Data-driven name reduction for record linkage

Marijn Schraagen and Walter Kosters
Leiden Institute of Advanced Computer Science
Leiden University, The Netherlands
{schraage,kosters}@liacs.nl

***Abstract*—Automatic record linkage of data containing personal names is difficult in the presence of name variation and spelling errors. This paper presents a standardization procedure for personal names to address the variation problem. A classification tree based model is constructed using a training set of 65,002 name-variant pairs. The method provides an efficient procedure for record linkage (3500 records per second, F-measure 0.96 on a sample of Dutch historical civil records). The results include links with large edit distance between the records, however recall is lower for this category. A bootstrapping procedure is used to improve recall.**

I. Introduction

Record linkage is the process of matching different records from a database that contain information about the same domain element (e.g., person). An example is linking patient records from different hospitals, or linking people from historical archives. In most record linkage approaches a match is based on similarity between fields that identify the subject of the record, such as person names. However, identification based on names (both given names and family names) can be difficult in the presence of name variation or spelling errors.

This paper presents a record linkage procedure based on the observation that some parts of a name appear to be more important for the identification of the name than others. The sequence of important characters can be considered the *core* of the name which remains constant between variants of the same name. An algorithm is presented to automatically reduce a name to a core representation which can be used in record linkage. The reduction is based on training examples of name variation in historical archives. On the conceptual level, the training procedure provides a data-driven foundation for record linkage that is missing from plain edit distance computation.

The construction of the core representations used for training is described in Section II. Related work is discussed in Section III. The method uses an algorithm for computation of Longest Common Subsequences for a set of strings which is described in Section IV. In Section V the training method and the set of features used to construct the model for the reduction algorithm are described. The record linkage method using the core representations resulting from the reduction algorithm is presented in Section VI. An evaluation of the linkage results is provided in Section VII, and Section VIII concludes.

II. Core representations

Spelling variation is very common for person names in European languages. The number of variants per name in large databases can range from only 2 or 3 to 50 or more for high frequent names. In this paper a training set containing 65,002 manually constructed Dutch name-variant pairs is used [1]. As an example, this set contains 73 variants of the Dutch female first name *Aaltje*, such as *Aaeltien, Aal, Aalie, Altje, Aaltgijn, Aaltjen, Aeltina, Aeltje, Aaeltjen, Alina, Altijen*. The sequence of characters that a name and its variants have in common, known as the Longest Common Subsequence or LCS, can be used as a core representation of this name. The 73 variants of the name *Aaltje* result in the LCS *Al*.

Ideally, the core of a name and its variants should be unique to prevent overgeneralization. Figure 1 shows the proportion of unique cores by length. In total 7.8% of all cores constructed from the training set is non-unique. This percentage is higher for shorter lengths, but the core remains discriminative in most cases. Moreover, a record generally contains multiple names. The sequence of cores is unique for virtually all name combinations (see Table VII for statistics).

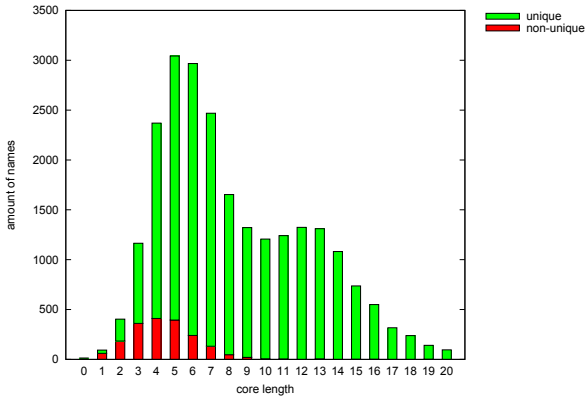

Fig. 1. Distribution of core length as stacked bar graph

III. Related work

A name core is essentially a key to identify a certain name and its variants. Various key extraction algorithms have previously been used in record linkage, for example phonetic keys like Soundex and Double Metaphone (see, e.g., [2]), or more syntactically oriented approaches such as suffix keys [3] or skeleton and omission keys [4]. However, these methods are based on general assumptions about word identity and word similarity, whereas the method presented in this paper constructs a key for a name based on automatically derived specific patterns as observed in training examples.

978-1-4673-2679-7/12/$31.00 ©2012 IEEE

Record linkage based on LCS (or the closely related concept of longest common substring) has been performed previously [5], [6]. Record linkage generally consists of two steps: selection of potential link pairs and comparison of selected pairs using a similarity measure. In the existing approaches, LCS is used as a similarity measure only. The current approach constructs an LCS-like key for a single record, which is used both for selection and as similarity measure.

The Longest Common Subsequence (LCS) problem for two strings a and b can be addressed using dynamic programming in $O(mn)$ time, with n and m denoting the length of string a and b, respectively [7]. Several other solutions exist with various complexity bounds, e.g., $O(p\ell + \ell \log \ell)$ or $O(p(m+1-p)\log n)$ [8], $O(n(m-p))$ [9]), or $O(r \log s)$ [10]. In these formulas p, ℓ and s denote the length of the LCS, $\max(m,n)$ and $\min(m,n)$, respectively, and r depends on the character distributions in a and b. These solutions can be very useful under certain circumstances, e.g., for small alphabets (DNA) or for a high degree of overlap (text file comparison). The problem can be extended to a set of strings, also called *multiple alignment*. The complexity of dynamic programming-based approaches becomes $O(n^k)$ in this case, for string length n and k elements in the set. The alternative solutions do not always require exponential time, however a full analysis is beyond the scope of this paper.

IV. LCS COMPUTATION

All of the previous bounds correspond to the worst-case time complexity of computing a LCS. However, for name variants a greedy backtracking algorithm can be applied with linear behaviour in practical cases. Algorithm 1 implements a straightforward exponential time method that generates all common subsequences for a string *name* and a set of *variants* of this name and returns the longest. The variable *lcs* is the common subsequence found so far (initialized as the empty string), *pos* represents the current position in *name*.

Algorithm 1 string LCSGREEDY(*lcs, pos*)

```
if pos > name.size then
  return lcs
else
  lcs_skip ← LCSGREEDY(lcs, pos+1)
  character ch ← name[pos]
  for all v ∈ variants do
    i ← (position of last match of lcs in v) + 1
    find ch in v starting from i
  if ch found in all v ∈ variants then
    lcs_match ← LCSGREEDY(lcs + ch, pos + 1)
  return LONGEST(lcs_match, lcs_skip)
```

This algorithm can be improved by executing the branching step that computes lcs_{skip} on demand only. If the remaining part of the string *name* is a subsequence of the remaining part of each variant, then branching is not necessary to find the LCS. Alternatively, a character in the remaining part of *name* may not be present in the remaining part of some variant. In this case lcs_{skip} needs to be computed for this character. Additionally, the algorithm checks whether this character has been discarded from the current variant in an earlier matching step (effectively preventing the later match). If this is the case, the earlier matching step is retracted and the algorithm proceeds to compute lcs_{skip} from that point.

When applied to the name variant data set, the median value (which effectively discards outlier values) of the number of steps needed to compute the LCS for a name is approximately 2.1 times larger than the length of the name (corrected for number of variants). Therefore, this dataset provides empirical evidence for the average case behaviour of the algorithm.

Current character	a	l	i	n	a
Position of character within word	0	1	2	3	4
Word length	5	5	5	5	5
Number of syllables in word	3	3	3	3	3
Part of syllable	2	1	2	1	2
Previous character	#	a	l	i	n
Next character	l	i	n	a	#
Distance to end of word	4	3	2	1	0
Included in LCS (class)	1	1	0	0	0

TABLE I
CLASSIFICATION FEATURES WITH EXAMPLE VECTORS

V. CLASSIFICATION

The name variants in the training set can be used as a look-up table for record linkage. However, a look-up approach is restricted to names that are present in the training set. To overcome this restriction, a model can be constructed that generates a core representation for arbitrary names. The longest common subsequences in the training data are used to construct this model. For each character in a name the model predicts whether it is included in the core representation or not. The features, listed in Table I, are therefore defined at character level. Some redundancy is present in the features to facilitate convergence of the training algorithm. Consider for example the name *Alina* with the associated LCS *Al* (see Section II). The first character is `a`, at position 0, word length 5, 3 syllables, second part of syllable, previous character is blank (#), next character `l`, distance to end of word is 4. This character is part of the LCS, therefore the class is 1. Table I shows all 5 training examples constructed from this name.

A. Syllabification

For two features the names need to be split into syllables. A syllable is a part of a word, typically consisting of a vowel, called the nucleus (2 in Table I), and the consonants preceding and following the vowel, called the onset (1) and the coda (3), respectively [11]. The nucleus is always present, while the onset and coda can be empty. Near-perfect identification of syllable borders can be performed using machine learning [12]. However, in the absence of syllable training data a simple rule-based procedure is sufficient for the present purposes. The name core training procedure is performed to derive patterns

from data, and patterns based on incorrect syllabification are still useful as long as the errors in syllabification are made in a consistent way. The implemented procedure (cf. [13]) parses a name from left to right. All consonants at the start of the string, if any, are added to the onset of the first syllable. The following (possibly multi-character) vowel is the nucleus of the syllable. Consonants following the vowel are added to the coda, except for the last consonant preceding the next vowel which is considered to be the onset of the next syllable.

B. Training

Preliminary classification experiments have been performed using a Naive Bayes classifier, a Bayesian network, a Support Vector Machine, a 1-nearest neighbor classifier and a C4.5 decision tree. The decision tree was selected based on accuracy and classification efficiency. This section describes the training process for the decision tree that implements the name reduction method. For categorical variables (current, previous and next character) the binary split algorithm of Breiman [14] is used to improve the efficiency of tree construction. The name variant data set [1] contains 65,002 names with a total of 554,450 characters with associated feature vectors. The tree is evaluated using 80% of the characters for training and the remaining 20% for testing (see Table II). The accuracy of the classification is 0.80, which is not perfect but a significant improvement over the prior class probabilities. For the majority of misclassified test examples the corresponding training examples display conflicting class labels, indicating noise in the data. However, the classification accuracy is sufficient to perform record linkage (see Section VI). The tree

Class distribution of training examples		
	amount	proportion
in core	303,996	0.69
not in core	139,562	0.31
Classification of test examples		
actual / *predicted*	in core	not in core
in core	68,104	8,033
not in core	13,738	21,015
accuracy: $(TP + TN)$/all examples		0.80

TABLE II
STATISTICS CLASSIFICATION TREE

construction algorithm creates leaf nodes based on an absolute impurity threshold, an impurity improvement threshold relative to the previous node, and a threshold on the number of instances in the node. These thresholds are intended to reduce the size of the tree, which results in less overfitting and improved efficiency in learning and classification. However, the number of training instances that belong to the minority class of a leaf node also serves as a quality measure of the classification made using this node (see Definition 1). The tree classifies each character of a name as present or absent in the core representation. Using the quality of the leaf nodes, the length of the generated core representation can be varied from permissive (including characters from lower quality leaf nodes) to strict (using only high-quality leaf nodes). Table III provides an example of the quality thresholds for the name *Geertien*. The eight characters in this name (with associated feature vectors) are pushed down the tree into eight leaf nodes. The lowest percentage of training examples with class 1 (3%) is found in the leaf node corresponding to the character `i` in *Geertien*. This means that if the leaf quality threshold is set to 0.03, the full name is generated as a core representation. For a threshold of 0.5 three leaf nodes remain with at least 50% training examples of class 1, resulting in the core `ger`.

								threshold	core
g	e	e	r	t	i	e	n	.03	geertien
g	e	e	r	t		e	n	.09	geerten
g	e	e	r	t		e		.29	geerte
g	e		r	t		e		.30	gerte
g	e		r	t				.47	gert
g	e		r					.70	ger
g	e							.83	ge
	e							.93	e

TABLE III
LEAF NODE QUALITY FOR THE NAME *Geertien*

Definition 1: The *quality* of a leaf node in the classification tree is defined as the proportion of positive training examples in the node. A leaf node returns class 1 if and only if the quality is above a given *threshold*.

VI. RECORD LINKAGE

The data used in the experiments is part of the Dutch Genlias database[1]. Genlias contains historical civil records from the 19th and early 20th century. In the current experiment marriage certificates are used. A marriage certificate contains the names of the bride and groom, and the parents of the bride and groom. The certificates containing couples mentioned as parents are linked to the certificates where these couples are mentioned as bride and groom. A *record* is defined as a complete certificate containing three couples. Figure 2 shows an example link between two records from the Genlias data set. The figure illustrates that a link is defined between certificates, primarily based on name similarity between couples.

The basic method of record linkage using core representations is straightforward: two records with the same sequence of core representations are considered as a match. The rest of this section describes heuristics to find additional matches.

The decision tree can be applied twice, first on the original name (single core) and again on the resulting core representation (double core). To compute the double core, a new feature vector is constructed for each of the characters in the single core (see Definition 2). Apart from the character itself, this feature vector can be completely different from the original. This can result in the removal of additional characters, which in turn can lead to additional matches. An example is provided in Table IV. This example contains two names with a variant:

[1] http://www.genlias.nl/en

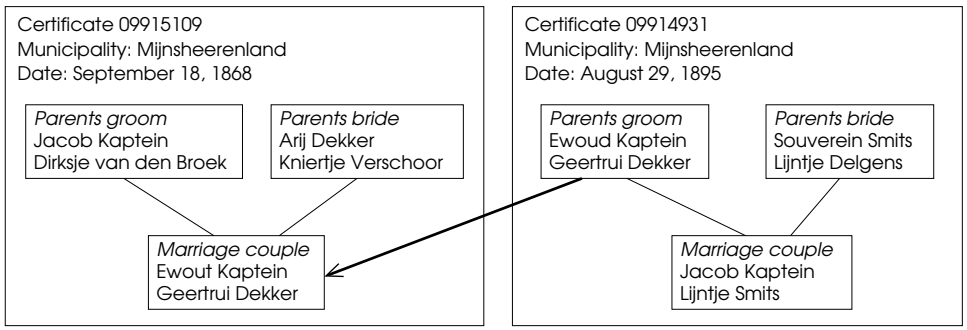

Fig. 2. Example of a link between two Genlias marriage certificates. The couple on which this link is based contains a small spelling variation in a first name: *Ewout* has changed into *Ewoud*.

Harm-Harmen and *Janneke-Jannetje*. For the first name, the single core extraction is sufficient: both variants are assigned the core `hr`. For the second name, the double core extraction is necessary to provide the core `jan` for *Janneke*. The core algorithm is trained on full names, therefore the second pass of the algorithm will target cores that morphologically resemble names (such as `janek`).

Definition 2: A *single core* of a name is generated by application of the trained decision tree on the characters of the name using feature vectors computed on the full name. A *double core* of a name is generated by application of the trained decision tree on the characters of the single core of the name using feature vectors computed on the single core.

record	*single core*	*double core*
Harmen van Buiten Janneke van der Kolk	hr bute janek kolk	hr but jan kolk
Harm van Buiten Jannetje van der Kolk	hr bute jan kolk	hr but jan kolk

TABLE IV
EXAMPLE OF DOUBLE CORE EXTRACTION

Another improvement can be obtained by using the inherent alphabetical order of core sequences. The basic method performs an exact match of the sequence of cores for a record. If no other record is found with the same sequence, a heuristic can be applied that selects the surrounding records using alphabetical order (similar to the record selection technique in [3]). The names can be rotated to avoid missing links with a spelling variation in the first part of a name sequence. The concept of rotation is described as follows:

Definition 3: A *rotation* of a core sequence is obtained by shifting the position of the elements of the sequence from right to left. The relative order of the sequence is preserved. The first element is placed at the end of the sequence.

The accuracy of the rotation heuristic can be improved by selecting only records that match in at least two rotations. This procedure is illustrated with a record containing a couple *Gerrit Engbringhof* and *Frouwkje van der Meulen* (Table V). This couple is used to construct the original sequence and three rotation sequences. For all other couples the core sequences (with rotations) are also computed and alphabetically ordered. For every rotation of the core sequence of the first couple the alphabetically preceding and succeeding sequences from the list of couple core sequences are extracted as possible matches. Two sequences are encountered twice

rotation	core sequence
preceding	`grt|engbnhof|ant|bonstra`
original	`grt|engbnhof|fr|mul`
succeeding	`grt|engbrhof|fru|mul`
preceding	`engbnhof|ant|bonstra|grt`
rotation 1	`engbnhof|fr|mul|grt`
succeeding	`engbnhof|r|schepe|mrte`
preceding	`fr|mul|gesk|gorhui`
rotation 2	`fr|mul|grt|engbnhof`
succeeding	`fr|mul|huber|bierman`
preceding	`mul|grt|elzng|pter`
rotation 3	`mul|grt|engbnhof|fr`
succeeding	`mul|grt|engbrhof|fru`

TABLE V
EXAMPLE OF CORE ROTATIONS

in different rotations, `grt|engbnhof|ant|bonstra` and `grt|engbrhof|fru|mul`. The first sequence is based on a match with the original couple sequence and another match with rotation 1. These two couple sequences both start with a core from the same person (*Gerrit Engbringhof*), either the core of the first name (`grt`) in the original sequence or the core of the family name (`engbnhof`) in rotation 1. A single person match is likely to be incorrect, therefore this link is rejected. The sequence `grt|engbrhof|fru|mul` is based on a match with the original sequence and another match with rotation 3. These two couple sequences start with a core from different people, therefore this match is accepted. The sequences correspond to the couple *Gerrit Engbrenghof, Froukje van der Meulen*, which is indeed a valid match.

Note that the cores in this match are different for both the last name of the first person (`engbnhof` vs. `engbrhof`) and the first name of the second person (`fr` vs. `fru`). This indicates that the core selection mechanism does not perform well for these names, however the rotation heuristic is capable of solving this issue in many cases (see Section VII).

An alternative for this method is to select records with

three cores in common (instead of all four cores as in the basic algorithm). This is sufficient for a large number of cases, but the current method has several advantages. First, not all matching records have three cores in common, like in the example described in this section. In that case two cores can be used, but this leads to a large number of incorrect matches. Second, this method uses the information from the non-matching cores by exploiting the alphabetical order of the full core sequence. In the example the core `engbnhof` does not match, but the alphabetical order provides the core `engbrhof` which is part of the correct match. When simply selecting three matching cores, the information from the fourth core is lost. Third, this method is efficient to implement using a binary search tree that provides the preceding and succeeding records in constant time.

A. Bootstrapping

The record linkage step results in variant combinations that partly are not present in the original training data. In a bootstrapping step these new variant combinations are used as additional training data for the decision tree. The original training data was biased towards the positive class (0.69), the additional examples are selected to reduce this bias to 0.53. The classification accuracy after bootstrapping does not significantly change, however the linkage results are improved (see Section VII).

VII. Evaluation

The record linkage problem is centered around two main aspects: linkage quality and scalability. Both aspects are essential for practical applicability of a linkage procedure. From a data mining point of view scalability is of key interest, while link quality is important on the linguistic level. The time complexity of computing the core sequence for a record is linear in the length of the string. The complexity of comparing the core sequence to the database is logarithmic in the number of records, therefore the method can be easily scaled up to larger databases. The linking phase for the current data set takes 24.5 minutes for 5.2 million records (2.6 million certificates which each contain two parent couples that can be linked), or around 3500 records per second. The linking phase is preceded by constructing the model (67 seconds for around 0.5 million training examples) and computing the reference core sequences (14.5 minutes). Experiments have been performed on a 3.16 GHz dual core CPU with 6GB memory using 64-bit Linux. All programs are written in C++.

True Positive (TP)	*correct match*
True Negative (TN)	*no match found, match does not exist*
False Positive (FP)	*incorrect match*
False Negative (FN)	*no match found, match does exist*
Precision	$\frac{TP}{TP+FP}$
Recall	$\frac{TP}{TP+FN}$
F-measure	$2 \cdot \frac{Precision \cdot Recall}{Precision+Recall}$

TABLE VI
Evaluation measures

The quality of record linkage methods can be evaluated by measuring precision and recall (see Table VI for definitions). The correct links are not known, and due to missing records also the amount of links is unclear. For evaluation purposes a selection of the Genlias data has been made that is known to contain a relatively small number of missing records in order to provide a realistic estimate of the recall of the method (see Table VII). Links with a Levenshtein edit distance (lv) of 4 or higher between the records in the match are manually evaluated. Links with $lv \leq 3$ are assumed to be correct by default. To test this claim, as well as to evaluate recall, a full manual verification on all matches is provided for a sample of 6212 records from a small town (Table VIII). The matches are computed using threshold 0.6 after bootstrapping. A number of indexing methods as described in a recent overview article [15] has been applied to the verification set for comparison. These methods are more oriented towards recall and efficiency, which is common in record linkage. The current method is more precision-oriented, however the overall quality is competitive. This holds also for scalability (not shown in Table VIII).

Indeed only a single match with a small edit distance is considered incorrect after verification. The maximum edit distance of 3 ranges over the full string which consists of four names, therefore the distance threshold is relatively strict. Additionally, linking on a combination of names prevents potentially incorrect matches. Consider for example the names *Jack* and *John*, which have edit distance 3 and therefore could be part of an incorrect match. However, in order for a match with maximum edit distance 3 to occur, the remaining three names in both records need to be equal which is improbable given the name distribution commonly found in data.

In Table VII cumulative matching rates are listed for the categories exact match, standard core, double core and rotations. The matching rate for the single and double core matching increases for higher thresholds, because a higher threshold implies a shorter core sequence with less variation. Conversely, the number of rotation matches decreases for higher thresholds. Rotations are intended as a repair mechanism in case standard core matching fails, therefore the decrease can be explained by a smaller need for repairs for higher thresholds. Bootstrapping has been performed for threshold 0.6 only, as a proof of concept. This results in a total matching rate of 0.899 for this threshold. Except for exact matching, all matching categories can produce matches with a large difference $(lv \geq 4)$ between the records. The proportion of large difference matches over all records is listed in Table VII. The table also shows that the generated core sequence is unique for virtually all target records, indicating that core extraction preserves the distinction between different records.

VIII. Conclusion and future work

The algorithm from Section V can be used in record linkage with practical recall and precision properties in a computationally efficient way. Because the model performs normalization

	threshold			
category	0.4	0.5	0.6	0.7
exact match	0.692	0.692	0.692	0.692
standard core	0.758	0.767	0.773	0.790
double core	0.766	0.784	0.787	0.812
rotations	0.848	0.861	0.857	0.864
bootstrapping	n/a	n/a	0.899	n/a
large difference	0.008	0.008	0.008	0.008
correct	70.5%	75.2%	70.7%	67.2%
duplicate core sequences	0.0007	0.0009	0.0014	0.0021

TABLE VII
RESULTS OF THE NAME CORE ALGORITHM FOR DIFFERENT DECISION TREE THRESHOLDS

	all matches	$1 \leq lv \leq 3$	$lv \geq 4$
TP	5656	1302	12
TN	50	50	50
FP	5	1	4
FN	440	377	68
Precision	0.99	0.99	0.75
Recall	0.93	0.78	0.15
F-measure	0.96	0.88	0.26

measure	1	2	3	4	5	6
Traditional blocking	1.00	0.79	0.89	1.00	0.15	0.26
q-grams	0.99	0.91	0.95	0.97	0.37	0.51
Suffix array	0.36	0.84	0.48	0.13	0.20	0.14
Suffix array substring	0.15	0.97	0.24	0.09	0.60	0.15
Suffix array robust	0.31	0.93	0.45	0.18	0.44	0.23
Sorted array	0.36	0.92	0.47	0.21	0.54	0.27
Sorted array adaptive	0.80	0.90	0.84	0.63	0.41	0.47
th-Canopy clustering	0.99	0.86	0.92	0.98	0.30	0.44
nn-Canopy clustering	0.13	0.93	0.22	0.06	0.47	0.11
Sorted inverted index	0.14	0.98	0.24	0.08	0.63	0.14
th-String map	0.45	0.83	0.49	0.28	0.22	0.15
nn-String map	0.48	0.82	0.54	0.27	0.20	0.16

TABLE VIII
RESULTS ON EVALUATION SET. PRECISION, RECALL AND F-MEASURE ARE LISTED FOR MATCHES WITH $1 \leq lv \leq 3$ (MEASURES 1–3, RESPECTIVELY) AND MATCHES WITH $lv \geq 4$ (MEASURES 4–6, RESPECTIVELY). FOR DEFINITIONS AND PARAMETERS OF COMPARISON METHODS SEE [15].

on individual records, there is no trade-off between computational efficiency and recall. The method produces a substantial number of links with high edit distance, which is desirable for any record linkage procedure. The accuracy of the method can be attributed to the fact that a comparison of cores is an informed way of performing edit distance. The core of a name represents the elements that are actually important for the identity of that name, based on training data. This provides a conceptual foundation for the method, unlike standard edit distance where all characters are considered equal. There have been various extensions and adaptations of edit distance to account for aspects of relative importance of characters, such as Soundex (position and grouping of characters), Jaro-Winkler distance (prefix matching), or weighted Levenshtein distance (learn common edit operations from data). The current work is an attempt to build a model that can cover all these aspects, and deduce from data what the most important aspects of strings (in this case names) actually are.

The contribution of this work consists of three aspects: a novel, morphologically motivated model of name variation; computational efficiency and high recall in discovering links with small edit distance; and additional discovery of a significant amount of links with large edit distance within practical levels of precision.

The method has been evaluated on the domain of historical archives in the Netherlands. However, the method itself is not restricted to the Dutch language or to historical data. Provided that training data on name variation is available, the method can be applied to various other domains.

In future work the training data can be chosen to be more specific and the feature set can be expanded in order to improve precision and recall. Bootstrapping can be developed to increase the use of information contained in the data.

ACKNOWLEDGMENT

This work is part of the research programme LINKS, which is financed by the Netherlands Organisation for Scientific Research (NWO), grant 640.004.804.

REFERENCES

[1] R. Alma, "Thesauri of standardized personal names in Drenthe," Personal communication. Data set supplied by Drents Archief, http://www.drentsarchief.nl, 2011.

[2] P. Christen, "A comparison of personal name matching: Techniques and practical issues," in *Proceedings of the Sixth IEEE International Conference on Data Mining — Workshops*. IEEE Computer Society, 2006, pp. 290–294.

[3] T. de Vries, H. Ke, S. Chawla, and P. Christen, "Robust record linkage blocking using suffix arrays," in *CIKM '09: Proceedings of the 18th ACM Conference on Information and Knowledge Management*. ACM, 2009, pp. 305–314.

[4] J. Pollock and A. Zamora, "Automatic spelling correction in scientific and scholarly text," *Communications of the ACM*, vol. 27, pp. 358–368, 1984.

[5] C. Friedman and R. Sideli, "Tolerating spelling errors during patient validation," *Computers and Biomedical Research*, vol. 25, pp. 486–509, 1992.

[6] M. Ektefa, F. Sidi, H. Ibrahim, M. Jabar, and S. Memar, "A comparative study in classification techniques for unsupervised record linkage model," *Journal of Computer Science*, vol. 7, pp. 341–347, 2011.

[7] D. Hirschberg, "A linear space algorithm for computing maximal common subsequences," *Communications of the ACM*, vol. 18, pp. 341–343, 1975.

[8] ——, "Algorithms for the longest common subsequence problem," *Journal of the ACM*, vol. 24, pp. 664–675, 1977.

[9] S. Kumar and C. Rangan, "A linear space algorithm for the LCS problem," *Acta Informatica*, vol. 24, pp. 353–362, 1987.

[10] G. Jacobson and K.-P. Vo, "Heaviest increasing/common subsequence problems," in *Combinatorial Pattern Matching*, ser. Lecture Notes in Computer Science. Springer, 1992, vol. 644, pp. 52–66.

[11] S. Bartlett, G. Kondrak, and C. Cherry, "Automatic syllabification with structured SVMs for letter-to-phoneme conversion," in *Proceedings of ACL-08: HLT*. ACL, 2008, pp. 568–576.

[12] G. Bouma, "Finite state methods for hyphenation," *Natural Language Engineering*, vol. 9, no. 01, pp. 5–20, 2003.

[13] M. Hammond, "Optimality theory and prosody," in *Optimality Theory: An Overview*. Blackwell Publishers, 1997, pp. 33–58.

[14] L. Breiman, *Classification and regression trees*, ser. The Wadsworth and Brooks-Cole statistics-probability series. Chapman & Hall, 1984.

[15] P. Christen, "A survey of indexing techniques for scalable record linkage and deduplication," *Transactions on Knowledge and Data Engineering*, vol. 24, pp. 1537–1555, 2012.