Evaluating String Comparator Performance for Record Linkage

William E. Yancey
Statistical Research Division
U.S. Census Bureau

KEY WORDS string comparator, record linkage, edit distance

Abstract

We compare variations of string comparators based on the Jaro-Winkler comparator and edit distance comparator. We apply the comparators to U.S. Census Bureau data to see which are better classifiers for matches and non-matches, first by comparing their classification abilities using a ROC curve based analysis, then by considering a direct comparison between two candidate comparators in record linkage results.

1. Introduction

We wish to evaluate the performance of some string comparators and their variations for use in record linkage software for U.S. Census Bureau data. For record linkage, under the conditional independence assumption, we compute a comparison weight for two records from the sum of the comparison weights of the individual matching fields. If we designate that the values of a matching field agree for two records by $\gamma = 1$ and that they disagree by $\gamma = 0$, then we define the agreement weight for the two fields by

$$a_w = \log \frac{\Pr(\gamma = 1|M)}{\Pr(\gamma = 1|U)}$$

and the disagreement weight by

$$d_w = \log \frac{\Pr(\gamma = 0|M)}{\Pr(\gamma = 0|U)}$$

where the probabilities are conditioned by whether the two records do in fact belong to the set M of true matches or the set U of true non-matches. Presumably $\Pr(\gamma = 1|M) > \Pr(\gamma = 1|U)$, so that $d_w < 0 < a_w$. If we wish to use a string comparator for the matching field with alphabet Σ, we generally use a *similarity function*

$$\gamma : \Sigma^* \times \Sigma^* \to [0, 1]$$

where $\gamma(\alpha, \beta) = 1$ whenever the strings α, β are identical. We then use an interpolation function w,

$$w : [0, 1] \to [d_w, a_w]$$

to assign a comparison weight $w(x)$ to a pair of strings α, β where the interpolation function is increasing with $w(1) = a_w$.

We next describe the string comparator functions that we used for this study. We then discuss the data sets that were used to test the comparators. Then we discuss how we interpreted the results of this data to try to evaluate the classification power of each of the string comparators. We also look at the difference that the string comparator choice can make in a matching situation.

2. The String Comparator Functions

In the following, let α, β be strings of lengths m, n respectively with $m \leq n$.

2.1 The Jaro-Winkler String Comparators

The Basic Jaro-Winkler String Comparator The Jaro-Winkler string comparator [3] counts the number c of common characters between two strings and the number of transpositions of these common characters. A character a_i of string α and b_j of string β are considered to be common characters of α, β if $a_i = b_j$ and

$$|i - j| < \left\lfloor \frac{n}{2} \right\rfloor,$$

the greatest integer of half the length of the longer string. A character of one string is considered to be common to at most one character in the other string. The number of transpositions t is determined by the number of pairs of common characters that are out of order. The number of transpositions is computed as the greatest integer of half of the number of out-of-order common character pairs. The Jaro-Winkler similarity value for the two strings is then given by

$$x = \frac{1}{3}\left(\frac{c}{m} + \frac{c}{n} + \frac{c - t}{c}\right),$$

[*] This report is released to inform interested parties of research and to encourage discussion.

unless the number of common characters $c = 0$, in which case the similarity value is 0.

There are three modifications to this basic string comparator that are currently in use.

Similar Characters The J-W string comparator program contains a list of 36 pairs of characters that have been judged to be similar, so that they are more likely to be substituted for each other in misspelled words. After the common characters have been identified, the remaining characters of the strings are searched for similar pairs (within the search distance d). Each pair of similar characters increases the count of common characters by 0.3. That is the similar character count is given by

$$c_s = c + 0.3s$$

where s is the number of similar pairs. The basic Jaro-Winkler formula is then adjusted by

$$x_s = \frac{1}{3}\left(\frac{c_s}{m} + \frac{c_s}{n} + \frac{c-t}{c}\right).$$

Common Prefix This adjustment increases the score when the two strings have a common prefix. If p is the length of the common prefix, up to 4 characters, then the score x is adjusted to x_p by

$$x_p = x + \frac{p\,(1-x)}{10}.$$

Longer String Adjustment Finally there is one more adjustment in the default string comparator that adjusts for agreement between longer strings that have several common characters besides the above agreeing prefix characters. The conditions for using the adjustment are

$$\begin{aligned} m &\geq 5 \\ c-p &\geq 2 \\ c-p &\geq \frac{m-p}{2} \end{aligned}$$

If all of these conditions are met, then length adjusted weight x_l is computed by

$$x_l = x + (1-x)\,\frac{c-(p+1)}{m+n-2\,(p-1)}$$

2.2 Edit Distance String Comparators

We wished to compare the Jaro-Winkler string comparators with some string comparators based on edit distance. All of the edit distance type comparator values are computed using a dynamic programming algorithm that computes the comparison value in $O\,(mn)$ time.

Standard Edit Distance The standard edit distance (or Levenshtein distance) [1] between two strings is the minimum number of edit steps required to convert one string to the other, where the allowable edit steps are insertion, deletion, and substitution. If we let α_i be the prefix of α of length i, β_j be the prefix of β of length j, and ε be the empty string, then we can initialize the edit distance algorithm with the distances

$$\begin{aligned} e\,(\alpha_i, \varepsilon) &= i \\ e\,(\varepsilon, \beta_j) &= j \\ e\,(\varepsilon, \varepsilon) &= 0 \end{aligned}$$

indicating the number of insertions/deletions to convert a string to the empty string. We can then build up the cost of converting longer prefixes by computing

$$e\,(\alpha_i, \beta_j) = \min \left\{ \begin{array}{l} e\,(\alpha_{i-1}, \beta_j) + 1 \\ e\,(\alpha_i, \beta_{j-1}) + 1 \\ \left\{ \begin{array}{ll} e\,(\alpha_{i-1}, \beta_{j-1}) & \text{if } a_i = b_j \\ e\,(\alpha_{i-1}, \beta_{j-1}) + 1 & \text{if } a_i \neq b_j \end{array} \right. \end{array} \right. \tag{1}$$

where a_i denotes the i^{th} character of α and b_j is the j^{th} character of β. The final minimum edit cost is then given by $e\,(\alpha, \beta) = e\,(\alpha_m, \beta_n)$.

While the edit distance function is a true metric on the space of strings, it is not a similarity function. We note that the maximum edit length between two strings is n (m substitutions and $n-m$ insertions/deletions), so that the comparator score

$$x_e = 1 - \frac{e}{n}$$

defines a similarity function for string pairs α, β where e is the edit distance between the strings. We note that character order is important to edit distance, so that the three common characters that are out of order result in three edits.

Longest Common Subsequence The length of the longest common subsequence (lcs) of two strings can also be computed by the same algorithm [1], except that this time the only possible edit steps are insertion and deletion, so that the recursive function is

$$e\,(\alpha_i, \beta_j) = \min \left\{ \begin{array}{l} e\,(\alpha_{i-1}, \beta_j) + 1 \\ e\,(\alpha_i, \beta_{j-1}) + 1 \\ e\,(\alpha_{i-1}, \beta_{j-1}) \quad \text{if } a_i = b_j \end{array} \right. \tag{2}$$

Clearly the longest possible common subsequence length for our strings is m, so we can define a longest common subsequence similarity function by

$$x_c = \frac{l}{m}$$

where l is the length of the longest common subsequence of the two strings.

Coherence Edit Distance As we have seen, both edit distance and *lcs* length depend strictly on the order of the characters in the strings. This is because the defining recursion functions (1) and (2) that determine the cost of the current prefix pair depend only on the last characters of the two prefixes, *i.e.* whether they are equal or not. There is no checking to see whether one of these characters occurred somewhere earlier in the other prefix. The Jaro-Winkler string comparator allows common characters to be out of order with a penalty for transpositions. If the edit distance recursion looked back over earlier characters in the prefixes, then a contextual edit score could be given. In [2], Jie Wei uses Markov field theory to develop such a recursion function, which looks like

$$C\left(\alpha_i, \beta_j\right) = \min \left\{ \begin{array}{l} C\left(\alpha_{i-1}, \beta_j\right) + 1 \\ C\left(\alpha_i, \beta_{j-1}\right) + 1 \\ \min_{\substack{1 \le a \le \mathsf{N} \\ 1 \le b \le \mathsf{N}}} \left\{ C\left(\alpha_{i-a}, \beta_{j-b}\right) + V_{a,b}\left(\alpha_i, \beta_j\right) \right\} \end{array} \right.$$

where $V_{a,b}\left(\alpha_i, \beta_j\right)$ is an edit cost potential function and N is a number that indicates the degree of coherence of the strings, *i.e.* the amount of context that should be considered when computing the edit cost. For $V_{a,b}\left(\alpha_i, \beta_j\right)$, we consider the substrings $\alpha_{i-a,i}$ and $\beta_{j-b,j}$ and let c be the number of common characters to these two substrings. Denoting $t = a + b$, we can express Jie Wei's coherence edit potential function as

$$V_{a,b}\left(\alpha_i, \beta_j\right) = \left\{ \begin{array}{ll} \frac{3}{4}\left(t - \frac{2}{3}\right) - c & \text{if } t \text{ is even} \\ \frac{3}{4}\left(t - 1\right) - c & \text{if } t \text{ is odd} \end{array} \right.$$

He also chooses $\mathsf{N} = 4$ as a reasonable coherence index for words. With this choice of N we can display all possible values of (a, b) and the corresponding range of V_{ab}

(a,b)	t	$\max V_{ab}$	$\max c$	$\min V_{ab}$
$(1,1)$	2	1	1	0
$(1,2)$	3	1.5	1	0.5
$(1,3)$	4	2.5	1	1.5
$(1,4)$	5	3	1	2
$(2,2)$	4	2.5	2	0.5
$(2,3)$	5	3	2	1
$(2,4)$	6	4	2	2
$(3,3)$	6	4	3	1
$(3,4)$	7	4.5	3	1.5
$(4,4)$	8	5.5	4	1.5

The coherence edit distance is always less than or equal to the standard edit distance, so the maximum possible distance is still the length of the longer string, and we can define a coherence edit similarity score by

$$x_w = 1 - \frac{C}{n}$$

where C is the coherence edit distance of the two strings.

Combination Edit Distance When we began studying and evaluating string comparators using the approach discussed in Section 5., we found that the edit distance similarity function did not perform as well as the J-W comparator. This may be because it does not use enough information from the strings. In particular, it does not consider the length of the shorter string. Thus we tried combining the edit distance and the *lcs* comparators by averaging them

$$x_{ec} = \frac{1}{2}\left(\left(1 - \frac{e}{n}\right) + \frac{l}{m}\right)$$

which seemed to produce better results. The example string pair has a combined edit score of $\frac{11}{24} \doteq 0.4583$. There could be a more optimal weighting of the two comparators.

Combination Coherence Edit Distance We also considered the effect of combining the coherence edit distance and the *lcs* comparators.

$$x_{wc} = \frac{1}{2}\left(\left(1 - \frac{C}{n}\right) + \frac{l}{m}\right).$$

2.3 Hybrid Comparators

Our initial analysis of the results of our experiment led us to consider combining a J-W comparator with an edit distance type comparator. We will consider this development after we describe the experiment.

3. Data Sets

There does not appear to be any theoretical way to determine which is the best string comparator. In fact, there does not appear to be a clear meaning of "best" other than the comparator that performs best on a given application. We therefore want to conduct an experiment to see which string comparator does the best job for the application of record linkage of U.S. Census Bureau data. Since the string comparator is just one component of the record linkage procedure, we first try to isolate the string comparator's contribution.

At the Census Bureau, we have some test decks that are pair of files where the matches have been clerically determined. One large pair of files come from the 2000 Census and the ACE follow-up. These files each have $606,411$ records of persons around the country where each record of one file has been matched with one record of the other. There are also three smaller pairs of files, denoted 2021,3031, and STL, from the 1990 Census and the PES follow-up. Each of these files is of persons in the same geographic area where not all of the records have matches.

The data sets were formed by bringing together the records that were identified as matches and writing out the pairs of last names or first names whenever the two name strings were not identical. We then removed all duplicate name pairs from the list. From the 2000 data we obtained a file of $65,325$ distinct first name pairs and $75,574$ distinct last name pairs. From the three 1990 files combined we obtained three files of 942, 1176, and 2785 distinct first and last name pairs.

4. The Computation

The purpose of the string comparator in record linkage is to help us distinguish between pairs of strings that probably both represent the same name and pairs of strings that do not. For the sets M of matched pairs, we will use data sets of nonidentical name pairs from matched records. Our data sets do not necessarily contain only string pairs representing the same name, since some records may have been linked based on information of other fields than this name field. However, they should tend to have similarities that one may subjectively judge to suggest that they refer to the same name. For our sets U of unmatched pairs, we will take the set of cross pairs of every first member name in the set paired with every second member name *other* than its match. We may think of these as unmatched pairs, but they are really more like random pairs, since there can be pairs of names in U which match exactly. Thus we can never completely separate the set M from the set U, but the test will be which string comparator can include the most elements of M with the fewest elements of U.

For each string comparator under examination, we compute the string comparator value of each first member name with each second member name. The sets of Census 2000 names are too large to store the resulting comparator values, so we split each of the sets into 24 subsets of name pairs, The first name pair subsets have 2722 pairs (2719 for the last one) and the last name pairs have 3149 pairs (3147 for the last one). Thus we compute the values of 13 string comparators, 8 J-W comparators with all combinations of the three modifications and 5 edit distance type comparators, of all cross pairs of strings in 51 sets of name pairs to generate our string comparator output data.

5. Analysis of Scores

We now face the problem of determining how to use this data to measure the performance of the string comparators. We can view the problem as a binary classification problem: a string pair either belongs to M or it belongs to U. One tool for analyzing classification effectiveness is the ROC curve. The ROC (receiver operating characteristic) curve originated for use in signal detection, but is now commonly used in medicine to measure the diagnostic power of a test. A list of references can be found on [6]. If we let γ be a string comparator similarity function, we measure the discriminatory power of the string comparator with the parameterized curve

$$\left(\Pr\left(\gamma \geq t|U\right), \Pr\left(\gamma \geq t|M\right)\right), \quad t \in [0,1]$$

in the unit square. The resulting ROC curve is then independent of the parameterization. This is sometimes referred to as plotting sensitivity against $1-$ specificity. If we denote the probability density of the M condition by $p_M(t)$ and the density conditioned on U by

$p_U(t)$ so that

$$\begin{aligned}\frac{d}{dt}\Pr(\gamma \geq t|M) &= -p_M(t)\\ \frac{d}{dt}\Pr(\gamma \geq t|U) &= -p_U(t)\end{aligned}$$

then we see that the slope of the tangent to the ROC curve is

$$\frac{dy}{dx} = \frac{\frac{dy}{dt}}{\frac{dx}{dt}} = \frac{p_M(t)}{p_U(t)}$$

the likelihood ratio of the two distributions. The diagnostic tool that is used is the AUC, the area under the ROC curve. To interpret this AUC, we define the independent random variables

$$\begin{aligned}X &: M \to [0,1]\\ Y &: U \to [0,1]\end{aligned}$$

to be the string comparator value for a pair randomly drawn from M or U respectively, which have probability density functions p_M and p_U respectively, then

$$\begin{aligned}AUC &= \int_0^1 \Pr(\gamma \geq t|M)\, d\Pr(\gamma \geq t|U)\\ &= \Pr(X \geq Y),\end{aligned}$$

the probability that a randomly chosen element of M will have a higher score than a randomly chosen element of U. Thus an $AUC = 1$ would indicate that the string comparator γ is a perfect discriminator between M and U, and an $AUC = \frac{1}{2}$ would indicate that γ has no discriminating power whatsoever between M and U. Hence the nearer AUC is to 1, the more effective the discriminator between the two sets.

We used our data to compute the AUC for each of our string comparators, but then we realized that for our application, the full AUC is not a very relevant statistic. In our record linkage program, the string comparator similarity score is linearly interpolated to produce an agreement weight between the full agreement weight and the full disagreement weight. When the interpolation value is less than the disagreement weight, the disagreement weight is assigned. Thus all similarity scores below a cutoff value are treated the same, as indicating that the sting pairs are in U. The only discrimination happens for string comparators above this cutoff. Thus for sufficiently small values of $\alpha \in [0,1]$, we instead looked at values of

$$\frac{1}{\Pr(\gamma \geq t_\alpha|U)}\int_0^\alpha \Pr(\gamma \geq t|M)\, d\Pr(\gamma \geq t|U)$$

where $t_a \in [0,1]$ such that $\Pr(\gamma \geq t_\alpha|U) = \alpha$. Since $d\Pr(\gamma \geq t|U) = -p_U(t)\, dt$,

$$\begin{aligned}&\int_0^\alpha \Pr(\gamma \geq t|M)\, d\Pr(\gamma \geq t|U)\\ &\quad= \int_{t_\alpha}^1 \Pr(\gamma \geq t|M)\, p_U(t)\, dt\\ &\quad= \int_{t_\alpha}^1 \int_t^1 p_M(s)\, p_U(t)\, ds dt\end{aligned}$$

we see that this is the probability that $X \geq Y$ and $Y \geq t_\alpha$. Thus

$$\begin{aligned}&\frac{1}{\Pr(\gamma \geq t_\alpha|U)}\int_0^\alpha \Pr(\gamma \geq t|M)\, d\Pr(\gamma \geq t|U)\\ &\qquad= \Pr(X \geq Y|Y \geq t_\alpha).\end{aligned}$$

As $\alpha \nearrow 1$, then $t_a \searrow 0$, and we see that the interpretation agrees with the standard one in the limit.

The weight interpolation function currently in use in the matching software is

$$w = a_w - 4.5\,(a_w - d_w)\,(1 - s)$$

where s is the string comparator score from the Jaro-Winkler comparator using all three modifications. With this interpolation, we see that we will get the full disagreement weight $w = d_w$ for

$$s_- = \frac{7}{9} \doteq 0.778.$$

When we look at the J-W comparator scores as a function of the "error rate" $\Pr(\gamma \geq t|U) = \alpha$, we see that we are past this boundary score by the time $\alpha = 0.02$, sometimes by $\alpha = 0.01$. Thus we are interested in only a small sliver of the total area under the ROC curve. Moreover, the region corresponding to a positive agreement weight is smaller still. For example, if we have parameters that result in $d_w = -a_w$ (*i.e.* $\Pr(\gamma = 1|M) + \Pr(\gamma = 1|U) = 1$), then the agreement weight $w = 0$ when

$$s_+ = \frac{8}{9} \doteq 0.889.$$

Suppose that we designate by p_+, p_- the "error rate" probabilities for the full Jaro-Winkler (all options) scores where for which

$\Pr(\gamma > s_+|U) = p_+$ and $\Pr(\gamma > s_-|U) = p_-$. These boundary error probabilities remain mostly consistent within the related groups of data sets: the three sets of names from 1990, the 24 sets of first names from 2000, and all but 3 of the sets of last names from 2000. The last three sets of last names are consistent with each other but have a different error/score profile than the first 21 sets. This appears to be because these last sets consist mostly of Hispanic last names and the random cross pairs contain a higher proportion of incidental exact matches. Approximate values of these cutoff error rates are given in the following table.

Data Sets	p_+	p_-
1990 Names	0.0014	0.017
2000 First Names	0.0012	0.014
Main 2000 Last Names	0.0004	0.01
Subgroup 2000 Last Names	0.006	0.022

In all cases we see that for the given weighting function for the full J-W string comparator results in a very small part of the range $0 \leq \Pr(\gamma > t|U) \leq 1$ is relevant for the diagnostic power of the ROC curve. We will consider only such restrictive ranges when comparing the relative strengths of the candidate string comparators. That is, for restrictive values of x_i, for the corresponding value of t_i, where

$$\Pr(\gamma > t_i|U) = x_i$$

we consider the value of

$$p(x_i) = \frac{1}{\Pr(\gamma > t_i|U)} \int_0^{x_i} \Pr(\gamma \geq t|M)\, d\Pr(\gamma \geq t|U).$$

5.1 Summary of Results

We summarize the results of comparing values of $p(x_i)$ for small values of x_i for each of the string comparators on each of the 51 data sets. We consider eight versions of the Jaro-Winkler comparator with each combination of the three adjustment functions. We consider four edit-distance type comparators: Levenshtein and coherence with and without LCS. More details with graphs and tables are available in [5].

The main observation is that the results were fairly consistent for the three 1990 sets, the 24 2000 first name sets, and the first 21 of the 2000 last name sets. The last three sets of last names from the 2000 census produced different results. This will be discussed further below.

For the 48 regular data sets, the 8 versions of the J-W comparator were fairly close. The two versions that used the suffix adjustment without using the prefix adjustment did noticeably worse than the other 6. The prefix adjustment helped the most, but each adjustment produced a small improvement, so that the version using all 3 adjustments did best. However, the $p(x)$ scores for these 6 versions did not differ by more that 0.03, For the last 3 last name data sets, the prefix adjustment helped, but the other two adjustments produced worse results, so that the best version used only the prefix adjustment and the worst of the 6 was the full J-W version, with the $p(x)$ scores differing by as much as 0.16.

For the edit-distance type comparators, the inclusion of the LCS function always produced better results. Also the coherence edit distance comparator always did worse than standard edit distance, differing more in the positive comparison weight range, so there does not seem to be any justification for the extra complexity for this application. One might experiment with different coherence index other than $N = 4$, but this does not appear to be very promising.

Comparing the full J-W comparator with the edit distance/LCS comparator, we found that the performance on the standard 48 files was about the same. The J-W comparator doing slightly better on the first name sets and the edit distance comparator slightly better on the last name sets. However, for the anomalous 3 last name sets, the edit distance comparator did better than the J-W comparator with $p(x)$ scores differing by around 0.1.

5.2 Hybrid Comparator

Differences Between the Jaro-Winkler and Edit Distance Type Comparators To understand how the performance of edit string comparators differs from that of the Jaro-Winkler comparators, we chose a selectivity cutoff value and looked at the pairs of names from matching records that are above the cutoff value for one comparator and below the cutoff value for the other comparator. We similarly looked at the analogous name pairs from non-matching records. Looking at the name pairs which have an above cutoff value for the J-W comparator and below cutoff value for the LCSLEV

comparator, we did not perceive a pattern to the name pairs from either matching or non-matching records. However, there was a similarity to the name pairs that exceeded the LCSLEV cutoff and were below the J-W cutoff. These name pairs highlight some asymmetries of the J-W comparator that can result in some low scores especially for double names.

We can understand the major asymmetry of the J-W comparator as follows. Let α, β be two strings both of length n with no characters in common and let $\alpha\beta$ be the concatenation of these two strings. If we use the J-W comparator on the pair $\alpha, \alpha\beta$, we get a fairly high score. Specifically, the two strings have n common characters with no transpositions for a basic J-W score of $\gamma(\alpha, \alpha\beta) = \frac{5}{6}$. If $n \geq 4$, the prefix adjustment raises the score to $\gamma(\alpha, \alpha\beta) = \frac{9}{10}$. On the other hand, $\gamma(\beta, \alpha\beta) = 0$ since all of the common characters are beyond the search distance. However, the LCSLEV comparator results in the same score for either pair $\gamma(\alpha, \alpha\beta) = \gamma(\beta, \alpha\beta) = \frac{3}{4}$. The three last files of last name pair from 2000 especially contain a lot of Hispanic names where the surname from one file is reported as a double name and the other file just has one of the two names, so this distinction in comparator behavior can be relevant.

Selecting a Hybrid Comparator The standard J-W comparator generally does well in our selectivity, average sensitivity analysis. The combination Levenshtein distance and longest common substring comparator performs comparably. However, in our extreme cases of the last three sets of the last name pairs, the edit distance type comparator does significantly better than the J-W comparator, probably because of the more robust handling of the single name/double name pairs. We consider that it might be advisable to use the J-W comparator except in those cases where it gives a small value compared to the edit comparator. However, we need to be able to compare the string comparator values from the J-W comparator and the edit distance comparator.

We tried combining the JW110 comparator with the LCSLEV comparator. We used the one without the suffix adjustment since this adjustment generally made a small difference, sometimes this difference was negative, and we thought of the LCSLEV comparator as offering a suffix correction. We considered the values of the JW110 and LCSLEV comparators for the same selectivity values, restricted to selectivity values in a range relevant to the assignment of varying matching weights. The comparators show a strong and consistent linear relationship in this range, where we estimated the regression coefficients to be

$$\hat{s}_{110} = 0.66 s_{lcslev} + 0.38$$

and define the hybrid string comparator score by

$$s_h = \max\left(s_{110}, \min\left(\hat{s}_{110}, 1\right)\right).$$

This may not be the best way to combine the two string comparators. This hybrid string comparator has some unappealing formal properties. The minimum value of s_h is 0.38 instead of 0, but comparator values this low will be assigned the full disagreement weight anyway. Also it is possible to have $s_h(\alpha, \beta) = 1$, but $\alpha \neq \beta$ For instance, if $\alpha = a_1a_2a_3a_4a_5a_6a_7a_8a_9$ has distinct characters and $\beta = a_2a_3a_4a_5a_6a_7a_8a_9$, then

$$s_{lcslev} = \frac{1}{2}\left(\frac{8}{9} + \frac{8}{8}\right) = \frac{17}{18} \doteq 0.944$$

which results in $\hat{s}_{110} > 1$. On the other hand, we have

$$s_{110} = \frac{1}{3}\left(\frac{8}{9} + \frac{8}{8} + \frac{8}{8}\right) = \frac{26}{27}$$

and

$$s_{111} = \frac{26}{27} + \frac{1}{27}\left(\frac{8-1}{8+9+2}\right) = \frac{167}{171} \doteq 0.97660.$$

This is a high score but it would receive somewhat less than the full agreement weight.

Assessing the Hybrid Comparator When we tested the hybrid comparator with the full J-W and the edit/LCS comparators, the hybrid comparator appeared to succeed in capturing the advantages of the two comparators. For the first name files, the hybrid comparator was close to the J-W comparator, closer than the edit/LCS comparator. For the standard last names, the hybrid comparator did a little better than either the J-W or the edit/LCS comparator. On the last three anomalous name sets, the hybrid comparator was close to the edit/LCS comparator and well above the J-W comparator.

6. Matching Results

We have analyzed several string comparators by examining their average sensitivity over selectivity intervals, concentrating on intervals where the string comparator values can have some influence on the matching weights for record pairs in record linkage. We have seen some, mostly slight, differences between these comparators. We would like to know if these measured differences are enough to effect the final record linkage result.

We ran matching programs using the full J-W comparator and the hybrid comparator. We computed 3 blocking passes on the 2000 data without one-to-one matching and then applied one-to-one matching to these outputs. We applied one-to-one matching to each of the three 1990 data sets. To briefly summarize, the differences in the matching results are fairly subtle, but one can detect a slight improvement using the hybrid comparator. Generally speaking, at the same level of false matches, there hybrid comparator produces a slightly higher number of true matches. Also at comparable cutoff levels, the hybrid comparator produces a clerical region of somewhat reduced size. More details are available in [5].

6.1 Summary

In the previous analysis using ROC curve values, we saw that the Jaro-Winkler comparator with all three adjustments and the hybrid comparator which combines the Jaro-Winkler comparator without the suffix adjustment and the combination of edit distance and longest common subsequence comparator both were good performers in classifying the Census name typographical error data. The hybrid comparator appeared to generally do somewhat better. When we use the two comparators in our matching software, the hybrid comparator continues to do slightly better in classifying the matches and the non-matches. It finds very few extra matches, but it does tend to separate the matches from the non-matches. The cost is that the hybrid matcher takes much longer to run, essentially performing three quadratic algorithms instead of one.

REFERENCES

[1] Stephen, Graham A. *String Searching Algorithms.* World Scientific Publishing Co. Pte. Ltd., 1994.

[2] J. Wei. "Markov Edit Distance". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol 26, No. 3, pp. 311–321, 2004.

[3] Winkler, William E. "String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage". *Proceedings of the Section on Survey Research Methods, American Statistical Association*, 1990, pp. 354–359.

[4] Winkler, William E. "Advanced Methods for Record Linkage". http://www.census.gov/srd/www/byname.html

[5] Yancey, William E. "Evaluating String Comparator Performance for Record Linkage." http://www.census.gov/srd/www/byname.html

[6] Zou, Kelly H. *Receiver Operating Characteristic (ROC) Literature Research.* http://splweb.bwh.harvard.edu:8000/pages/ppl/zou/roc.html.