML2)	0.82338	0	0	12(12)	116(128)	80(208)	17
6	d(ADHD1, ADHD2)	0.79820	6	34	78(118)	74(192)	24(216)	9
	d(ADHD2, ML1)	0.83187	0	0	5(5)	77(82)	105(187)	38
	d(ADHD2, ML2)	0.82904	0	0	5(5)	92(97)	98(195)	30
	d(ML1, ML2)	0.81844	0	0	21(21)	137(158)	55(213)	12
7	d(ADHD1, ADHD2)	0.78085	27	63	87(177)	35(212)	13(225)	0
	d(ADHD2, ML1)	0.81463	0	0	60(60)	97(157)	59(216)	9
	d(ADHD2, ML2)	0.81169	0	4	63(67)	102(169)	46(215)	10
	d(ML1, ML2)	0.79933	0	10	112(122)	88(210)	15(225)	0

The mean values of *d*(*ADHD2*, *ML1*) and *d*(*ADHD2*, *ML2*) are larger than *d*(*ADHD1*, *ADHD2*) and *d*(*ML1*, *ML2*) in all four experiments, but the difference is not large.

In the first experiment the original documents were used to compute the distance matrix. After removing a short list of stop words (in the second experiment) and removing a long list stop words (in the third experiment), mean values decrease comparing to previous experiments. The difference between mean values of d(ADHD2, ML2) and d(ML1, ML2) becomes 0.00481 and 0.00563, which are a little better. If looking at values less than 0.88, there are more values in d(ADHD1, ADHD2) and d(ML1, ML2) less than 0.88 comparing to d(ADHD2, ML1) and d(ADHD2, ML2).

In the fourth experiment, all non-alphabetic characters were removed. The mean values decrease in all four matrix. The difference between mean value of *d*(ADHD2, ML2) and *d*(ML1, ML2) becomes 0.00514.

The next three experiments make use of the documents which only contain letters (obtained for the previous fourth experiment).

In experiments 5-7, words in each document were sorted in alphabetic order. The distance values decrease a lot comparing to the first 4 experiments. And if choosing 0.825 as threshold, we can find out there are more values in *d*(ADHD1, ADHD2) and *d*(ML1, ML2) less than 0.825.

In the fifth experiment, if still taking the difference between mean value of *d*(ADHD2, ML2) and *d*(ML1, ML2) as example, we get 0.01045, it is much better than the first four experiments.

In the sixth experiment words were lemmatized before being sorted. Lemmatization is a more complex approach to stemming that involves first determining the part of speech of a word, and applying different stemming rules for each part of speech. Plural noun will lemmatize to singular, for example, 'cats' would be 'cat'. The vocabulary for the 60 documents that only contain letters is 2398 words. After lemmatizing, the vocabulary reduces to 2172. The difference between mean values of *d*(ADHD2, ML2) and *d*(ML1, ML2) show a small increase.

In the seventh experiment stemming was used instead of lemmatizing. Stemming is similar to lemmatizing, to keep the basic form of a word. The difference is it chops the affix and only keeps the stem. For example, 'include', 'including', 'included' will be stemming to 'includ'. The vocabulary reduces to 1733 after stemming all words. The difference between mean values of *d*(ADHD2, ML2) and *d*(ML1, ML2) increases to 0.01266.

Cosine similarity is widely used for measuring similarity between two vectors. It can also be applied to compute the similarity between two document vectors. In traditional Vector Space Model, a document can be represented by a *k*-dimensional vector, where *k* is the size of the vocabulary in corpus. Cosine similarity is defined as

$$\cos(x, y) = \frac{x \cdot y}{\|x\| \|y\|}$$

In Table VI we show the cosine similarity using the documents from the sixth experiment, the documents with sorting and lemmatizing words, and the seventh experiment, the documents with sorting and stemming words.

To examine the relationship between cosine similarity matrix and the compression dissimilarity matrix we used the matrices from the sixth and the eight experiments, considering each column in the matrix as a vector. We show the correlations between the 15 columns of these tables in Table VII.

TABLE VII: CORRELATION BETWEEN COMPRESSION DISSIMILARITY AND COSINE SIMILARITY

col	d(ADHD1, ADHD2)	d(ADHD2, ML1)	d(ADHD2, ML2)	d(ML1, ML2)
1	0.1536	-0.2286	-0.8173	-0.4203
2	-0.3563	-0.1225	-0.6185	-0.4121
3	-0.3953	-0.3623	-0.5938	-0.5824
4	-0.0660	0.1501	-0.4626	-0.7206
5	0.0027	-0.1639	-0.5309	-0.4552
6	-0.2258	-0.3250	-0.5930	-0.4173
7	-0.3770	-0.1591	0.0551	-0.0091
8	0.2077	-0.3292	-0.6123	-0.5253
8	-0.1767	0.0785	-0.3939	0.0649
10	0.0919	-0.2045	-0.5225	-0.6969
11	-0.6875	0.3656	-0.7442	-0.4740
12	0.0094	-0.3317	-0.6206	-0.7170
13	-0.1778	-0.4335	-0.2593	-0.4234
14	-0.1147	0.0009	-0.6306	-0.7185
15	-0.5423	-0.1628	-0.2461	-0.5295

As more preprocessing is applied, the mean value of the compression dissimilarity decreases, and the difference be tween mean values of *d*(ADHD2, ML1), *d*(ADHD2, ML2), *d*(ADHD1, ADHD2), and *d*(ML1, ML2) keeps growing.

Various preprocessing techniques such as removing stop words, removing non-letter characters, sorting words in alphabetic order, lemmatizing, and stemming improve performance.

In every one of the seven experiments about compression, *d*(ADHD1, ADHD2) is less than *d*(ML1, ML2). One reason might be machine learning is a more general topic than ADHD and therefore, the documents are more diverse.

Most of the correlation between compression and cosine are negative, as the similarity of documents increases as compression distance value decreases. Compression does not outperform cosine similarity. However, it is inexpensive and easy to implement comparing to cosine similarity and it saves a lot of work required in computing in cosine similarity like building vocabulary, creating document vector, computing dot product with vectors (document vector usually has a high dimension). Thus, clustering documents based on compression dissimilarity can serve as a useful preprocessing tool in document clustering.

Compression also allows the evaluation of the faithfulness of abstract construction for paper corpora. The idea is that a good abstraction process for a corpus of papers yields abstracts whose dissimilarity has a high degree of correlation with the dissimilarity between the papers themselves.

Two more corpura (ADHD3 and ML3) were downloaded from *PubMed* and we retained the full text of each document. Then, each document was split into two parts: abstract and text resulting in two groups of abstracts and two groups of full text documents: ADHD3(a), ML3(a), and ADHD3(d), ML3(d), respectively. Figures, tables, and sections like acknowledgement, references, supplements were removed from the full text corpora ADHD3(d) and ML3(d).

The resulting corpora were preprocessed by removing stop words, non-letter characters, lemmatizing and sorting words in alphabetic order. The mean values of resulting dissimilarity matrices are shown in Table VIII.

TABLE VIII: THE MEANS FOR MATRICES FOR ADHD3(A), ADHD3(D), ML3(A), ML3(D)

Exp.	Matrix	Mean
10	d(ADHD3(a), ADHD3(a))	0.77875
	d(ADHD3(d), ADHD3(d))	0.79342
	d(ML3(a), ML3(a))	0.78910
	d(ML3(d), ML3(d))	0.79301

We estimate that the summarization processes for the articles nin corpora ADHD3 and ML3 are of good quality since the correlations between the elements of the distance matrices of the abstracts and papers are close to 1 (as shown in Table IX).

TABLE IX: CORRELATION BETWEEN ABSTRACTS AND TEXTS IN ADHD3 AND ML3

Exp.	Abstract Matrices	Text Matrices	Correlation
11	d(ADHD3(a), ADHD3(a))	d(ADHD3(d), ADHD3(d))	0.9275
	d(ML3(a), ML3(a))	d(ML3(d), ML3(d))	0.9664

V. FREQUENT ITEMS SETS AND COMPRESSION RATIO

A market basket data set consists of a multiset $\mathfrak{T}$ of transactions. Each transaction *T* is a subset of a set of items $I = \{i_1, \ldots, i_N\}$. The multiplicity of a transaction *T* in the multiset $\mathfrak{T}$ is denoted by *m*(*T*).

A transaction is described by its characteristic *N*-tuple *t* = ($t_1, \ldots, t_N$), where

$$t_k = \begin{cases} 1 & \text{if } i_k \in T. \\ 0 & \text{otherwise.} \end{cases}$$

for $1 \leqslant k \leqslant N$. The length of a transaction *T* is $|T| = \sum_{k=1}^{N} t_k$, while the average size of transactions is $\frac{\sum\{|T| \mid T \text{ in } \mathfrak{T}\}}{|\mathfrak{T}|}$.

The support of a set of items *K* of the data set $\mathfrak{T}$ is the number

$$\mathsf{supp}(K) = \frac{|\{T \in \mathfrak{T} \mid K \subseteq T\}|}{|\mathfrak{T}|}$$

The set of items *K* is s-frequent if supp(*K*) > *s*.

The study of market basket data sets is concerned with the identification of association rules. A pair of item sets (X, Y) is an association rule, denoted by $X \rightarrow Y$. Its support, $\mathrm{supp}(X \rightarrow Y)$ equals $\mathrm{supp}(X)$ and its confidence $\mathrm{conf}(X \rightarrow Y)$ is defined as

$$\mathrm{conf}(X \rightarrow Y) = \frac{\mathrm{supp}(X \cup Y)}{\mathrm{supp}(X)}$$

Using the artificial transaction ARMiner generator described in [26], we created a basket data set.

Transactions are represented by sequences of bits ($t_1, \ldots, t_N$). The multiset $\mathcal{T}$ of M transactions was represented as a binary string of length MN obtained by concatenating the strings that represent transactions.

We generated files with 1000 transactions, with 100 items available in the basket, adding up to 100K bits.

For data sets having the same number of items and transactions, the efficiency of the compression increases when the number of patterns is lower (causing more repetitions). In an experiment with an average size of a frequent item set equal to 10, the average size of a transaction equal to 15, and the number of frequent item sets varying in the set {5, 10, 20, 30, 50, 75, 100, 200, 500, 1000,} the compression ratio had a significant variation ranging between 0.20 and 0.75, as shown in Table X. The correlation between the number of patterns and the compression ratio was 0.544. Although the frequency of 1s and baseline compression ratio were roughly constant (at 0.75), the number of patterns and compression ratio were correlated.

TABLE X: NUMBER OF ASSOCIATION RULES AT 0.05 SUPPORT LEVEL AND 0.9 CONFIDENCE

Number of Patterns	Frequency of 1s	Baseline compression	Compr. ratio	Number of rules
5	16%	0.75	0.20	9,128,841
10	17%	0.73	0.34	4,539,650
20	17%	0.73	0.52	2,233,049
30	17%	0.76	0.58	106,378
50	19%	0.75	0.65	2,910,071
75	18%	0.75	0.67	289,987
100	18%	0.75	0.67	378,455
200	18%	0.75	0.70	163
500	18%	0.75	0.735	51
1000	18%	0.75	0.75	3

Further, there was a strong negative correlation (-0.92) between the compression ratio and the number of association rules indicating that market basket data sets that satisfy many association rules are very compressible.

VI. CONCLUDING REMARKS

Compression ratio of a file can be computed fast and easy, and in many cases offers a cheap way of predicting the existence of embedded patterns in data. Thus, it becomes possible to obtain an approximative estimation of the usefulness of an in-depth exploration of a data set using more sophisticated and expensive algorithms.

The presence of patterns in strings leads to a high degree of compression (that is, to low compression ratios). Thus, a low compression ratio for a file indicates that the mining process may produce interesting results. Compressibility however, does not guarantee that a sequence contains repetitions. Strings that are free of repetitions but contain patterns can display a high degree of compressibility as shown by the well-known Thue-Morse binary string.

The use of compression as a measure of minability is illustrated on a variety of paradigms: text data, market basket data, etc. Compression has been applied in bioinformatics as a tool for reducing the size of immense data sets that are generated in the genomic studies. Furthermore, specialized algorithms were developed to mine data in compressed form [27].

Our current work shows that identifying compressible areas of human DNA by comparing the compressibility of certain genomic regions is a useful tool for detecting areas where the gene replication mechanisms are disturbed (a phenomenon that occurs in certain genetically based diseases).

REFERENCES

[1] H. Mannila, "Theoretical frameworks for data mining," *SIGKDD Exploration*, vol. 1, pp. 30–32, 2000.

[2] E. Keogh, S. Lonardi, and C. A. Ratanamahatana, "Towards parameter free data mining," in *Proc. 10th ACM SIGKDD Intnl Conf. Knowledge Discovery and Data Mining*, 2004, pp. 206–215.

[3] E. Keogh, S. Lonardi, C. A. Ratanamahatana, L. Wei, S. Lee, and J. Handley, "Compression-based data mining of sequential data," *Data Mining and Knowledge Discovery*, vol. 14, pp. 99–129, 2007.

[4] E. J. Keogh, L. Keogh, and J. Handley, "Compression-based data mining," in *Encyclopedia of Data Warehousing and Mining*, 2009, pp. 278–285.

[5] L. Wei, J. Handley, N. Martin, T. Sun, and E. J. Keogh, "Clustering workflow requirements using compression dissimilarity measure," in *Proc. ICDM Workshops*, 2006, pp. 50–54.

[6] B. J. L. Campana and E. J. Keogh, "A compression based distance measure for texture," in *SDM*, 2010, pp. 850–861.

[7] A. Siebes, J. Vreeken, and M. van Leeuwen, "Items sets that compress," in *Proc. SIAM International Conference on Data Mining*, 2006, pp. 393–404.

[8] J. Vreeken, M. van Leeuwen, and A. Siebes, "KRIMP: Mining items that compress," *Data Mining and Knowledge Discovery*, vol. 23, pp. 169–214, 2011.

[9] T. Welch, "A technique for high performance data compression," *IEEE Computer*, vol. 17, pp. 8–19, 1984.

[10] D. A. Simovici, S. Baraty, and D. Pletea, "Evaluating data minability through compression – an experimental study," *International Journal on Advances in Software*, vol. 6, no. 3-4, pp. 237–245, 2013.

[11] J. P. Allouche and J. Shallit, "The ubiquitous Prouhet-Thue-Morse sequence," in *Sequences and their Applications*, Springer London, 1999, pp. 1–16.

[12] L. Youran R. Ricklund, and S. Mattias, "The Thue-Morse aperiodic crystal, a link between the Fibonacci quasicrystal and the periodic crystal," *International Journal of Modern Physics B*, vol. 1, pp. 121–132, 1987.

[13] R. Yarlagadda and J. Hershey, "Counter synchronization using the Thue-Morse sequence and psk," *IEEE Transactions on Communications*, vol. 32, pp. 947–977, 1984.

[14] T. Kuyel D. Chen L. Jin, K. Parthasarathy, and R. Geiger, "Accurate testing of analog-to-digital converters using low linearity signals with stimulus error identification and removal,"

IEEE Transactions on Instrumentation and Measurement, vol. 54, pp. 1188–1199, 2005.

[15] L. Jin, K. L. Parthasarathy, T. Kuyel, R. L. Geiger, and D. Chen, "High-performance adc linearity test using low-precision signals in nonstationary environments," in *Proc. IEEE International Test Conference*, 2005, pp. 1182–1191.

[16] M. Morse and G. A. Hedlund, "Unending chess, symbolic dynamics, and a problem in semigroups," *Duke Mathematical Journal*, vol. 11, pp. 1–7, 1944.

[17] A. Salomaa, *Jewels of Formal Language Theory*, Rockville, Maryland: Computer Science Press, 1981.

[18] A. Salomaa, *Formal Languages*, New York: Academic Press, 1973.

[19] D. A. Simovici and R. L. Tenney, *Formal Language Theory with Applications*, World Scientific, Singapore, 1999.

[20] C. Manning and H. Schutze, *Foundations of Statistical Natural Language Processing*, Cambridge, MA: MIT Press, 1999.

[21] A. Srivastava and M. Sahami, *Text Mining: Classification, Clustering, and Applications*, Boca Raton: CRC Press, 2009.

[22] N. Indurkhya and F. Damerau, *Handbook of Natural Language Processing*, Second ed., Boca Raton: CRC Press, 2010.

[23] R. Feldman and J. Sanger, *The Text Mining Handbook*, Cambridge: Cambridge University Press, 2006.

[24] R. Bilisoly, *Practical Text Mining with Perl*, New York: John Wiley and Sons, 2008.

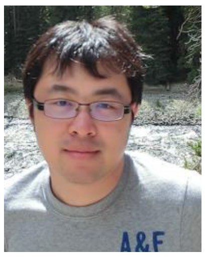
[25] S. Ananiadou and J. McNaught, *Text Mining for Biology and Biomedicine*, Norwood MA, Artech House, 2006.

[26] L. Cristofor. (2000). The ARMiner Project, Univ. of Massachusetts Boston. [Online]. Available: http://www.cs.umb.edu/ _laur/ARMiner

[27] P. R. Loh, M. Baym, and B. Berger, "Compressive genomics," *Nature Biotechnology*, vol. 30, pp. 627–630, 2012.

Dan Simovici Dr. Dan Simovici is a professor of Computer Science at University of Massachusetts Boston and an associate member of Dana-Farber Cancer Institute. His main research interests are in Data Mining and in the algebraic aspects of multiple-valued logic. Dr. Simovici is the author or coauthor of more than 160 research papers and he co-authored several books. His latest book" Mathematical Tools for Data Mining" appeared this year in its second edition at Springer.

Ping Chen Dr. Ping Chen is an Associate Professor of Computer Engineering and the Director of Artificial Intelligence Lab at the University of Massachusetts Boston. His research interests include Bioinformatics, Data Mining, and Computational Semantics. Dr. Chen has received five NSF grants and published over 50 papers in major Data Mining, Artificial Intelligence, and Bioinformatics conferences and journals. Dr. Ping Chen received his BS degree on Information Science and Technology from Xi'an Jiao Tong University, MS degree on Computer Science from Chinese Academy of Sciences, and Ph.D degree on Information Technology at George Mason University.



Tong Wang Tong Wang is a second year PhD student of Computer Science at the University of Massachusetts Boston advised by Dr. Ping Chen. His research interests include Data Mining, Machine Learning and Natural Language Processing. Tong received his BS degree on Information and Computing Science at Huazhong Agricultural University in China, MS degree on Computer Systems Engineering from Northeastern University in US.