Mr. Shakespeare, Meet Mr. Tucker, Part III

Charles Nicholas

Overview of Research in the DREAM Lab

CMSC 491/691 Spring 2021

April 21, 2021

DREAM Lab

- Discovery, Research, Exploration, and Analysis of Malware
- Home of the CyberDawgs `https://umbccd.umbc.edu/`
- Home of the Malware Research Group `http://groups.google.com/`

Areas of Work

- Methods for Detecting and Comparing Executable Binaries, with Applications to Clustering
- Tensor Algebra Methods for Static (and Dynamic) Malware Analysis
- Evaluation of Malware Classifiers (for another time)

Comparing BLOBS

Comparing large binary objects can be tricky and expensive. We describe a method for comparing such objects, based on ideas used for data compression, that is both fast and effective. We implemented the LZJD distance metric, and ran experiments on data that represent areas of interest to the cyber community. [1]

[1] If you don't want to listen to the first half of the talk, please see Raff, E., Nicholas, C., 2017. "An alternative to NCD for large sequences, Lempel-Ziv Jaccard Distance. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD 2017.

Normalized Compression Distance

Li et al. [5] define a function $C(x)$, which returns the length of string x when compressed, we get the NCD distance (1), where $C(xy)$ is the length of the string that results when strings x and y are concatenated and then compressed.

$$\mathrm{NCD}(x, y) = \frac{C\left(xy\right) - \min\left(C(x), C(y)\right)}{\max\left(C(x), C(y)\right)} \tag{1}$$

(Technically, the string x and the compression algorithm are both parameters of C, but we for now we assume that some compression algorithm has been chosen, and we use the shorter, more convenient notation.)

NCD: Advantages and Disadvantages

+ Intuitive: easy to explain, even with gestures
+ Easy to implement: even with a few lines of shell script
+ Versatile: works on many kinds of input objects, as long as they're not already compressed too much

– Not a good fit: compression algorithms are usually not designed with similarity functions, especially NCD, in mind
– Inefficient: compression is relatively expensive, and the object $C(ab)$ is used once and discarded
– Not a distance metric: identity, symmetry, and triangularity are not preserved

Observations on NCD

- Many compression algorithms make use of the Lempel-Ziv (LZ) technique for creating a compression dictionary of previously seen sub-strings[8, 9]
- We do not care about the actual compressed output of any compression algorithm when computing NCD, just the length of that output
- Other things being equal, we prefer a distance metric (which satisfies the properties of identity, symmetry, and triangularity)

Lempel-Ziv Jaccard Distance

Given a string a, we can compute its compression dictionary LZSet(a) in reasonable ($O(n)$) time using Algorithm LZSet (defined in a moment)
Given two strings a and b to be compared, we define the Lempel-Ziv Jaccard Distance (LZJD) metric as the Jaccard similarity between LZSet(a) and LZSet(b)

Lempel-Ziv Jaccard Distance, LZSet Algorithm

Algorithm 1 Simplified Lempel-Ziv Set

```
1: procedure LZSet(Byte sequence $b$)
2:     $s \leftarrow \emptyset$
3:     $start \leftarrow 0$
4:     $end \leftarrow 1$
5:     while $end < |b|$ do
6:         $b_s \leftarrow b[start : end]$
7:         if $b_s \notin s$ then
8:             $s \leftarrow s \cup \{b_s\}$
9:             $start \leftarrow end$
10:        end if
11:        $end \leftarrow end + 1$
12:    end while
13:    return $s$
14: end procedure
```

Lempel-Ziv Jaccard Distance

The Jaccard similarity between two sets A and B is defined as:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{2}$$

and we define LZJD for two strings x and y as:

$$\mathrm{LZJD}(x, y) = 1 - J(\mathrm{LZSet}(x), \mathrm{LZSet}(y)) \tag{3}$$

Since the metric space properties (identity, symmetry, and triangularity) hold for Jaccard in general, they hold for LZJD specifically.

Min-Hashing

Jaccard similarity as defined in Equation (3) works well for small LZsets, but the set intersection operation ($O(n \log n)$) is a little too slow. There are approximation methods for set intersection, and therefore LZJD, which are more efficient.
Let $h(a)$ be a hash function that returns an integer given some input string a, and $h_{min}(A) = \min_{a \in A} h(a)$ returns the minimum hash value over every object a in a set A. Then it has been shown ([1]) that

$$J(A, B) = P(h_{min}(A) = h_{min}(B))$$

That is, for two sets A and B, the Jaccard similarity of A and B is equal to the probability that the min hash values over the sets A and B are equal.

So What? Applications of LZJD

Given a fast and effective similarity metric, several tasks become easier:

- malware triage - have we seen anything like this before?
- malware detection - any malware in memory now?
- malware clustering - any new malware families today?
- forensics[2]

[2] If you don't want to listen to the rest of the talk, please see Raff, E., Nicholas, C., "Lempel-Ziv Jaccard Distance, an effective alternative to ssdeep and sdhash", Digital Investigation 24 (2018) pages 34-49.

Forensics tools: ssdeep and sdhash

Forensics questions include

- can we rebuild this corrupted file?
- any illicit materials on this computer?
- are there other versions of this file?
  In computer forensics, fuzzy similarity is often computed using either ssdeep, or sdhash.
  The ssdeep program is fast, but brittle. The sdhash program is slower, but handles more file types, and is better at finding more complicated patterns

Comparing ssdeep, sdhash, and LZJD

- When comparing two files, LZJD is comparable to ssdeep in speed, and much (orders of magnitude) faster than sdhash.
- When comparing two files, the LZJD score can be interpreted as a lower bound on how similar the binary contents of two files are. The scores from ssdeep and sdhash have no such interpretability.
- LZJD is better at matching a file fragment with its source file (i.e., the source file receives the highest matching score compared to all other files) compared to both ssdeep and sdhash. (See the paper for table and graphs.)
- The digest size, or "thumbnail" generated by LZJD is of fixed size, making index construction simpler.

Conclusions and Next Steps

LZJD gives accuracy and efficiency far beyond *NCD*
BWMD is even faster than *LZJD* - a talk for another day In progress:

- Clustering malware corpora,
- Hierarchical clustering...
- Finding "surprises" in systems that are mostly benign?!

Applications of Tensor Decomposition

Objectives

To investigate the use of tensor decomposition in static malware analysis - on a large scale

- Malware analysis is often done "in the small", that is, on one specimen at a time [7]
- We need to do malware analysis "in the large"
- Can we use tensor decomposition to gain insight into large collections of malware?

Building the Tensor

We selected a specific malware family, the well-known Zeus Trojans (Mohaisen, Alrawi, and Mohaisen [6]), as test subjects.
The tensor X is constructed so that: for each Zeus file i, entry $x_{i,j,k}$ is how many times 4-gram j occurs in decile k of the file. That is,

- $1 <= i <= 8020$, the number of Zeus specimens available to us
- $1 <= j <= 2^{32}$, the upper bound on the number of distinct 4-grams. The actual number of distinct 4-grams of course varies from file to file.
- $1 <= k <= 10$, since we chose to represent the approximate location in each specimen by dividing each specimen into ten parts of equal length.

Tucker Decomposition from Kolda Kolda and Bader [4]

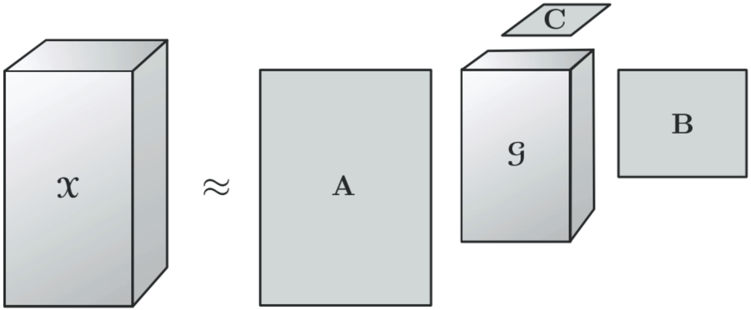
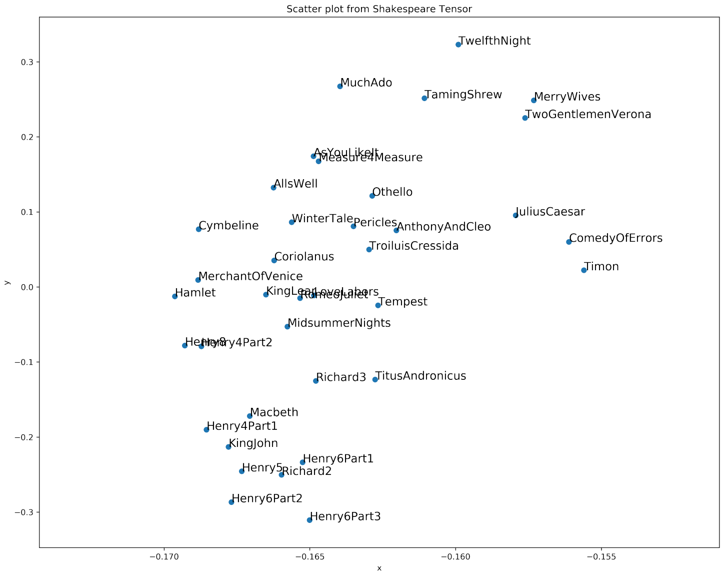


Fig. 4.1 *Tucker decomposition of a three-way array.*

Sanity Check - Shakespeare

Before trying the Zeus data, we wanted to try a smaller corpus - the Shakespearean plays. [3]
Using Python packages `sklearn` (to parse the text data) and `tensorD`Hao et al. [2] and `tensorflow` (to do the tensor calculations), in a Jupyter Notebook, we built the tensor X as described earlier, and ran both HOSVD and HOOI versions of Tucker.

Plot of First Two Factors from Tucker Decomposition



Tensor and Results

- In the Shakespearean tensor X, entry $x_{i,j,k}$ is the number of times word j occurs in Act k of play i. The value of i ranges from 1 to 37, j ranges from 1 to about 30,000, and k ranges from 1 to 5. The tensor is quite sparse.
- Plotting the first two factors produced by HOOI, HOSVD gave similar results
- We are pleased with the (unsupervised!) clustering of the history plays at the bottom of the plot.

Research Continues: Update

- Malware binaries will have *many* more terms than Shakespeare does, so we must be selective.
- Only some of the Zeus binaries are unpacked, so focus on those first.

Acknowledgements

- This work was supported by a research contract with the Laboratory for Physical Sciences, and a gift from Cisco
- An earlier version of this talk was presented as a poster at the High Performance Computing and Data Analytics Workshop, September 10-11, 2019.
- Email: nicholas@umbc.edu

References I

A.Z. Broder et al. "Syntactic clustering of the web". In: *Computer Networks and ISDN Systems* 29.8-13 (Sept. 1997), pp. 1157–1166. doi: 10.1016/S0169-7552(97)00031-7.

Liyang Hao et al. "TensorD: A tensor decomposition library in TensorFlow". In: *Neurocomputing* 318 (Nov. 2018), pp. 196–200. doi: 10.1016/j.neucom.2018.08.055.

Phani Teja Kesha. *Detection of Malware using Tensor Decomposition*. Tech. rep. UMBC M.S. Writing Project, 2019.

Tamara G. Kolda and Brett W. Bader. "Tensor Decompositions and Applications". In: *SIAM Review* 51.3 (2009), pp. 455–500. doi: 10.1137/07070111X.

M. Li et al. "The Similarity Metric". In: *IEEE Transactions on Information Theory* 50.12 (Dec. 2004), pp. 3250–3264. doi: 10.1109/TIT.2004.838101.

References II

Abedelaziz Mohaisen, Omar Alrawi, and Omar Mohaisen Abedelaziz Alrawi. “Unveiling Zeus: automated classification of malware samples”. In: *Proceedings of the 22nd international conference on World Wide Web companion*. 2013, pp. 829–832. isbn: 978-1-4503-2038-2.

Michael Sikorski and Andrew Honig. *Practical Malware Analysis*. no starch press, 2012.

Jacob Ziv and Abraham Lempel. “A universal algorithm for sequential data compression”. In: *IEEE Transactions on Information Theory* 23.3 (May 1977), pp. 337–343. doi: 10.1109/TIT.1977.1055714.

Jacob Ziv and Abraham Lempel. “Compression of individual sequences via variable-rate coding”. In: *IEEE Transactions on Information Theory* 24.5 (Sept. 1978), pp. 530–536. doi: 10.1109/TIT.1978.1055934.