Harry Version 0.4.3
— User Manual —

Konrad Rieck

May 8, 2019

Contents

1 NAME

harry - A tool for measuring string similarity

2 SYNOPSIS

harry [**options**] [**-c** *config*] *input* [*input*] *output*

3 DESCRIPTION

harry is a small tool for measuring the similarity of strings. The tool supports common distance and kernel functions for strings as well as some exotic similarity measures. The focus of **harry** lies on implicit similarity measures, that is, comparison functions that do not give rise to an explicit vector space. Examples of such similarity measures are the Levenshtein distance, the Jaro-Winkler distance or the normalized compression distance.

During operation **harry** loads a set of strings from *input*, computes the specified similarity measure and writes a matrix of similarity values to *output*. If one *input* is given, **harry** computes the similarities between all strings in *input*. If two *input* sources are provided, **harry** computes only the similarities between the two sources. The similarity measure can be computed based on the granularity of bits, bytes as well as tokens contained in the strings. The configuration of this process, such as the input format, the similarity measure and the output format, are specified in the file *config* and can be additionally refined using command-line options.

```
.-------.                   .--------.
| lines |                   |  text  |
|  dir  |  \  .-------.  /  | libsvm |
| stdin |  -- | Harry |  -- | matlab |
|  ...  |  /  '-------'  \  |  ...   |
'-------'                   '--------'
  Input    Similarity Measures Output
```

harry is implemented using OpenMP, such that the computation time for a set of strings scales linear with the number of available CPU cores. Moreover, efficient implementations of several similarity measures, effective caching of similarity values and low-overhead locking further speedup the computation.

harry complements the tool **sally**(1) that embeds strings in a vector space and allows computing vectorial similarity measures, such as the cosine distance and the bag-of-words kernel.

4 CONFIGURATION

The configuration of **harry** is provided by a configuration file. This file is structured into the three sections **input**, **measures** and **output**, which define the parameters of the input format, the similarity measures and the output format, respectively. If no configuration file is provided, **harry** resorts to a default configuration. This default configuration can be printed using the command-line option **-D** (see **OPTIONS**).

Input formats

harry supports different formats for reading sets of strings, which may range from plain files to directories and other structured resources. The input format is specified in the configuration file of **harry**, but can also be defined on the command line using the option **-i** (see **OPTIONS**). Following is a list of supported input formats:

input = {

input_format = ″lines″;
This parameter specifies the input format.

″lines″
The input strings are available as lines in a text file. The name of the file is given as *input* to **harry**. The lines need to be separated by newline and may not contain the NUL character. Labels can be extracted from each line using a regular expression (see **lines_regex**).

″stdin″
The input strings are provided on standard input (stdin) as text lines. The lines need to be separated by newline and may not contain the NUL character. Labels can be extracted from each line using a regular expression (see **lines_regex**). This input format is also enabled when *input* is set to -, otherwise *input* is ignored.

″dir″
The input strings are available as binary files in a directory and the name of the directory is given as *input* to **harry**. The suffixes of the files are used as labels for the strings.

″arc″
The input strings are available as binary files in a compressed archive, such as a zip or tgz archive. The name of the archive is given as *input* to **harry**. The suffixes of the files are used as labels for the strings.

″fasta″
The input strings are available in FASTA format. The name of the file is given as *input* to **harry**. Labels can be extracted from the description of each sequence using a regular expression (see **fasta_regex**). Comments are allowed if they are preceded by either ′;′ or ′>′.

″raw″
The input strings are provided on standard input (stdin) in ″raw″ format. This input module is designed to efficiently interface with other environments. The binary format for strings has the form

```
| len (uint32) | array (uint8) ... |
| len (uint32) | array (uint8) ... |
| ...
```

where *len* is a 32-bit unsigned integer in host byte order indicating the length of the following *array* containing the string data in bytes. Labels cannot be extracted from this representation. This input format is also enabled when *input* is set to =, otherwise *input* is ignored.

chunk_size = 256;

To enable an efficient processing of large data sets, **harry** loads strings in chunks. This parameter defines the number of strings in one of these chunks. Depending on the lengths and type of the strings, this parameter can be adjusted to improve loading times.

decode_str = false;

If this parameter is set to *true*, **harry** automatically decodes strings that contain URI-encoded characters. That is, substrings of the form %XX are replaced with the byte corresponding to the hexadecimal number XX.

fasta_regex = " (\\+|-)?[0-9]+";

The FASTA format allows to equip each string with a short description. In several data sets this description contains a numerical label which can be used for supervised learning tasks. The parameter defines a regular expression that matches these numerical labels, such as +1 and -1.

lines_regex = "^(\\+|-)?[0-9]+";

If the strings are available as text lines, the parameter can be used to extract a numerical label from the strings. The parameter is a regular expression matching labels, such as +1 and -1.

reverse_str = false;

If this parameter is set to *true*, the characters of all input strings will be reversed. Such reversing might help in situations where the reading direction of the input strings is unspecified.

stoptoken_file = "";

Stop tokens (irrelevant tokens) can be filtered from the strings by providing a file containing these tokens; one per line. Non-printable characters can be escaped using URI encoding (%XX). Stop tokens can only be filtered, if the **granularity** is set to *tokens*.

soundex = false;

All tokens in the strings are mapped to the soundex index. For example, "Pfister" is mapped to "P236" and "Jackson" to "J250". The soundex index has been originally designed for comparing names, however, in **harry** it can be applied to all sorts of tokens, if they are composed of alphabetic letters. Punctation characters are ignored and thus the string "Hey, I am here with Harry!", gets mapped to "H000 I000 A500 H600 W300 H600".

};

Similarity Measures

harry supports different similarity measures for comparing string, including common distance and kernel functions. The similarity measure can be specified in the configuration file as well as on the command line using the option **-m** (see **OPTIONS**). The name of each similarity measure is prefixed by its type (*dist_* for distances, *kern_* for kernels and *sim_* for similarity measures). For convenience, this prefix can be omitted. Moreover, the names of some similarity measures are aliased, for example, the normalized compression distance is available as *dist_compression* and *dist_ncd*.

Parameters of the similarity measures are organized in individual configuration groups. For instance, parameters of the Levenshtein distance are defined in the group **dist_levenshtein**, while parameters for the Jaro and Jaro-Winkler distance are given in **dist_jarowinkler**.

measures = {

type = "dist_levenshtein"

The parameter **type** specifies the similarity measure that is used for comparing the strings. Supported similarity measures are:

"dist_hamming"
Hamming distance. See configuration group **dist_hamming**.

"dist_levenshtein", "dist_edit"
Levenshtein distance. See configuration group **dist_levenshtein**.

"dist_damerau"
Damerau-Levenshtein distance. See configuration group **dist_damerau**.

"dist_osa"
Optimal string alignment (OSA) distance. See configuration group **dist_osa**.

"dist_jaro"
Jaro distance. See configuration group **dist_jarowinkler**.

"dist_jarowinkler"
Jaro-Winkler distance. See configuration group **dist_jarowinkler**.

"dist_lee"
Lee distance. See configuration group **dist_lee**

"dist_compression", "dist_ncd"
Normalized compression distance (NCD). See configuration group **dist_compression**.

"dist_bag"
Bag distance. See configuration group **dist_bag**.

"dist_kernel"
Kernel substitution distance. See configuration group **dist_kernel**.

"kern_subsequence", "kern_ssk"
Subsequence kernel (SSK). See configuration group **kern_subsequence**.

"kern_spectrum", "kern_ngram"
Spectrum kernel (also n-gram kernel). See configuration group **kern_spectrum**.

"kern_wdegree", "kern_wdk"
: Weighted-degree kernel (WDK) with shifts. See configuration group **kern_wdegree**.

"kern_distance", "kern_dsk"
: Distance substitution kernel (DSK). See configuration group **kern_distance**.

"sim_simpson"
: Simpson coefficient. See configuration group **sim_coefficient**.

"sim_jaccard"
: Jaccard coefficient. See configuration group **sim_coefficient**.

"sim_braun"
: Braun-Blanquet coefficient. See configuration group **sim_coefficient**.

"sim_dice", "sim_czekanowski"
: Dice-coefficient (Czekanowsi coefficient) See configuration group **sim_coefficient**.

"sim_sokal", "sim_anderberg"
: Sokal-Sneath coefficient (Anderberg coefficient). See configuration group **sim_coefficient**.

"sim_kulczynski"
: Second Kulczynski coefficient. See configuration group **sim_coefficient**.

"sim_otsuka", "sim_ochiai"
: Otsuka coefficient (Ochiai coefficient). See configuration group **sim_coefficient**.

granularity = "bytes";

This parameter controls the granularity of strings. It can be set to either *bits*, *bytes* or *tokens*. Depending in the granularity a string is considered as a sequence of bits, bytes or tokens, which results in different similarity values during comparison.

token_delim = "";

The parameter **token_delim** defines characters for delimiting tokens in strings, for example " %0a%0d". It is only considered, if the granularity is set to *tokens*, otherwise it is ignored.

num_threads = 0;

The parameter **num_threads** sets the number of threads for the calculation of the similarity measures. If set 0, **harry** determines the number of available CPU cores using OpenMP and sets the number of threads accordingly.

cache_size = 256;

The parameter **cache_size** specifies the maximum size of the internal cache in megabytes (Mb). The general-purpose cache is used to speed up computations of **harry** for some similarity measures.

global_cache = false;

By default **harry** caches only internal computations. If this parameter is set to *true*, all similarity values are stored in the cache. This feature should only be enabled if many of the compared strings are identical and thus caching similarity values can provide benefits.

col_range = "";

row_range = "";

These two parameters control which strings are used for computing the matrix of similarity values. **col_range** defines a range of indices on the columns and **row_range** on the rows of the matrix. The format of the ranges is similar to indexing of Python arrays: A range is given by *"start:end"*, where *start* defines the index of the first string and *end* defines the index after the last string. For example, *"0:4"* selects the strings at index 0, 1, 2, and 3. If the start or end index is omitted, the minimum or maximum value is substituted, respectively. For example, *":4"* selects strings starting from the index *0* and *":"* chooses all strings. If the end index is negative, it is substracted from the maximum index, that is, *":-1"* selects all strings except for the last one.

The parameters **col_range** and **row_range** are ignore if two input sources are given on the command line.

split = "";

To ease the computation of large similarity matrices, **harry** supports automatically splitting a matrix into blocks. This splitting is defined by a string of the form *"blocks:idx"*, where *blocks* defines the number of blocks and *idx* the index of the block to compute. The matrix is splitted across the y-axis. For many output formats the blocks can be simply concatenated to get the original matrix.

The parameter **split** is ignore if two input sources are given on the command line.

dist_hamming = {

This module implements the Hamming distance (see Hamming, 1950). The runtime complexity of a comparison is linear in the length of the strings. If the compared strings have unequal length, the length difference is added to the distance. The following parameters are supported:

norm = "none";

This parameter specifies the normalization of the distance. Supported values are *"none"* for no normalization, *"min"* for normalization on the minimum length, *"max"* for normalization on the maximum length, *"avg"* for normalization on the average length of the compared strings.

};

dist_levenshtein = {

This module implements the Levenshtein distance (see Levenshtein, 1966). The runtime complexity of a comparison is quadratic in the length of the strings. The following parameters are supported:

norm = "none";

This parameter specifies the normalization of the distance. Supported values are *"none"* for no normalization, *"min"* for normalization on the minimum length, *"max"* for normalization on the maximum length, *"avg"* for normalization on the average length of the compared

cost_ins = 1.0;

cost_del = 1.0;

cost_sub = 1.0;

The computation of the distance can be adapted using three parameters defining the cost for an insertion, deletion and substitution, respectively. The default costs are *1.0* for each operation.

};

dist_damerau = {

This module implements the Damerau-Levenshtein distance (see Damerau, 1964). The runtime and space complexity of a comparison is quadratic in the length of the strings. The following parameters are supported:

norm = "none";

This parameter specifies the normalization of the distance. Supported values are *"none"* for no normalization, *"min"* for normalization on the minimum length, *"max"* for normalization on the maximum length, *"avg"* for normalization on the average length of the compared strings.

cost_ins = 1.0;

cost_del = 1.0;

cost_sub = 1.0;

cost_tra = 1.0;

The computation of the distance can be adapted using four parameters defining the cost for an insertion, deletion, substitution and transposition, respectively. The default costs are *1.0* for each operation.

};

dist_osa = {

This module implements the optimal string alignment (OSA) distance, which is often confused with the Damerau-Levenshtein distance. The difference between the two is that the OSA distance computes the number of edit operations needed to make the strings equal under the condition that no substring is edited more than once. (see the Wikipedia article on the Damerau-Levenshtein distance). The runtime and space complexity of a comparison is quadratic in the length of the strings. The following parameters are supported:

norm = "none";

This parameter specifies the normalization of the distance. Supported values are *"none"* for no normalization, *"min"* for normalization on the minimum length, *"max"* for normalization on the maximum length, *"avg"* for normalization on the average length of the compared strings.

cost_ins = 1.0;

cost_del = 1.0;

cost_sub = 1.0;

cost_tra = 1.0;

The computation of the distance can be adapted using four parameters defining the cost for an insertion, deletion, substitution and transposition, respectively. The default costs are *1.0* for each operation.

};

dist_jarowinkler = {

This module implements the Jaro distance (Jaro, 1989) and the Jaro-Winkler distance (Winkler, 1990). In contrast to the original formulation, a valid distance function is implemented, where similar strings yield a low value and dissimilar strings a high value. The runtime complexity of a comparison is quadratic in the length of the strings. The following parameters are supported:

scaling = 0.1;

If this parameter is set to *0*, the original Jaro distance is returned, otherwise the Jaro-Winkler distance is calculated. This distance uses a **scaling** which gives more favorable ratings to strings that match from the beginning up to 4 symbols. The default value is *0.1*.

};

dist_lee = {

This module implements the Lee distance (Lee, 1958) for strings. The runtime complexity of a comparison is linear in the length of the strings. If the compared strings have unequal length, the remaining symbols of the longer string are added to the distance. The following parameters are supported:

min_sym = 0; =item max_sym = 255;

These parameters specify the range of symbols, that is, the minimum and maximum value of a symbol in all strings. If the strings consist of bytes, **min_sym** is typically set to *0* and **max_sym** to *255*. For printable characters the range can be further narrowed to *32* and *126*. If tokens are analyzed using the parameter **token_delim**, **min_sym** must be set to 0 and **max_sym** to *65535*, as the tokens are mapped to integers in this range.

};

dist_compression = {

This module implements the normalized compression distance for strings (Cilibrasi and Vitanyi, 2005). The distance is "symmetrized". The compression is implemented using **zlib**. Note that the comparison of strings highly depends on the characteristics of the compressor (Cebrian et al., 2005). The strings should not be longer than 16 kilobytes, such that two strings fit into the window of **zlib**. The runtime complexity of a comparison is linear in the length of the strings, though with a large constant factor. The following parameters are supported:

level = 9;

This parameter defines the compression level used by **zlib** and must be between *1* and *9*, where *1* gives the best speed and *9* the best compression. See **zlib(3)**

};

dist_bag = {

This module implements the bag distance (see Bartolini et al., 2002). The distance approximates and lower bounds the Levenshtein distance. The runtime complexity of a comparison is linear in the length of the strings. The following parameters are supported:

norm = "none";
This parameter specifies the normalization of the distance. Supported values are *"none"* for no normalization, *"min"* for normalization on the minimum length, *"max"* for normalization on the maximum length, *"avg"* for normalization on the average length of the compared strings.

};

dist_kernel = {

This module implements a kernel-based distance, that is, a distance is computed given a kernel function for strings. The specified kernel function is mapped to a Euclidean distance using simple geometry. The runtime complexity depends on the kernel function. The following parameters are supported:

kern = "kern_wdegree";
This parameter selects the kernel function to use for the distance. The kernel is mapped to a Euclidean distance using simple geometry.

norm = "none";
This parameter specifies the normalization of the kernel. Supported values are *"none"* for no normalization and *"l2"* for the standard l2 normalization of kernels.

squared = true;
The module computes a Euclidean distance from the given kernel function. If this parameter is enabled a squared Euclidean distance is returned which is slightly faster due to the omitted root computation.

};

kern_wdegree = {

This module implements the weighted-degree kernel with shifts (Sonnenburg et al., 2007). The runtime complexity is linear in the length of the strings. If the strings have unequal length, the remaining symbols of the longer string are ignored, in accordance with the kernel definition. The following parameters are supported:

norm = "none";
This parameter specifies the normalization of the kernel. Supported values are *"none"* for no normalization and *"l2"* for the standard l2 normalization of kernels.

degree = 3;
This parameter specifies the degree of the kernel, that is, the length of considered k-mers/k-grams. As the kernel computation is implicit, the k-mers are not extracted but implicitly counted by blocks of matching symbols.

shift = 0;
To compensate noise in the strings, the kernel can be computed with "shifts". The strings are compared multiple times with different positive and negative offsets up to **shift** symbols. The different kernel values are added. The runtime complexity is increased by twice the value of **shift**.

};

kern_subsequence = {

This module implements the subsequence kernel (Lodhi et al., 2002). The runtime complexity is quadratic in the length of the strings. The following parameters are supported:

norm = "none";
This parameter specifies the normalization of the kernel. Supported values are *"none"* for no normalization and *"l2"* for the standard l2 normalization of kernels.

length = 3;
This parameter specifies the length of subsequence to consider.

lambda = 0.1;
This parameter is a weighting term for gaps within subsequences.

};

kern_spectrum = {

This module implements the spectrum kernel (Leslie et al., 2002). The runtime complexity is linear in the length of the strings. The spectrum kernel is closely related to bag-of-words kernels. Thus, the tool **sally(1)** may be alternatively used to compute the kernel using an explicit vector space. The following parameters are supported by the implementation:

norm = "none";
This parameter specifies the normalization of the kernel. Supported values are *"none"* for no normalization and *"l2"* for the standard l2 normalization of kernels.

length = 3;
This parameter specifies the length of k-mers/k-grams to consider.

};

kern_distance = {

This module implements distance substitution kernels (Haasdonk and Bahlmann, 2004). The empty string is considered the origin of the underlying implicit vector space. The runtime complexity depends on the selected distance function. The following parameters are supported:

dist = "dist_bag";
This parameter selects the distance function to use for the kernel. Depending on the type of the substitution and the selected distance, the kernel might not be positive semi-definite.

type = "linear";
Four types of substitutions can be selected for creating a kernel from a distance function: *"linear"*, *"poly"*, *"neg"* and *"rbf"*. For a detailed explanation of each substitution see the paper by Haasdonk and Bahlmann (2004).

norm = "none";
This parameter specifies the normalization of the kernel. Supported values are *"none"* for no normalization and *"l2"* for the standard l2 normalization of kernels.

gamma = 1.0;
This parameter specifies a scaling factor for the substitution types *"poly"* and *"rbf"*.

degree = 1.0;
This parameter defines a polynomial degree for the substitution types *"poly"* and *"neg"*.

};

sim_coefficient = {

This module implements several similarity coefficients for strings (see Cheetham and Hazel, 1969). The runtime complexity of a comparison is linear in the length of the strings. The following parameters are supported:

matching = "bin";
The parameter specifies how the symbols of the strings are matched. If the parameter is set to *"bin"*, the symbols are considered as binary attributes that are either present or not. If the parameter is set to *"cnt"*, the count of each symbol is considered for the matching.

};

};

Output formats

Once strings have been compared, **harry** stores the similarity values in one of several common formats, which allows for applying typical tools of statistics and machine learning to the data. Following is a list of supported output formats and respective parameters. Additionally, the output format can be specified using the command-line option **-o** (see **OPTIONS**).

output = {

output_format = "text";
Following is a list of output formats supported by **harry**:

"text"
The similarity values are stored as plain text.

"stdout"
The similarity values are written to standard output (stdout) as plain text. This output format is also enabled when *output* is set to -, otherwise *output* is ignored.

"libsvm"
The similarity values are stored as precomputed kernel for libsvm.

"json"
The similarity values are stored in JSON object.

"matlab"
The similarity values are stored in Matlab format (version 5).

"raw"

The similarity values are written to standard output (stdout) in raw format. This output module is designed for interfacing with other analysis environments. The format of the similarity matrix has the following form

```
| rows (uint32)  | cols (uint32)     |
| fsize (uint32) | array (float) ... |
```

where *rows* and *cols* are unsigned 32-bit integers specifing the dimensions of the matrix, *fsize* is the size of a float in bytes and *array* holds the matrix as floats. Indices, labels and sources are not output. This output format is also enables when *output* is set to =, otherwise *output* is ignored.

precision = 0;

Precision of the output in terms of decimal places. A precision of *0* selects the full single float range for output.

separator = ",";

This parameter defines the separator used in text mode for separating the similarity values.

save_indices = false;

If this parameter is to *true* and supported by the output format, the indices of the strings will be additionally stored.

save_labels = false;

If this parameter is to *true* and supported by the output format, the labels of the strings will be additionally stored.

save_sources = false;

If this parameter is to *true* and supported by the output format, the sources of the strings will be additionally stored.

compress = false;

If this parameter is set to *true*, the output is stored using zlib compression, which can significantly reduce the required disk space. Several programs support reading files compressed using zlib. Alternatively, the tools gzcat(1) and gunzip(1) can be used to access the data.

};

5 OPTIONS

The configuration of **harry** can be refined using several command-line options. Moreover, some parameters of the configuration can be overwritten on the command line. Following is the list of options:

I/O options

```
  -i,  --input_format <format>   Set input format for strings.
```

```
     --decode_str              Enable URI-decoding of strings.
     --reverse_str             Reverse (flip) all strings.
     --stoptoken_file <file>   Provide a file with stop tokens.
     --soundex                 Enable soundex encoding of tokens.
     --benchmark <seconds>     Perform benchmark run.
-o,  --output_format <format>  Set output format for matrix.
-p,  --precision <num>         Set precision of output.
-z,  --compress                Enable zlib compression of output.
     --save_indices            Save indices of strings.
     --save_labels             Save labels of strings.
     --save_sources            Save sources of strings.
```

Module options:

```
-m,  --measure <name>           Set similarity measure.
-g,  --granularity <type>       Set granularity: bytes, bits, tokens.
-d,  --token_delim <delim>      Set delimiters for tokens.
-n,  --num_threads <num>        Set number of threads.
-a,  --cache_size <size>        Set size of cache in megabytes.
-G,  --global_cache             Enable global cache.
-x,  --col_range <start>:<end>  Set the column range (x) of strings.
-y,  --row_range <start>:<end>  Set the row range (y) of strings.
-s,  --split <blocks>:<idx>     Split matrix into blocks and compute one.
```

Generic options:

```
-c,  --config_file <file>      Set configuration file.
-v,  --verbose                 Increase verbosity.
-l,  --log_line                Print a log line every minute
-q,  --quiet                   Be quiet during processing.
-M,  --print_measures          Print list of similarity measures
-C,  --print_config            Print the current configuration.
-D,  --print_defaults          Print the default configuration.
-V,  --version                 Print version and copyright.
-h,  --help                    Print this help screen.
```

6 FILES

PREFIX/share/doc/harry/example.cfg

An example configuration file for **harry**. See the configuration section for further details.

7 LIMITATIONS

harry supports only symmetric similarity measures, that is, m(x,y) = m(y,x) for all x and y. This restriction saves considerable run-time and memory in most cases. However, some similarity measures need to be artificially "symmetrized", such as the normalized compression distance, thereby requiring additional computations.

8 REFERENCES

Bartolini, Ciaccia, Patella. String Matching with Metric Trees Using an Approximate Distance. String Processing and Information Retrieval, LNCS 2476, 271-283, 2002.

Cebrian, Alfonseca, and Ortega. Common pitfalls using the normalized compression distance. Communications in Information and Systems, 5 (4), 367-384, 2005.

Cheetham and Hazel. Binary (Presence-Absence) Similarity Coefficients. Journal of Paleontology, 43:5, 1130-1136, 1969

Cilibrasi and Vitanyi. Clustering by compression, IEEE Transactions on Information Theory, 51:4, 1523-1545, 2005.

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

Haasdonk and Bahlmann. Learning with Distance Substitution Kernels. Pattern Recognition ; DAGM Symposium, 220-227, 2004.

Hamming. Error-detecting and error-correcting codes. Bell System Technical Journal, 29(2):147-160, 1950.

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.

Lee. Some properties of nonbinary error-correcting codes. IRE Transactions on Information Theory 4 (2): 77-82, 1958.

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

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

Sonnenburg, Raetsch, and Rieck. Large scale learning with string kernels. In Large Scale Kernel Machines, pages 73--103. MIT Press, 2007.

Winkler. String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage. Proceedings of the Section on Survey Research Methods. 354-359, 1990.

9 COPYRIGHT