A String Metric Based on a One-to-one Greedy Matching Algorithm

Horacio Camacho and Abdellah Salhi

The University of Essex, Colchester CO43SQ, U.K.
{jhcama, as} @essex.ac.uk

Abstract. We introduce a novel string similarity metric based on a matching problem formulation. This formulation combined with other heuristics generates comparatively more accurate string similarity scores than some other methods. The results of the proposed method are improved by training the method on domain data. A detailed description of the method as well as computational results on many databases are given.

1 Introduction

Automatic methods for duplicate record detection such as record linkage, [1], merge/purge, [2], duplicate detection, [3], and Hardening, [4], among others, have been suggested for many years now. Although different in concept, they all require, in one form or other, the use of string similarity metrics, in order to decide if two records are similar enough to be considered as duplicates.

String similarity metrics can be roughly divided into three general groups [5]: Token-based metrics, character-based metrics and hybrid metrics. The token-based metrics, of which Jaccard, [6], Cosine and TFIDF, [7], are members, consider strings as "bags of words", [7]. Character-based metrics such as the Jaro metric, [8], and its variants, count the number of similar characters in a pair of strings. Edit metrics, such as the Levenshtein, [9], and its variants, count the number of character-level operations (delete, insert, substitute) required to transform one string into another treating the string as a sequence of characters. Hybrid metrics combine both the token-based and the character-based metrics. In a hybrid metric, a token-based metric uses scores obtained by a character-based metric. Common examples of hybrid metrics are SoftTFIDF [5], and the metric due to Monge and Elkan [3], also known as Level2 method. For a good survey of string metrics, the reader is advised to consult [5]. There, a comparison between several string metrics has been carried out and SoftTFIDF performed best on average.

Because the results from using individual metrics often lack consistency, techniques such as the Support Vector Machine (SVM) approach, [10], that combines results from different metrics, has been introduced. This regressional type approach may be limited in its applicability due to computational costs particularly when the number of participating metrics is high and the input databases

© A. Gelbukh, S. Torres, I. López (Eds.)
Advances in Computer Science and Engineering
Research in Computing Science 19, 2006, pp. 171-182

are large. The consistency issue spawned other approaches such as those which rely on training. In [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], and [21] trained techniques have been suggested. Because parameter tuning is done according to what domain the input database is concerned with, consistency in performance is therefore enhanced. So far, however, trained techniques are mainly of the character-based type.

Here we suggest a novel metric for string similarity. The approach is hybrid in nature; it combines a character-based metric and a token-based metric, both explained later. Moreover, it can also be trained for a given domain. The training procedure will be explained later too. Both the non-trained and the trained versions of it are compared on a set of databases with several approaches as can be found in SecondString, [5], and Simmetrics Java toolkits, [22]. The non-trained version performs consistently well in all experiments. But, the trained version performs better and compares favorably with all metrics considered.

In the next section we provide a motivation for this metric. In section 3 we formalize the presented ideas into a model. In section 4 we explain the string metric. In section 5 we illustrate the method with an example. In section 6 we define a method to estimate the parameters of the proposed string metric. Section 7 contains comparison results and section 8, the conclusion.

2 Motivation

Consider a pair of strings T and U, which, through tokenisation, we break into substrings (tokens) as: $T = \{T_1, T_2, ..., T_I\}$ and $U = \{U_1, U_2, ..., U_J\}$. A character-based method calculates all similarity scores $s_{ij}(T_i, U_j)$ between token T_i and token U_j for $i = 1, ..., I$ and $j = 1, ..., J$. Based on these scores, a token-based method selects the most adequate token pairs in order to compute the similarity score $s(T, U)$ of strings T and U. Both scores $s_{ij}(T_i, U_j)$ and $s(T, U)$ take positive and possibly zero values with the large values corresponding to good matches and low values to (potentially) non-matches in both character and token comparisons.

Ideally, similarity metrics (hybrid or otherwise) must be consistent. In other words, the similarity of similar or near similar strings returned must be high and that of "not so similar strings" must be low in comparison. Unfortunately, this is often not the case and the reason may well lie with the way the token similarity is calculated.

Consider the two most common hybrid methods: SoftTFIDF and Level2. SoftTFIDF defines $CLOSE(\theta, T, U)$ as a triplet containing strings T and U and a scalar θ such that for any token T_i included, there is some token U_j such that $s_{ij}(T_i, U_j) > \theta$, and s_{ij} is a similarity score from a character-based string metric, such as Jaro-Winkler, [23]. In contrast, the Level2 method considers the complete set of tokens in T, and then chooses as similar to each token T_i, those tokens U_j from U having: $\max_{j=1}^{J} s_{ij}(T_i, U_j)$ where s_{ij} is a similarity score from a character-based string metric, such as a variant of the Levenshtein method due to Monge and Elkan, [23].

To illustrate what we have just said, we consider the following example: $T =$ {John, Johnson}, $U =$ {Johns, Charleston}, $V =$ {J., Charletson}. SoftTFIDF computes $s(T,U) = 0.95$ and $s(U,V) = 0.49$. Level2 computes $s(T,U) = 1.0$ and $s(U,V) = 0.84$. While one might almost be certain that T and U are not pointing to the same real object, and there is more evidence that U and V are more similar than T and U, those methods score $s(T,U)$ higher since they consider the same token U_k if all tokens in T are very similar and each token score $s_{1k}, s_{2k}, ..., s_{Ik}$ is very close to 1.

We consider the most appropriate token pairs in a different manner. We define the most appropriate token pairs as a one-to-one matching setting, similar to the following assignment problem:

$$\max z = \sum_{i,j} s_{ij} x_{ij} \tag{1}$$

$$\sum_{i} x_{ij} = 1, \forall j \text{ and } \sum_{j} x_{ij} = 1, \forall i \tag{2}$$

$$i = 1, ..., I, j = 1, ..., J, s_{ij} \in R^{+}, x_{ij} \in \{0,1\}$$

where x_{ij}, is a binary variable which takes value 1 if token T_i and token U_j are considered as a match, and 0 otherwise, but with the difference that here we define matched pairs as the set of highest scored pairs of tokens. In the next section we formalize our method. We also provide an example that explains some drawbacks of the assignment model.

3 Formalization

A one-to-one match between tokens in T and tokens in U implies that the assignment constraints (2) are satisfied. Note that the set of matching pairs we look for are potentially different from those returned when the assignment problem is solved. While the assignment problem maximizes the function (1), subject to restrictions (2), we instead suggest the following procedure:

Algorithm 1: Select a maximum score value $s_{kl} = \max_{ij} s_{ij}$ such that token T_k and token U_l have not been found to match another string yet, or they satisfy: $\sum_i x_{il} = 0$ and $\sum_j x_{kj} = 0$. The token pair (T_k, U_l) is then considered as a matching pair, thus setting $x_{kl} = 1$. This procedure is repeated until restrictions (2) are completely satisfied.

An alternative way to find the set of matching pairs, is by sorting the set of scores s_{ij} in descending order. The pair with maximum score is chosen to be part of the overall match. The process is repeated bearing in mind that restrictions (2) must be satisfied at all time. This process is not computationally expensive and amounts mostly to the cost of a sorting algorithm. In fact it can be reduced further by removing from the subsequent list of scores those which

do not correspond to pairs satisfying the restrictions. These are found trivially since all pairs in the row and column of the chosen pairs so far are barred from being chosen by the assignment constraints. This consideration reduces the work substantially.

This differs from the assignment problem in that we always choose those token pairs with a score which is as high as possible for a match, when there is no clear match. For instance, consider the example of Table 1.

Table 1. Example of two similar strings: T and U

String T	Strng U
Gerardo F. Jolmes Dorado	Francisco D. Jolmes Dorado

Let: $s_{14} = 0.6, s_{21} = 0.6, s_{33} = 1.0, s_{42} = 0.6, s_{44} = 1.0$, and all the other scores are set to 0. The solution to the assignment problem is $\{x_{14}, x_{21}, x_{33}, x_{42}\}$ corresponding to the matching pairs {(Gerardo, Dorado), (F., Francisco), (Jolmes, Jolmes), (Dorado, D.)}. Since s_{44} (Dorado, Dorado) has a higher score than s_{42} (Dorado, D.), then the better matching pairs would be {(F., Francisco), (Jolmes, Jolmes), (Dorado, Dorado)} corresponding to the solution $\{x_{21}, x_{33}, x_{44}\}$. This, however, has a lower assignment objective ($z = 2.6$) in (1) than that of the solution returned for the assignment problem ($z = 2.8$). However, we do believe, and this is backed by our results, that in fact this is a better way to settle the matching problem between tokenised strings, in particular when there is no perfect match between the strings. This situation occurs often, since in most databases the ratio of redundant to non-redundant records is more likely to be low than high. It is important to add that when this is not the case then the solution advocated here will return a similar result to that of the solution to the assignment problem.

4 The Hybrid Method

The method about to be suggested is hybrid in nature. In the following we introduce the two aspects of it, the character and the token. As we shall see, some techniques used in the past have also been implemented here, but also some improvements have been introduced. Even though each of the character and the token based metric are defined differently, both apply the same way of assigning pairs (characters and tokens) as defined in Algorithm 1.

4.1 Character-based Similarity Score

Let each token T_i and U_j, $i = 1, ..., I$ and $j = 1, ..., J$, of a pair of strings (T, U), be broken into a set of characters $T_i = \{T_i^1, T_i^2, ..., T_i^F\}$ and $U_j = \{U_j^1, U_j^2, ..., U_j^G\}$. Let $\delta\left(T_i^f, U_j^g\right)$, $f = 1, ..., F$, and $g = 1, ..., G$, be a function equal to 1 if characters: $T_i^f = U_j^g$, and 0 otherwise.

Same characters can be found in non similar tokens, for example, from Table 1 the pair of tokens: (Gerardo, Dorado) share the same characters "a", "r", "d" and "o", but the positions of "a" and "r" in this example are making an important difference. In order to account for the order in position of characters, we introduce the following function:

$$d_{ij}^{fg} = \begin{cases} \delta\left(T_i^f, U_j^g\right) - \frac{1}{\gamma}|f-g|, \text{ if } |f-g| < \gamma \text{ and } \delta\left(T_i^f, U_j^g\right) = 1 \\ 0, \text{ otherwise} \end{cases} \quad (3)$$

where γ is a constant which penalizes the difference between the positions of matching pairs by the quantity: $-\frac{1}{\gamma}|f-g|$ and at the same time restricts the number of positions where a possible match might be found, if for example, $\gamma = 3$, and if $|f-g| > 3$, then $d_{ij}^{fg} = 0$. This condition also speeds up the computation since no characters are worth exploring when position is bigger than: $|f-g| = 3$.

Given a set of similarity scores of characters d_{ij}^{fg}, by Algorithm 1 we can define a set of matching pairs of characters (T_i^f, U_j^g) by setting $s_{fg} = d_{ij}^{fg}$. Thus, we define our token score as:

$$s_{ij} = \sum_{fg} s_{fg} x_{fg}/(2F) + \sum_{fg} s_{fg} x_{fg}/(2G) \quad (4)$$

where x_{fg} is returned by Algorithm 1.

We can make further improvements to the proposed string metric value s_{ij} by implementing the Winkler scorer [23]. This heuristic has been applied in the past to the Jaro method and is defined as: $s'_{ij} = s_{ij} + prefix * prefixScale * (1 - s_{ij})$, where $prefix$ is the largest prefix which characters of T_i and U_j match, such that a prefix is no larger than 4, $prefixScale$ is a scaling factor which is meant to temper down the upwards adjusted score because of the shared prefix between the strings considered.

4.2 Token-based Similarity Score

Among the token based string metrics discussed in [5], the TFIDF was shown to have the best results. TFIDF [7] is a vector space approach widely used by the information retrieval community, and it has also been implemented in for the task of matching names such as SoftTFIDF. The latter variant has a very good record in name matching. We try here to replicate this success, through a modification of the basic method.

Recall that TF_{T_i} is the frequency of token T_i in T and IDF_{T_i} is the inverse frequency of token T_i in the current dataset or "corpus". Notice that TFIDF was initially designed for the task of searching documents, here, the searching task is limited to match short strings compound of only few tokens (see Table 1). For example, consider a document which describes the "history of computers", as it is expected, the token: "computer" will appear several times in the document,

in contrast, in our case, the token: "Gerardo" in string T of Table 1, appears only once. Notice that we are also not penalizing the positions of the tokens, as we did in the last section for the positions of the characters, since the number of tokens included into a string is relatively small and repetition of similar tokens is very rare in names.

Here we consider a different way to measure the contribution of the tokens. Instead of measuring the frequency of token T_i (TF-term) we measure a match rate term (MR-term), defined as:

$$MR_{ij} = c_{ij} / \min(|T|, |U|) \tag{5}$$

where c_{ij} is the number of matched characters of a pair of tokens T_i and U_j and $|T|$ and $|U|$ are the lengths of strings T and U, respectively. For example, from Table 1 in the pair of tokens: (Gerardo, Dorado) $c_{14} = 4$ ("a", "r", "d" and "o"), $|T| = 7$, $|U| = 6$ and $MR_{14} = 4/6$.

As in [5] we compute the IDF-term as follows: $IDF_{ij} = a_i a_j$, where: $a_i = b_i / \sqrt{\sum_i b_i^2}$, $b_i = \log(IDF_{T_i})$. We then define the set of token scores as:

$$d_{ij} = s'_{ij} \left[\frac{1}{2} IDF_{ij} + \frac{1}{2} MR_{ij} \right] \tag{6}$$

Given a set of similarity scores of tokens d_{ij}, by Algorithm 1 we can define a set of matching pairs of tokens (T_i, U_j) by setting $s_{ij} = d_{ij}$. Thus, we define our string similarity score as:

$$s = \sum_{ij} s_{ij} x_{ij}. \tag{7}$$

where x_{ij} is the solution from Algorithm 1.

5 Example

Consider the pair T = {Jhon, Johnson}, U = {Johns, Charleston}. We compute the character-based and token-based similarity scores as follows.

By Character-Based String Metric:

In order to compute the score of a token pair, say s_{11} of tokens: ("Jhon", "Johns"), we partition each token into the set of characterers: T_1 = {"j", "h", "o", "n"} and U_j = {"j", "o", "h", "n", "s"}. We arbitrarily set $\gamma = 3$, and the list of character scores d_{ij}^{fg} of (3) is then computed, for instance: $d_{11}^{11} = \delta$("j", "j") $- \frac{1}{3}|1 - 1| = 1$, $d_{11}^{12} = 0$, and repeat this process until all the character scores are computed. Once the list of scores d_{ij}^{fg} is computed, we apply Algorithm 1 in order to select the most appropriate character pairs. The output of Algorithm 1 is the set: $\{x_{11}, x_{23}, x_{32}, x_{44}\}$, and its corresponding scores are shown in Table 2, hence $\sum_{fg} s_{fg} x_{fg} = 3.33$ and $s_{11} = 3.33/8 + 3.33/10 = 0.75$ (see (4)). By the Winkler scorer, we set $prefixScale = 0.1$. Since the only prefix with perfect match is the first character pair ("j", "j"), $prefix = 1$. Thus: $s'_{11} = 0.75 + 0.1(1 - 0.75) = 0.775$.

By Token-Based String Metric:

We compute the scores of all pairs s'_{ij} in the same way as previously illustrated. In this particular case, the IDF term for each token T_i and U_j is the same, since the frequency of each token is equal to 1. The size of the corpus for this small example is 2, thus $IDF_{T_i} = IDF_{U_j} = \ln(2/1)$ and $a_i = a_j = IDF_{T_i}/\sqrt{\sum_i (IDF_{T_i})^2} = 0.707$. We compute the number of matched characters for a given pair of tokens c_{ij}, for instance: $c_{11} = 4$ ("j", "h", "o" and "n"), thus, MR-term is obtained, for instance MR_{11}=4/ min(11, 15) (see (5)). The list of scores, MR-terms and the set of scores d_{ij} for all pairs of tokens T_i and U_j are shown in Table 3. By Algorithm 1 we select the most appropriate pair of tokens. The output of Algorithm 1 is the set: $\{x_{21}, x_{12}\}$, hence $s = 0.436+0.052 = 0.488$.

Table 2. Character based string metric example of a pair of tokens: ("Jhon","Johns"). The remaining scores are equal to 0

T_1^f	U_1^g	s_{11}^{fg}
T_1^1	U_1^1	1
T_1^2	U_1^3	0.667
T_1^3	U_1^2	0.667
T_1^4	U_1^4	1

6 Parameter Estimation

It is possible to improve the performance of the proposed string metric if we set values of the function $\delta\left(T_i^f, U_j^g\right)$ and the transposition constant γ taking account of the domain of the data. In some cases, a pair of characters might have a chance to be matched if the characters are considered equivalent in the given domain. For example, the characters "-" and "/" might be considered to be equivalent if the data we intend to match a set of telephone numbers. We would then have δ("-", "/") = 1. In other cases similar characters like "e" and "c" might be considered to be similar if the information was extracted via OCR, thus δ("e", "c") = 1.

Given a training set of matching and non matching pairs, finding the equivalent character pairs by hand can be difficult, since the number of possible pairs to tune may be very large.

As in [20], we initially assume independence between matching characters, i.e. a matching character in a pair of tokens is independent of other matching or non matching pairs of characters. If we have n different characters in a given vocabulary and if $\delta\left(T_i^f, U_j^g\right) = \delta\left(U_j^g, T_i^f\right)$, there are is $\frac{n(n-1)}{2}$ combinations of different pairs of characters. If for each corresponding pair: $\delta\left(T_i^f, U_j^g\right)$ can be equal to 1 or 0, then the number increases to $n\,(n-1)$.

Table 3. Token based string metric example of the pair of strings ("Jhon","Johns")

T_i	U_j	s_{ij}	MR_{ij}	IDF_{ij}	d_{ij}
T_1	U_1	0.775	0.364	0.5	0.335
T_2	U_1	0.914	0.455	0.5	0.436
T_1	U_2	0.175	0.090	0.5	0.052
T_2	U_2	0.121	0.364	0.5	0.052

We can reduce the number of parameters if we define a candidate list of possible matching pairs of characters. Such candidate list can be defined by the algorithm described in [20], where given a set of matching pairs, the probabilities of edit operations of characters such as insertion, deletion and transposition are estimated by an *Expectation Maximization* algorithm, [20].

Once the probabilities are estimated, we ignore the insertion and deletion probabilities and we only consider substitution pairs whose probability is significantly bigger than, for instance, 1×10^{-3}.

Since the independence assumption between characters might not hold in all cases, we set the matching scores iteratively in a greedy manner. We test all the candidate pairs of characters and set as matching those pairs which best improve performance. We repeat this process until no further improvement is achieved. The transposition constant γ can also be iteratively set in the same manner. We consider eleven possible values of γ in the range $(0, 1)$.

The performance we measure in order to find the best parameter is given by the *non-interpolated average precision* as defined in [5]. Where N candidate pairs are ranked by score in a task of m matching pairs it is defined as $\frac{1}{m}\sum_{i=1}^{n}\frac{c(i)d(i)}{i}$, where $c(i)$ is the number of correct pairs before rank position i and $d(i)$ is equal to 1 if the actual pair is a match, or 0 otherwise.

7 Experimental Results

7.1 Implementation

The method, both in its trained and nontrained forms was implemented in Java. Source codes are available at: privatewww.essex.ac.uk/~jhcama/TagLink.htm. We also implemented the tokenizer provided in the SecondString package.

7.2 The Data

Experiments were performed on each of the datasets listed in Table 4 and 3 other datasets randomly generated by the UIS database generator, [2]. The UIS database generator creates sets of records which are randomly corrupted. The level of random corruptions per record, the total number of records to be generated and the number of redundant records to be included in the artificial database are preset.

Table 4. Experimental data, source [5]

Dataset	Records	Redundancies
BirdKunkel	337	38
BirdScott2	719	310
Census	841	671
Cora	923	902
Parks	654	505
Restaurant	863	228

Table 5. Experimental results. Non-interpolated average precision of proposed methods VS best 14 methods. The best methods for each dataset are marked with "*". UIS column is the average result of the 3 randomly generated datasets

String metric	BirdKunkel	BirdScott	Census	Cora	Parks	Restaurant	UIS
Suggested	0.939	0.977	0.447	0.908	0.937	0.939	0.956
Suggested trained	0.955*	0.985*	0.469	0.920*	0.980*	0.980*	0.992*
SoftTFIDF	0.526	0.936	0.410	0.911	0.937	0.963	0.956
TFIDF	0.740	0.970	0.107	0.911	0.922	0.964	0.927
S.W.Gotoh	0.902	0.735	0.354	0.873	0.914	0.645	0.944
UnsmoothedJS	0.808	0.969	0.117	0.865	0.833	0.787	0.927
JelinekMercerJS	0.800	0.968	0.122	0.848	0.816	0.763	0.926
Jaccard	0.691	0.953	0.117	0.876	0.825	0.804	0.928
SmithWaterman	0.903	0.564	0.371	0.871	0.913	0.598	0.943
DirichletJS	0.608	0.965	0.121	0.861	0.832	0.765	0.926
MongeElkan	0.910	0.766	0.263	0.784	0.906	0.471	0.920
OverlapCoefficient	0.530	0.959	0.107	0.764	0.780	0.834	0.856
QGramsDistance	0.020	0.786	0.325	0.869	0.902	0.693	0.952
Level2JaroWinkler	0.055	0.531	0.484*	0.783	0.873	0.687	0.907
DiceSimilarity	0.165	0.729	0.112	0.791	0.772	0.754	0.861
CharJaccard	0.016	0.499	0.377	0.628	0.887	0.630	0.878

7.3 Experimental Methodology

Having N candidate pairs ranked by score in a task of m matching pairs, we measure the *non-interpolated average precision* as mentioned in section 6. We also measure the *maximum F1*, as $\max_i F1(i)$, [5], where $F1(i) = \frac{2*p(i)*r(i)}{p(i)+r(i)}$ is the harmonic mean at rank position i, $p(i) = \frac{c(i)}{i}$ is called the precision at position i, $r(i) = \frac{c(i)}{m}$ is called the recall at position i, and $c(i)$ is the number of correct pairs before rank position i.

We compare our non-trained method against methods contained in the SecondString and Simmetrics packages. Since the number of similarity methods to be evaluated is large, and since the number of all possible string pairs is $O(l^2)$, where l is the size of the dataset, the computational time required for the evaluation can be excessive. In order to reduce it, we compute a similarity score by a cheap string metric for each string pair in the dataset. If the score is greater than a certain threshold, then the string pair is kept for further testing, otherwise it

Table 6. Average precision and average maximum F1 of proposed method VS 14 selected methods. Sources: 1=SecondString, 2=SimMetrics

String metric	Precision	F1	Time (min.)	Source
Suggested	0.872	0.887	5.771	-
Suggested trained	0.897	0.895	8.334	-
SoftTFIDF	0.806	0.811	6.408	1
TFIDF	0.791	0.788	1.496	1
SmithWatermanGotoh	0.767	0.789	330.072	2
UnsmoothedJS	0.758	0.777	1.361	1
JelinekMercerJS	0.749	0.773	1.393	1
Jaccard	0.742	0.757	1.306	1
SmithWaterman	0.738	0.769	8.890	2
DirichletJS	0.725	0.739	1.391	1
MongeElkan	0.717	0.735	60.268	1
OverlapCoefficient	0.690	0.709	1.264	2
QGramsDistance	0.649	0.664	32.720	2
Level2JaroWinkler	0.617	0.658	8.069	1
DiceSimilarity	0.598	0.644	1.274	2
CharJaccard	0.559	0.580	1.855	1

is dropped. In our experiments, we use *cosine* [22] similarity as the cheap metric and 0.2 as its threshold. Recall that we consider each row of a dataset as an input string.

The domain dependent method is trained over both positive and negative training samples. We define the training set as stated in [18]. Positive training samples include all the real matching strings. Negative examples are selected by the non-trained method so that the closest estimated match are included in the training set, i.e. the non-match pairs with highest score. We sample a total of negative samples five times the positive sample size. As in [10] and [18], we split the available data into two, half for training and the other half for testing, and repeat the process with the sets interchanged.

Since the positive training set might be very small in some cases, the candidate list of matching characters might exclude some matches. To avoid this, we sample all possible different pairs of tokens in the dataset and obtain a matching score s_{ij} (T_i, U_j), so that all matched pairs greater than a certain threshold Φ are included in the training set or as input for the parameter estimator algorithm. Here we set $\Phi = 0.7$.

7.4 Results

We report the evaluated precision of the 14 methods that performed best on average on all datasets. As shown in Table 5, the non-trained method performs best in 4 out of 7 cases, and its performance is very close to the best method in all other cases. The trained method performs best in all cases, except the census dataset, which is the most corrupted. The main advantage of the method proposed here is that it is consistent in its performance, unlike the other methods

which show poor results in some cases. The average performance in both Precision and maximum F1 is shown in Table 6. In this case, both the trained and the non-trained methods perform in average better than the rest of the methods and the average computation time is also favorable compared to SoftTFIDF and Level2 hybrid methods.

8 Conclusions

We proposed a novel hybrid string metric, which selects matching pairs of tokens in a one-to-one setting, similar to the assignment problem. We believe this setting selects pairs of tokens in a better way than past approaches and it is computationally competitive. We use the same idea of assigning pairs of tokens to assign pairs of characters in order to define a new character based method. This method is combined with the Winkler scorer [23] and that improves its accuracy. For our token-based method, we define a variant of TFIDF weighting scheme which measures the ratio of matching characters common in pairs of strings. As mentioned before, this weighting scheme is better than TFIDF for the task of matching short strings.

The parameters of the proposed string metric can be estimated using the domain of the data to be processed. Although existing methods can perform very well in some cases, they can show a very poor performance in others. Our method, particularly when trained, performed consistently well, at least in all cases considered.

References

1. Newcombe, H.B., Axford, S., James, A.: Automatic linkage of vital records. Science **130** (1959) 954–959
2. Hernandez, M.A., Stolfo, S.J.: The merge/purge problem for large databases. In: SIGMOD: Proceedings of the International conference on Management of data. (1995)
3. Monge, A.E., Elkan, C.P.: An efficient domain-independent algorithm for detecting approximately duplicate database records. In: SIGMOD: Proceedings of the workshop on data mining and knowledge discovery. (1997)
4. Cohen, W., McAllester, D., Kautz, H.: Hardening soft information sources. In: KDD: Proceedings of the international conference on Knowledge discovery and data mining, Boston, Massachusetts, USA (2000)
5. Cohen, W., Ravikumar, P., Fienberg, S.E.: A comparison of string distance metrics for name-matching tasks. In: IJCAI and IIWEB, Acapulco, Mexico (2003)
6. Jaccard, P.: The distribution of the flora of the alpine zone. New Phytologist **11** (1912) 37–50
7. Salton, G., Wong, A., Yang, C.S.: A vector space model for automatic indexing. Communications of the ACM **18 issue 11** (1975) 613– 620
8. Jaro, M.A.: Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. Journal of the American Statistical Society **89** (1989) 414–420

9. Levenshtein, V.: Levenshtein distance algorithm. Keldysh Institute of Applied Mathematics, Moscow (1965)
10. Bilenko, M., Mooney, R.J.: Learning to combine trained distance metrics for duplicate detection in databases. Technical Report AI 02-296, University of Texas at Austin (2002)
11. Bilenko, M., Mooney, R.J.: Adaptive duplicate detection using learnable string similarity measures. In: KDD: Proceedings of the international conference on Knowledge discovery and data mining, New York, NY, USA (2003)
12. Bilenko, M., Mooney, R.J.: Employing trainable string similarity metrics for information integration. In: IIWEB, Acapulco, Mexico (2003)
13. Bilenko, M., Mooney, R.J.: On evaluation and trainingset construction for duplicate detection. In: KDD: Proceedings of the Workshop on Data Cleaning, Record Linkage, and Object Consolidation, Washington DC, USA (2003)
14. Bilenko, M., Mooney, R.J.: Alignments and string similarity in information integration: A random field approach. In: Proceedings of the Dagstuhl Seminar on Machine Learning for the Semantic Web, Dagstuhl, Germany (2005)
15. Bilenko, M., Mooney, R.J., Cohen, W., Ravikumar, P., Fienberg, S.E.: Adaptive name-matching in information integration. IEEE Intelligent Systems **18 number 5** (2003) 16–23
16. Cohen, W., Richman, J.: Learning to match and cluster entity names. In: SIGIR: Workshop on Mathematical/Formal Methods in Information Retrieval, New Orleans, LA, USA (2001)
17. Leung, Y.W., Zhang, J.S., Xu, Z.B.: Optimal neural network algorithm for on-line string matching. IEEE Transactions on Systems, Man, and Cybernetics, Part B **28 number 5** (1998) 737–739
18. McCallum, A., Pereira, F.: A conditional random field for discriminatively-trained finite-state string edit distance. In: UAI: In Proceedings of the Conference on Uncertainty in Artificial Intelligence. (2005)
19. Monge, A.E.: An adaptive and efficient algorithm for detecting approximately duplicate database records. (2000)
20. Ristad, E.S., Yianilos, P.N.: Learning string-edit distance. IEEE Transactions on Pattern Analysis and Machine Intelligence **20 number 5** (1998) 522–532
21. Yancey, W.E.: An adaptive string comparator for record linkage. In: ASA: Proceedings of the Section on Survey Research Methods. (2003)
22. Chapman, S.: Simmetrics web intelligence, Natural Language Processing Group, Department of Computer Science, University of Sheffield, Regent Court, 211 Portobello Street, Sheffield, S1 4DP, UK. sam@dcs.shef.ac.uk. (2006)
23. Winkler, W.E.: String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage. In: ASA: Proceedings of the Survey Research Methods Section. (1990)