Journal of Machine Learning Research 17 (2016) 1-5 Submitted 11/14; Revised 5/15; Published 3/16

Harry: A Tool for Measuring String Similarity

Konrad Rieck KONRAD.RIECK@CS.UNI-GOETTINGEN.DE
Christian Wressnegger CHRISTIAN.WRESSNEGGER@CS.UNI-GOETTINGEN.DE
University of Göttingen
Goldschmidtstraße 7
37077 Göttingen, Germany

Editor: Antti Honkela

Abstract

Comparing strings and assessing their similarity is a basic operation in many application domains of machine learning, such as in information retrieval, natural language processing and bioinformatics. The practitioner can choose from a large variety of available similarity measures for this task, each emphasizing different aspects of the string data. In this article, we present Harry, a small tool specifically designed for measuring the similarity of strings. Harry implements over 20 similarity measures, including common string distances and string kernels, such as the Levenshtein distance and the Subsequence kernel. The tool has been designed with efficiency in mind and allows for multi-threaded as well as distributed computing, enabling the analysis of large data sets of strings. Harry supports common data formats and thus can interface with analysis environments, such as Matlab, Pylab and Weka.

Keywords: string kernels, string distances, similarity measures for strings

1. Introduction

The comparison of strings is a basic operation in many applications of machine learning. Several problems of information retrieval and natural language processing center on comparing strings (see Salton and McGill, 1986). Similarly, several tasks in bioinformatics revolve around assessing the similarity of sequences (see Borgwardt, 2011). The problem underlying these tasks—*measuring the similarity or dissimilarity of two strings*—has been a vivid topic of research for over five decades, ranging from early telecommunication to modern machine learning and data analysis. As a result, the practitioner can choose from a large variety of available methods for assessing the similarity of strings[1], each emphasizing different aspects and characteristics of the data.

In this article we present Harry, a small tool specifically designed for measuring the similarity of strings and making various comparison methods available for analysis of string data. Harry implements over 20 common similarity measures, including classic string distances, such as the Damerau (1964) and Levenshtein (1966) distance, as well as modern string kernels, such as the Spectrum and Subsequence kernel (Lodhi et al., 2002). As the pairwise comparison of strings usually induces a quadratic run-time, Harry has been designed with efficiency in mind and allows for multi-threaded as well as distributed computing, which enables calculating large distance and kernel matrices. Harry supports common data formats and thus can interface with analysis tools and environments, such as Matlab, Pylab, Weka and LibSVM.

1. For ease of presentation, we use the term *similarity* synonymously with *dissimilarity* in this article.

While there also exist other tools implementing similarity measures for strings, such as the popular module python-Levenshtein, the generic toolbox Shogun (Sonnenburg et al., 2010) and the R package KeBABS for biological sequences (Palme et al., 2015), none of these tools provides a similarly broad range of similarity measures in comparison with Harry. Moreover, Harry complements the tool Sally (Rieck et al., 2012) which maps strings to vectors and allows for applying vectorial comparison functions to strings, such as the Euclidean and Manhattan distance. In combination, Harry & Sally provide a versatile basis for analyzing string data.

2. A Brief Overview of Harry

Harry implements a generic framework for the comparison of strings and assessing their similarity. In the following, we briefly discuss the main features of this framework.

2.1 Supported Similarity Measures

The current version of Harry supports the similarity measures listed in Table 1. Included are classic string distances by Damerau (1964) and Levenshtein (1966) as well as more recent methods, such as distances by Jaro (1989) and Bennett et al. (1998). Furthermore, the tool implements string kernels as described by Shawe-Taylor and Cristianini (2004) and allows for mapping kernels to distances and vice versa. In addition to distances and kernels, Harry also implements so-called similarity coefficients, such as the Jaccard index, which assess similarity in terms of matching sets of characters or words (Sokal and Sneath, 1963).

String Distances (10)			
Bag distance	Hamming distance	Kernel-substitution distance	String alignment distance
Compression distance	Jaro distance	Lee distance	
Damerau-Levenshtein distance	Jaro-Winkler distance	Levenshtein distance	
String Kernels (4)			
Distance-substitution kernel	Spectrum kernel	Subsequence kernel	Weighted-degree kernel
Similarity Coefficients (7)			
Braun-Blanquet coefficient	Kulczynski coefficient	Simpson coefficient	Sokal-Sneath coefficient
Jaccard coefficient	Otsuka coefficient	Soerensen-Dice coefficient	

Table 1: Similarity measures for strings supported by Harry (version 0.4.1).

2.2 Scalable Computation

For the efficient processing of large sets of strings, Harry makes use of multi-threading and distributes the workload over multiple CPU cores (see option `-n`). The run-time performance of Harry thus scales linearly with the number of cores and enables computing large distance and kernel matrices. Furthermore, Harry can split the calculation of large matrices into smaller chunks, where these chunks are either defined manually as sub-matrices (see options `-x` and `-y`) or automatically from the number of requested splits (option `-s`). Using this splitting the computation of similarity measures can be easily distributed over multiple hosts and carried out with systems for high performance computing, such as the LSF or SGE platform.

The run-time performance of Harry is further improved by caching recurrent computations (option `-a`), such as during the normalization of string kernels and distances. The underlying cache builds on a lightweight synchronization that ensures little overhead, even if several threads concurrently access the data. The caching can also be applied directly to the computed similarity values (option `-G`) and speed up the comparison of data sets with duplicate strings.

2.3 Interfaces and Preprocessing

To interface with other analysis tools and environments, Harry implements support for common input and output formats. The tool can read string data from text files, directories and compressed archives (option `-i`). Moreover, it is able to store computed similarity matrices in output formats suitable for Matlab, Pylab, Weka and LibSVM (option `-o`). Additionally, Harry is bundled with a Python module that enables efficiently comparing strings in Python without storing data on disk.

Harry supports several preprocessing functions that improve string comparison in particular fields of application. For example, the tool enables the user to change the granularity of the comparison to either bytes, bits or tokens (option `-g`). In the latter setting the input strings are partitioned into tokens using a set of delimiter characters, thereby enabling the analysis of structured data, such as text and log entries. Moreover, Harry supports removing stop tokens from strings, transforming tokens to Soundex codes and encoding non-printable characters in texts.

The data format, preprocessing functions and the selected similarity measure can be specified on the command line as well as in a configuration file (option `-c`). As a result, experiments with Harry can be easily reproduced using stored configurations.

3. Experiments with Harry

We demonstrate the efficiency of Harry in an empirical evaluation, where we first study its scalability (Section 3.1) and then compare its performance to related tools (Section 3.2). In all experiments we consider the data sets listed in Table 2 which contain strings of DNA snippets, protein sequences, Twitter messages and network traces, respectively.

Data set	**ARTS**	**SPROT**	**TWEETS**	**WEBFP**
Type of strings	DNA snippets	Protein sequences	Twitter messages	Network traces
Average length of strings	2,400 bases	457 proteins	126 characters	1,312 packets
Size of alphabet	4 bases	22 proteins	69 characters	6 packet sizes

Table 2: Data sets for evaluation. Each data set consists of 1,000 strings randomly drawn from the original source: ARTS (Sonnenburg et al., 2006); SPROT (O'Donovan et al., 2002); TWEETS (Twitter.com); WEBFP (Cai et al., 2012).

3.1 Scalability of Harry

In our first experiment, we compute the Levenshtein (1966) distance, the normalized compression distance (Bennett et al., 1998) and the Subsequence kernel (Lodhi et al., 2002) on all four data sets using Harry. We repeat the computation with a different number of available CPU cores and measure the run-time in terms of comparisons per second.

Figure 1 shows the results of this experiment. The Levenshtein distance and the Subsequence kernel scale perfectly linear with the number of CPU cores, reaching peak performances of 10^5 and 10^4 comparisons per second, respectively. The compression distance scales almost linearly, as the caching used for normalization induces a slight overhead when more than 8 cores are used.

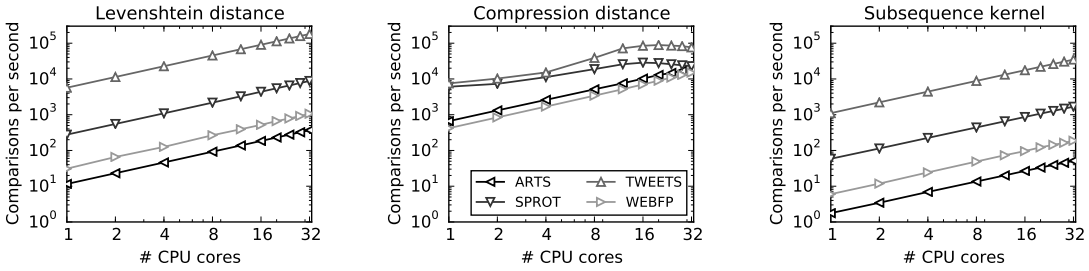

Figure 1: Run-time performance of Harry with varying number of CPU cores.

3.2 Comparative Evaluation

In the second experiment, we compare Harry with other tools for measuring string similarity. We consider the Python modules *python-Levenshtein* (0.11.2) and *python-jellyfish* (0.5.0) that implement the Levenshtein distance and its variants, the library *CompLearn* (1.1.7) that focuses on compression distances, and the machine learning toolbox *Shogun* (4.0.0) that provides several string kernels. For each of the four data sets, we randomly draw 100 strings, compute a full similarity matrix with each tool and measure the run-time over 10 runs.

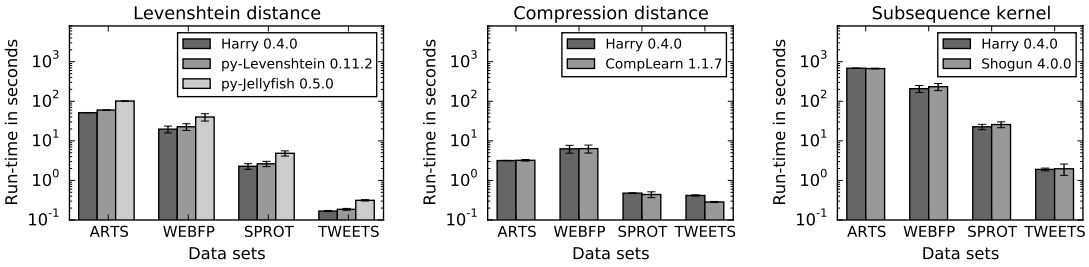

Figure 2: Comparative evaluation of Harry and related software tools.

The averaged results for the comparative evaluation are shown in Figure 2. In all settings, Harry is on par with the other tools and provides a similar and often even better performance. Given that each of the related tools focuses only on a subset of similarity measures, this experiment demonstrates the versatile yet efficient implementation of Harry.

4. Conclusions

Harry provides access to a wide range of similarity measures and enables their efficient computation on string data. In combination with Sally (Rieck et al., 2012), the tool is a perfect fit for analyzing and learning with strings. The source code of Harry along with documentation and a tutorial is available at *http://www.mlsec.org/harry*.

Acknowledgments

The authors gratefully acknowledge funding from BMBF (16KIS0154K) and DFG (RI 2469/1-1).

References

C. Bennett, P. Gacs, M. Li, P. Vitanyi, and W. Zurek. Information distance. *IEEE Transactions on Information Theory*, 44(4):1407–1423, July 1998.

K. M. Borgwardt. *Kernel Methods in Bioinformatics*, chapter Handbook of Statistical Bioinformatics, pages 317–334. Springer, 2011.

X. Cai, X. C. Zhang, B. Joshi, and R. Johnson. Touching from a distance: Website fingerprinting attacks and defenses. In *Proc. of ACM Conference on Computer and Communications Security (CCS)*, pages 605–616, 2012.

F. Damerau. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3):171–176, 1964.

M. A. Jaro. Advances in record linkage methodology as applied to the 1985 census of tampa florida. *Journal of the American Statistical Association*, 84(406):414–420, 1989.

V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Doklady Akademii Nauk SSSR*, 163(4):845–848, 1966.

H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins. Text classification using string kernels. *Journal of Machine Learning Research*, 2:419–444, 2002.

C. O'Donovan, M. Martin, A. Gattiker, E. Gasteiger, A. Bairoch, and R. Apweiler. High-quality protein knowledge resource: SWISS-PROT and TrEMBL. *Briefings in Bioinformatics*, 3(3): 275–284, 2002.

J. Palme, S. Hochreiter, and U. Bodenhofer. KeBABS: an R package for kernel-based analysis of biological sequences. *Bioinformatics*, 2015. (doi: 10.1093/bioinformatics/btv176).

K. Rieck, C. Wressnegger, and A. Bikadorov. Sally: A tool for embedding strings in vector spaces. *Journal of Machine Learning Research (JMLR)*, 13(Nov):3247–3251, Nov. 2012.

G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1986.

J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

R. Sokal and P. Sneath. *Principles of Numerical Taxonomy*. W.H. Freeman and Company, San Francisco, CA, USA, 1963.

S. Sonnenburg, A. Zien, and G. Rätsch. ARTS: Accurate recognition of transcription starts in human. *Bioinformatics*, 22(14):e472–e480, 2006.

S. Sonnenburg, G. Rätsch, S. Henschel, C. Widmer, J. Behr, A. Zien, F. de Bona, A. Binder, C. Gehl, and V. Franc. The shogun machine learning toolbox. *Journal of Machine Learning Research (JMLR)*, 11(Jun):1799–1802, 2010.