INSTITUTE FOR SOCIAL RESEARCH

New Mexico Statistical Analysis Center

Assessing Record Linkage Matches Using String Distance Measures

Prepared by
Kristine Denman and Vaughn Fortier-Shultz

Edited by
Ashleigh Maus

July 2019

This project was supported by Grant # 2017-BJ-CX-K018 from the State Justice Statistics program. The State Justice Statistics program is a component of the Office of Justice Programs, which also includes the Bureau of Justice Statistics, the National Institute of Justice, the Office of Juvenile Justice and Delinquency Prevention, and the Office for Victims of Crime. Points of view or opinions in this document are those of the author and do not represent the official position or policies of the United States Department of Justice.

Introduction

The New Mexico Statistical Analysis Center (NMSAC) often links records in two or more datasets using personal identifiers. Typically, the procedure the NMSAC uses is to match by last name, first name, date of birth (DOB) and the last four digits of the social security number (SSN). However, name changes, missing data, typing errors, and different formatting standards complicates the matching of records using exact criteria. Thus, after the initial match, we create and use Soundex name variables to perform "fuzzy" matching on records that do not initially achieve a deterministic match along with other identifiers (date of birth, SSN).[1] We then loosen the criteria (e.g., Soundex last name, DOB, SSN). After the matches are completed, staff manually checks each matching name pair and assigns a match value using a table of commonly encountered discrepancies. This process is time consuming. Therefore, we explored the efficacy of using string distance algorithms to minimize the amount of time spent on manual review. This report summarizes our findings.

String distance algorithms

String distance algorithms generate a number representing the disparity between two string variables. String variables include non-numeric values (letters, commas, spaces, etc.). Here, we examine the use of string distance algorithms for last and first names.

According to van der Loo (2014), there are three general types of string distance algorithms:

1. Edit-based distances;
2. Q-gram based distances; and
3. Heuristic distances.

Edit-based distances:

These measures count the number of discrete edit operations required to turn one string into another string. Edit-based distances allow one or more of the following operations: substitution of a character (e.g., a for e), deletion of a character (e.g., Martine -> Martin), insertion of a character (e.g., Jon -> John), and transposition of two adjacent characters (e.g., Kaira -> Akira). There are several different measures within this category of algorithms, each of which are calculated using slightly different criteria. We explored the following:

- Longest Common Substring (LCS) – This measure counts the number of deletions and insertions required for one string to be transformed into another string. For two strings with lengths *x* and *y*, the LCS varies between 0 (perfect match) and $x + y$ (no characters in common). It also takes into account an order-preserved substring of matching characters between the two strings, hence the name "longest common substring." For example, consider the names Megan and Marvin. Maintaining order, it is possible to form the substring {M, A, N} from both names. The number of leftover letters is the LCS distance: {E, G, R, V, I}, for an LCS of 5. However, if the names were written as Megna and Marvin, the longest common substring would be {M, N} due to the order of the letters. The LCS score would then be 7 {E, G, A, A, R, V, I}.

- Levenshtein distance (LV) – Like LCS, this measure counts the number of insertions, deletions, and substitutions required to transform one string into another string. The minimum distance is zero for identical strings, while the maximum is bound by the length of the longer string. For a step-by-step illustration, consider again the case of "Megan" and "Marvin." Starting with

[1] Sample code to create Soundex variables is available in Appendix A.

Megan, how many insertions, deletions, and substitutions does it take to arrive at Marvin? Again, there are many possible routes to consider, but here is one that delivers the most efficient distance of 4:
Step 1: Megan -> Magan (substitution)
Step 2: Magan -> Mavan (substitution)
Step 3: Mavan -> Marvan (insertion)
Step 4: Marvan -> Marvin (substitution)

Those four steps show us that the Levenshtein distance between these two names is 4.

- Damerau-Levenshtein distance (DL) – This is an extension of the Levenshtein distance which counts the number of insertions, deletions, and substitutions needed to change one string into another, but allows for transpositions. The DL distance has the same range of values as the LV distance, but DL distance would be lower if the difference between two strings is due to one or more transpositions. For example, the DL value for the difference between "Pendleton" and "Pendelton" is 1:

  Step 1: Pendleton -> "Pendelton" (transposition)

  The LV value for the same pair would be 2:
  Step 1: Pendleton -> Pendeeton (substitution)
  Step 2: Pendeeton -> Pendelton (substitution)

  Since there are no transpositions needed to change Marvin to Megan, the DL value would be 4, like the LV value.

- Optimal String Alignment (OSA) – This measure is a variation of DL. The range of values which OSA distance can assume is the same as for other edit-based distances. The difference between OSA and DL is that in OSA, no substring can be edited more than once without penalty. In other words, each change needed to convert a substring adds a value to OSA. Thus, this value may exceed DL though these measures are most often the same. Indeed, the values are the same for the name pairs we have used to date.

  An example showing the difference in values (the added penalty with OSA) is the difference from "Lea" to "Al."[2]

  For LV, the value would be 2:
  Step 1: Al -> La (transposition)
  Step 2: La-> Lea (insertion)

  The OSA value would be 3, since you cannot make a change to a string more than once (cannot insert "e" once you have transposed it without an additional step):
  Step 1: Al -> L (deletion)
  Step 2: L-> Le (insertion)

[2] Example adapted from https://en.wikipedia.org/wiki/Damerau%E2%80%93Levenshtein_distance

Step 3: Le -> Lea (insertion)

Essentially, DL is the most liberal and LCS is the most conservative of these measures.

Q-gram based distances: This set of measures allows the user to determine the length of the substrings (q-grams) compared. Q-grams compare substrings of *q* consecutive characters. This can range from one to infinity. For example, bigrams are q-grams of length 2 (e.g., "ab"), trigrams are q-grams of length 3 (e.g., "abc"), etc. The following string distance algorithms use q-grams to determine string distance.

- Q-gram distance –This measure compares all possible q-grams and returns a value that is a discrete count of unpaired q-grams between two sets of strings. Thus, a value of 0 means there are no unpaired strings and any value above that is the number of unpaired strings.
  This measure is obtained by comparing the set of q-grams in string 1 to the set of q-grams in string 2, and counting the number of q-grams that are *not* shared (i.e., the count of q-grams that do not appear in both string's sets of q-grams).

  Q-gram at a value of 1 does not require a preserved order. Thus, when comparing the names "Diaz" and "Daiz" at a q-gram value of 1, the q-gram distance will equal "0" (a perfect match). This is because the number of shared letters in each set {Diaz} and {Daiz} are the same even though they are not in the same order.

  At *q*=2, the total number of possible bigrams between "Diaz" and Daiz" is 6: {Di, ia, az} and {Da, ai, iz}.[3] The q-gram distance for a q-gram of 2 equals 6 because there are no common substrings. At q=3, the q-gram distance would be 4: there are no common substrings among the total of four possible trigrams {Dia, iaz} and {Dai, aiz}.

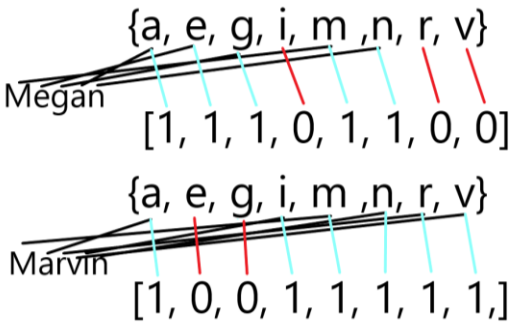
- Jaccard distance – The Jaccard distance is similar to the q-gram distance, but calculates the number of shared q-grams between two strings (the intersection) divided by the union of all q-grams in the two strings. The union includes the total number of shared letter sets, plus the total number of unshared letter sets from each name. The result is subtracted from 1. A perfect match returns a score of zero, while no shared q-grams returns a score of 1. As with other q-gram measures, the user defines the number of q-grams (substring lengths) compared.

  As an example, compare the names "Joe" and "Jose" setting q=2. "Joe" can be broken down into the bigrams "jo" and "oe." "Jose" can be broken down into the bigrams "jo," "os," and "se." The intersection of these sets is "jo," the only bigram that appears in both names. The union of these sets is the set of 4 bigrams {jo, oe, os, se}. The Jaccard distance is $D_j = 1 - ¼ = ¾$ or .75. The value ¼ in this equation indicates that there is one common match ("jo") out of four possible unique matches. The Jaccard distance is a (ratio) measure whose values range from 0 (perfect match) to 1 (no matches).

- Cosine distance – Cosine distances are more difficult to calculate than the other measures discussed thus far. This measure captures the distance of the angle between two vectors rather than differences in attributes alone. The measure takes all of the letters from each name (individually if q is set at 1, in pairs if q is set at 2, etc.), sorts them alphabetically into unique values, then compares each to the original names to create vectors. The angles of the vectors are calculated and then subtracted from 1. Like the Jaccard distance, the values range from 0 to 1, with 0 indicating a perfect match.

  To begin calculating the cosine distance, first define the q-gram, then identify the set of all characters which appear at least once in one or both of the strings, and sort them alphabetically. For example, take the names Megan and Marvin at q=1. The set of all characters which appear at least once in one or both strings would be, alphabetically, {a, e, g, i, m, n, r, v}. Using this, we can generate vectors for each string, with entries corresponding to the number of times each character appears in each string, following the order established in the set of characters from the previous step. So, for Megan, the corresponding vector would be [1, 1, 1, 0, 1, 1, 0, 0], and for Marvin, the corresponding vector would be [1, 0, 0, 1, 1, 1, 1, 1].

  Figure 1 - Creating Vectors for Character Occurrences

  Next, calculate the dot product of the two vectors. Returning to our example of Megan and Marvin, the two vectors are: [1, 1, 1, 0, 1, 1, 0, 0] and
  [1, 0, 0, 1, 1, 1, 1, 1].
  Multiply the corresponding entries, then sum the products:
  [(1*1)+(1*0)+(1*0)+(0*1)+(1*1)+(1*1)+ (0*1)+(0*1)]=3.

  Next, find the square root of the sum of the squares of the elements in each vector. For 1s and 0s, the squared values equal the original values, and so in our case, we simply take the square root of the sum of values in each vector:

  Megan: $\sqrt{}(1^2 + 1^2 + 1^2 + 0^2 + 1^2 + 1^2 + 0^2 + 0^2) = \sqrt{5}$

  Marvin: $\sqrt{}(1^2 + 0^2 + 0^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2) = \sqrt{6}$

  Finally, calculate the cosine distance: $D_{cos} = 1 - \frac{3}{\sqrt{5}*\sqrt{6}} \cong 0.4523.$

R subtracts this value from 1 to preserve consistency with the other string distance algorithms. Thus, the final reported value for cosine distance ranges from 0 (perfect match, all q-grams are shared) to 1 (no q-grams in common).

Heuristic Distances: Two measures are included here: the Jaro distance and the Jaro-Winkler distance. The premise of these measures is that likely matches involve typing errors (mismatches, transpositions) with keys on the keyboard that are near one another. Errors involving keys that are further apart are likely mismatches.

- Jaro distance –The Jaro distance measures the number of matching characters in two strings that are not too far apart on a keyboard, with a penalty for transposed matching characters (van der Loo: 119). Nonadjacent transpositions are allowed. The Jaro distance reports a value between 0 (perfect match) and 1 (no matching characters).

The Jaro string distance measure includes a penalty for transpositions. The formula is: $D_{Jaro} = 1 - (\frac{1}{3})(\frac{m}{x} + \frac{m}{y} + \frac{m-t}{m})$, where x is the length of string 1, y is the length of string 2, m is the number of letters that match between the strings, and t is the number of transpositions. Transpositions are counted in the matched number ("m") if the number of transpositions are equal to or less than the following: the length of the longer string divided by 2 minus 1.

For example, consider Salvador and Salvadro. The length of each string is 8, and the number of transpositions allowed is 3 ((8/2)-1). The first six letters {S, A, L, V, A, D} are a perfect match. The last two letters match if they are transposed. Since one pair is considered one transposition, this example includes only one transposition and those letters are allowed as a match. Thus, the Jaro distance $D_{Jaro} = 1 - \left(\frac{1}{3}\right)\left(\frac{8}{8} + \frac{8}{8} + \frac{8-1}{8}\right) = \frac{23}{24} = 0.042.$

- Jaro-Winkler distance – This measure is an extension of the Jaro distance. Jaro-Winkler modifies the formula by incorporating a penalty for mismatches among the first 4 characters. Winkler's rationale was that people entering data are less likely to make mistakes in the first 4 characters, or that errors in the first 4 characters are more likely to be noticed and corrected (van der Loo: 119). As such, the Jaro-Winkler distance is less forgiving on such errors, believing that they may likely present evidence that the strings are not a good match.

  Jaro-Winkler is an extension of the Jaro measure, but adds a penalty. The user assigns the weight of the penalty, *p,* which is constrained between 0 and 0.25. Winkler also introduces the variable *l*, denoting the length of the longest common prefix between the two strings, up to 4 characters in length. Then, the Jaro-Winkler distance is equal to $D_{JW} = D_{Jaro} * (1 - (p)(l))$; in other words, it's equal to the difference between the Jaro distance and the Jaro distance times the product of the penalty weight and the prefix length. When *p* is equal to zero, the Jaro-Winkler distance is the same as the original Jaro distance.

Like the Jaro measure, the Jaro-Winkler measure is a proportion ranging from 0 (a good match) to 1 (a non-match).

Using the example above, the Jaro-Winkler distances for the pair {Salvador, Salvadro} are calculated at different values of *p*. The longest common prefix has a length of 6, but the formula only allows up to 4:

p=0: Same as Jaro distance, 0.042
p=0.10: $D_{JW} = (0.042) * \big(1 - (0.10)(4)\big) = (0.042) * (0.60) = 0.0252$
p=0.20: $D_{JW} = (0.042) * \big(1 - (0.20)(4)\big) = (0.042) * (0.2) = 0.0084$
p=0.25: $D_{JW} = (0.042) * (1 - (0.25)4) = (0.042) * (0.00) = 0.000$

If *p*=0.25 is selected, any string pair with the same first 4 letters will return a Jaro-Winkler distance of 0.

Data and methods

The purpose of the study is to evaluate whether we could use the string distance measures defined above to minimize manual matching. The data used in this study consists of 2,880 unique pairs of last names and 2,562 unique pairs of first names that were not perfect matches. Several key variables are included. First, for each name pair, SAC staff assigned a match score. Scores range from 0 to 88. A match score of "0" indicates there is not a match, while scores of "1" (not included here) indicate a perfect match. Scores beginning with 2 (e.g., 21) are probable matches while scores beginning with 3 are possible matches, but more suspect. We weigh scores beginning with 3 less heavily than scores in the 20s when determining whether a true match has been found. While we assign different values to matches that fall within the 20s range or 30s range, this is simply for us to know why the match differs and does not indicate strength of a match within those ranges. Finally, a score of "88" indicates data are missing from one or both records. For this analysis, several staff members assigned the match scores. Staff agreed on most scores, but some were re-evaluated and a final decision made.

A second variable included in this dataset is whether or not the name pair includes a Soundex match. We typically search for matches using Soundex variables. We wanted to explore whether the inclusion of Soundex matches in conjunction with the string distance measures improved the accuracy of the predicted match.

Third, for each name pair, we calculated the string distance measures described above. The user must make decisions about cutoffs for q-gram and the Jaro-Winkler measures. For each of the q-gram measures, we calculated the distance for q=1, q=2, and q=3. For the Jaro-Winkler measure, the user defines a penalty/weight. We used: 0 (which returns the same score as obtained using Jaro), .10 and .20.

Finally, we calculated the difference between the lengths of each name included in the sample. In other words, we computed how many characters were in name 1 and in name 2 and calculated the difference.

In addition to the primary dataset used to calculate and explore the string distance measures, we used a small dataset of first name pairs and one with last name pairs to confirm the results. The first dataset consisted of 71 imperfect last name pairs, and the second of 51 imperfect first name pairs.

Manual match scores

Summarized in Table 1 below is the distribution of last name match scores overall and by Soundex match. The first column indicates the number assigned by staff, the second is a definition of that number. Staff deemed just over half of the last name pairs as a non-match. Approximately one-third of the last names were considered probable matches. These differences were typically due to minor misspellings, hyphenated names in reverse order, the inclusion of a suffix in one field, or partial name match when compound names were present. Less likely matches, those coded as "3," account for 8% of the sample. These include names that could be the same, but are less certain. This accounts for problems that arise because fields are truncated or differences due to adding a letter in one name and omitting it from the second (Martine vs. Martin).

The last two columns summarize the matche scores by whether the pair has a Soundex match. The majority of names with a Soundex match are associated with a good match score while most of those without a Soundex match have a match score of "0." However, if we used only this algorithm without the additional step of manual checking, we would include 20% of cases that are not true matches and miss 26% that are likely or possible matches. Thus, while Soundex matching is useful, we must check the results to ensure accuracy.

Table 1 - Last name match score frequencies

Last name match score	Description of match score	All cases	Soundex matches	Soundex does not match
0	Not a match (Clearly different names, unlikely to be typing error/truncation of field)	55.7%	20.3%	74.1%
21	Probable match (e.g., minor misspelling, hyphenated names in which the order is switched)	15.6%	40.9%	8.0%
22	Probable match (hyphenated name/one part of name matches, Jones-Smith vs. Jones; suffix in one and not the other, e.g., Jones III vs. Jones)	20.5%	30.0%	2.4%
3	Possible match (e.g., could be a good match, but unsure as when field is truncated, e.g., Martin vs. Martinez)	8.3%	8.8%	15.5%
N		2880	987	1893

Table 2 summarizes the match scores for the first name pairs. The coding scheme is slightly different from those used for last names. As can be seen there, approximately one-third of the first name pairs in our sample are not good matches. Another 31% were assigned a score of "20" indicating that the match is very likely- there is just a minor misspelling or even the inclusion of a space in one name and not the other (e.g., Maryjo vs. Mary Jo). Another common discrepancy is the use of a nickname rather than a full name (John/Johnny). One discrepancy that occurs reflects the population in New Mexico. That is, we have a relatively high Hispanic population. In some instances, the same individual's name will be recorded as the Spanish version of the English name or vice versa. For example, one dataset may identify someone as Michael, and the other identifies the same person as Miguel. While people typically choose one version over the other, it is not unusual for people to go by a different version depending on the circumstances.

Table 2 - First name match score frequencies

First name match score	Description of match score	All cases	Soundex match	Soundex non-match
0	Not a match: Include names that are clearly different (Joseph/Jolene; Clarence/Clarise; Joseph/Jordan; Joseph/Jacob; Jerry/Randy), even if they begin with the same letter.	33.6%	14.9%	43.2%
20	Probable match: minor misspelling (John/Jon), one letter off or very common misspellings (Stephen/Steven). This does not include names that are distinct names in their own right (Andres/ Andrea; Adam/Adan). While distinct names could be spelling errors, these are be coded differently (see 32 below).	30.7%	51.2%	20.1%
21	Probable match: names that are very similar in Spanish and English (Christine vs. Christina; Julie vs. Julia; Robert vs. Roberto; Thomas vs. Tomas). Include only those that suggest a spelling error. Do not include Spanish versions of names that are significantly different from English (Juan/John), which do not suggest a spelling error.	1.6%	3.5%	0.6%
22	Probable match: suffix (one or the other has jr., sr., III, etc.).	0.7%	0.6%	0.7%
23	Probable match: middle name included one of the first name variables (ex: Richard Joe vs. Richard).	7.8%	3.9%	9.8%
24	Probable match: names reversed (middle name and first name reversed; last and first name reversed).	0%	0%	0%
25	Probable match: nicknames (e.g., Johnny vs. John); diminutive version of names (Isabel, Isabella).	11.0%	3.7%	14.7%
30	Possible match: same name has very different spelling (EG, Lewis/Louis, Damian/Damien, Kaytlin/Caitlin), not just a spelling error.	3.9%	6.7%	2.4%
31	Possible match: Spanish version of English name or vice versa (Juan/John; Esteban/Steven) that are clearly not a spelling error.	3.9%	4.0%	3.8%
32	Possible match: similar names but not the same (Lee versus Leo; Adam vs. Adan.	4.9%	6.1%	4.3%
33	Possible match: Names that are similar, but correspond to a different gender (could be a misspelling, but could be a completely different name). E.g., Angela vs. Angelo or Angel/Angelo	2.0%	5.3%	0.4%

Procedures

Staff calculated string distances using the *stringdist* package in R. The matched pair data was exported from SPSS as a .csv and read into RStudio. We used a modified version of Raffael Vogler's R script[4] to

[4] https://www.joyofdata.de/blog/comparison-of-string-distance-algorithms/, accessed 12/5/2018

generate the string distances for a range of string-distance algorithms. The string distance scores for each match pair were exported as .csv and imported back into SPSS (see instructions in Appendix A).

Next, the investigators conducted exploratory analyses to identify string distance algorithms (e.g., Jaccard, OSA), and combinations of algorithms that correlated with various match scores. In the results section below, rather than include all analyses conducted, we describe only the measures we found that best correlate with the manual match scores.

Results

We compared the various algorithms against the manual check of the data to assess which measure, combinations of measures, and cutoff scores were associated with each match score. We analyzed the first name and last name match pairs separately, and found slightly different results for each. Our analyses focused on minimizing errors (either false positives or false negatives) and maximizing true positive and negative results. Our goal was to include measures that had an error rate of 2% or less; we chose this error rate because it is approximately the same as the manual match error rate. Regardless of whether we assessed first or last name pairs, we found that we could not rely on a single measure to effectively identifying most matches and non-matches. Rather, combinations of measures were best.

Last Names

We found seven measures, described below, that appear to identify the majority of good and non-matches.[5] The error rate in this sample for each measure varied from a low of 0% to a high of 2.2% (just slightly higher than the target error rate).

Likely true matches

We found three measures or combination of measures provided results that were likely true matches. Table 3 summarizes the results; the sections that follow describe each measure.

Table 3 – String distance measures for likely true last name matches

		Measure 1	Measure 2	Measure 3
Last name match score	N	LCS <=2	LCS >= 3 & diff length last names and qgram 2 <=1	OSA <=1 and Jarro <=.12
0	1607	2%	2%	0%
21	442	82%	<1%	95.3%
22	589	1%	88%	0%
3	242	14%	10%	4.7%
N	2880	509	619	379

[5] We identified some other measures that also identified good and bad matches, but the number of cases captured by these measures was smaller and overlapped with the measures we describe. Measures that identified good matches were: cosine set at 1 or 2 with a value <=.2; Jaccard set at 1 with a value <=.30; and Jaccard set at 2 with a value <=.40. Measures that identified bad matches were: cosine set at 0 >= .75; and Jaccard set at 0 with a value >= .80.

1) *LCS less than or equal to 2*

The first of these is LCS with a cutoff less than or equal to 2, indicating that two or fewer transformations are required to change one name to another. With this cutoff, we identified 98% of likely or probable matches, with an error rate of 2%. As might be expected, most of the matches made were "21," which is a minor misspelling or similar error. Approximately 14% of the cases with this cutoff were associated with a manual score of "3" or a probable match.

1) *LCS >=3 and difference between length of last 2 names and qgram 2 <=1*

Last names have unique characteristics that make it challenging to identify a match when one likely exists. For example, many people have hyphenated or compound surnames. One data source may list only one of the names, while the other lists both. This second measure appears to work well for identifying names like this.

The measure incorporates the difference in the length of last names and compares that to the q-gram set at 2 (bigram comparisons). Specifically, when the difference between the last names and the bigram comparison is less than or equal to one and the LCS is greater than or equal to 3, this measure is positive. The vast majority of cases that fall into this category are at least probable matches, and 88% of the cases in this category correspond to a manual match score of "22," indicating a hyphenated name in one data source corresponding to a single last name in the second source.

Since both measures 1 and 2 use different LCS cutoff scores in the calculation, the two measures are mutually exclusive. Thus, they identify different sets of good matches.

2) *OSA <=1 and Jarro (or JW0) <=.12.*

The final measure is OSA <=1 with a JW set at 0 that is less than or equal to .12. All of the cases were at least a probable match, with 95% of those associated with manual match score of "21" indicating a very minor misspelling or difference. Although this measure was impeccable, the number of cases that fell into this category was lower than those cases captured by the other two measures.

Combining true match results for last names

In practice, we would combine these results. Since measure 3 was never associated with a non-match, we would use this to identify probable matches, with possible matches (those in which we have slightly less confidence) falling into the measure 1 but not captured in measure 3. The second measure never overlaps with the other two measures. This measure we would consider a likely match for hyphenated names (probable match) and a possible match for names that don't include a hyphen. The decisions are summarized below:

Table 4 – Combined string distance measures for true last name matches

	OSA <=1 & JW <=.12 (measure 3)	
	In this category	**Not in this category**
Yes, LCS <=2 (measure 1)	379- probable matches	130- possible matches
Yes, LCS >=3 diff qg2 <=1 (measure 2)	0	619- If hyphenated: probable matches for compound names If not hyphenated: Possible match

Likely non-matches

Of the name pairs left after calculating the measures above, manual checks identified 90% as non-matching. The error rate of false negatives with just the measures above would be 10%, which is above our threshold of 2%. Thus, we explored combinations of algorithms that would better identify cases that were likely non-matches. We found five such measures.

Table 5 – Measures for predicted non-matching last names

		Measure 1	Measure 2	Measure 3	Measure 4	Measure 5
Last name match score	N	differences between length of names and LCS, dl, or qgram2 is >=6 for at least 2 of these measures	JW with penalty of .10 is .3 or higher and diff in length of names is 4 or less	Qgram1 >= 5 and diff last names <=3	LCS >= 4 & 4 or more of Cosine or jaccard =1	Jaccard 0 >=.80
0	1607	98%	97.8%	99.0%	99.7%	100%
21	442	1%	.6%	.1%	0%	0%
22	589	<1%	.8%	.1%	0%	0%
3	242	1%	.9%	.8%	.3%	0%
N	2880	1054	906	895	749	416

1) *Differences between length of names and lcs, dl, or qgram2 is >=6 for at least 2 of these measures*

The first of these measures calculates the difference between the lengths of the two names and LCS, DL, or Q-gram set at 2. If two or more of these measures is greater than or equal to 6, this measure is computed as "yes" (likely wrong). With this measure, we found a very low error rate: 2% false negative.

2) *Jaro-Winkler with a penalty of .1 is .3 or higher and difference in length of names is 4 or less*

The second measure of likely non-matches uses the Jaro-Winkler statistic with a penalty set at .10. If this results in a score of .3 or higher and the difference in the length of the names is 4 or less, the measure is computed as "yes." The false negative rate is slightly higher than the first measure, at 2.2%, but still very low.

3) Q-gram set at 1 >= 5 and difference in the length of last names <=3

This measure flags those cases that have a q-gram set at 1 (checking each letter) with a value of 5 or higher (5 or more letters differ) and have a difference in the length of the last names that is less than or equal to 3. Using this measure, there is a false negative rate of 1%; in other words, this measure correctly classifies 99% of the cases that fit these criteria.

4) LCS >= 4 & 4 or more of Cosine or Jaccard at any q-gram =1

A fourth measure makes use of multiple algorithms. We calculate this measure is "yes" if LCS is 4 or higher, and four or more of the following have a value of 1: cosine, jaccard (at 1 2 or 3). This means that among those names that take 4 or more steps to convert, once we compare pairs (ab, bc) or groups of three (abc, bcd) there are no common groups. Thus, it makes sense that the error rate would be

minimal; it is just .3% for cases coded as a possible match. One could omit the restriction of an LCS >=4; the error rate would just slightly increase to .8%. However, the number of cases is not that different (756 vs. 749) and increasing the number of cases increases the error.

5) *Jaccard 0 >=.80*

Finally, the Jaccard measure set at 0 compares all letters that are the same relative to the union of all letters. Using a cutoff of .80 or higher, all of the cases in this set are non-matches. However, fewer cases fall into this category than the other measures. Thus, while the measure outperforms all the others, this measure alone will not account for the majority of non-matching names.

Combining string distance measures to identify non-matches for last names

Unlike the first two "true match" measures, all of these "non-match" measures overlap. When all measures were taken into account (i.e., one or more "non-match" measures indicated a non-match), he error rate was 2.4%. As one might expect, the accuracy increased with the number of measures that match. When just one measure indicated a non-match, the error rate was 7.1%. This was lower for cases identified by 2 measures, at 4.3%. At 3, the error rate declined to just 1.2%, and if 4 or 5 measures indicated a non-match, the error rate was 0% (see table below).

Table 6 – Combined measures for non-match last name matches

	Possible non-match		Probable non-match		
	Number of measures predicting non-matches				
Last name match score	**1.00**	**2.00**	**3.00**	**4.00**	**5.00**
0	92.9%	95.7%	98.8%	100.0%	100.0%
21	0.4%	2.4%	0.0%	0.0%	0.0%
22	2.0%	0.8%	0.3%	0.0%	0.0%
3	4.7%	1.2%	0.9%	0.0%	0.0%
N	255	255	336	258	243

As seen above, the error rate with a single measure was higher than our target of 2%, but the error rate decreased with an increasing number of measures that predicted a non-match. Thus, in practice, we would identify those that match with just 1 or 2 measures as a possible non-match and those with a score of 3 or higher as a probable/likely non-match.

Confirmation of last name match scores

We confirmed the match criteria with small dataset of 71 non-perfect last name matches. The predicted true matches were perfect. The predicted non-matches had an error rate of 3.2%, higher than what we would like. However, like the results above, if we separate this variable into 1 or 2 measures versus 3 or more, we get an error rate of 0% for cases in the latter category. This suggests that we should identify probable and possible matches from this variable. We did not have any overlap in good and bad measures in this dataset.

Table 8 – Confirmation of last name match algorithms

	Predicted true match overall	Predicted possible true match	Predicted probable true match	Predicted non-match overall	Probable non-match	Possible non-match	No match made
0	0%	0%	0%	96.8%	100%	88.9%	64.3%
21	61.5%	66.7%	60.9%	0%	0%	0%	0%
22	30.8%	33.3%	34.8%	3.2%	0%	11.1%	28.6%
3	7.7%	0%	4.3%	0%	0%	0%	7.1%
N	26	3	23	31	22	9	14

First names

Like last names, first names have some unique problems. Perhaps among the most challenging are the use of a nickname in one dataset but not the other (e.g., Mike and Michael) and both Spanish and English versions of names used by an individual (e.g., Michael and Miguel). Another common but difficult problem is the inclusion of a middle name in the first name field that is absent from the other dataset (e.g., Mary Jo and Mary).

Likely true matches

We began our analysis by using the same combination of measures for true and non-matches as we did for last names. The results are in Table 9 below. As can be seen there, the measures of "true" matches were generally effective. Measure 1, LCS <=2, had a match rate of 98.5%, with an error rate of false positives at 1.5%; this measure included 1092 cases. As observed in the last names analysis, Measure 3 had an error rate of 0%, with 793 cases included in this category for the first name pairs. Measure 2 had the highest error rate at 2.9%. However, the majority of cases that were identified using this measure were "23" (one name includes the middle name, and the second does not such as Mary Jo vs. Mary) or "25" (one name is a nickname, such as Joe and Joseph).

Table 9 – String distance measures for likely true first name matches

	All	Predicted true match		
		Measure 1	Measure 2	Measure 3
First name match score	N	LCS <=2	LCS >= 3 & difference between length of names and qgram 2 <=1	OSA <=1 and Jarro <=.12
0	861	1.5%	2.9%	0%
20	783	66.6%	1.1%	75.0%
21	40	3.3%	0%	4.2%
22	17	.5%	1.3%	0%
23 (middle included)	199	3.6%	53.1%	1.6%
25 (nicknames)	280	4.0%	34.7%	1.8%
30	98	4.3%	0%	2.9%
31	99	1.3%	4%	0.3%
32	125	10.3%	0%	9.1%
33	51	4.7%	0%	5.2%
N	2553	1092	277	793

Combining likely true match results for first names

As we observed with last names, some of these measures overlap. In practice, we would combine these to create a "probable" match and "possible" match, which is slightly less likely to be a match. All 793 cases captured with Measure 3 were also captured with Measure 1. We would consider these probable matches. Those captured by Measure 1 only we would consider possible matches, as would those captured by Measure 2. These decisions are summarized below.

Table 10 – Combined string distance measures for true first name matches

	OSA <=1 & JW <=.12 (measure 3)	
	In this category	**Not in this category**
Yes, LCS <=2 (measure 1)	793- probable match	299- Possible match
Yes, LCS >=3 diff qg2 <=1 (measure 2)	0	277- Possible match

Likely non-matches

The predicted non-matches had a higher error rate (false negatives) than we observed with the last names. Whereas the error rate for Measure 2 (JW with a penalty of .10 is >= .3 and difference in length of names is 4 or less) is 2.2% for last names, here it is 5.9%. Measure 2 has the highest error rate, regardless of whether we are assessing last name pairs or first name pairs. For first names, the errors are primarily for cases categorized as "25" (nicknames versus full name) or "31" (Spanish/English pairs).

Table 11 - Measures for predicted non-matching first names

		Measure 1	Measure 2	Measure 3	Measure 4	Measure 5
First name match score	N	differences between length of names and lcs, dl, or qgram2 is >=6 for at least 2 of these measures	JW with penalty of .10 is .3 or higher and diff in length of names is 4 or less	LCS >=4 & 4 or more of cosine or jaccard = 1	Cosine set at 0 >= .75	Jaccard set at 0 >= .80
0	861	96%	94.1%	96.3%	99.2%	99.4%
20	783	0%	0%	.2%	0%	0%
21	40	0%	0%	0%	0%	0%
22	17	0%	0%	0%	0%	0%
23	199	0%	.4%	0%	0%	0%
25	280	1.9%	2.4%	1.2%	0%	0%
30	98	.2%	.3%	.3%	0%	0%
31	99	2.0%	2.8%	1.8%	0.8%	0.6%
32	125	0%	0	0%	0%	0%
33	51	0%	0	0%	0%	0%
N	2553	594	712	596	239	322

As we found for last names, when the number of non-match measures is three or more, the error rate is very low. However, for those that have only one or two matches, the error rate is much higher. Neither the inclusion of Soundex matches nor the inclusion of gender matches helped to distinguish between good and non-matches (not shown in table below). Like last names, in practice, we would identify a probable non-match as those with three or more non-match measures and a possible non-match as those with only one or two non-match measures.

Table 12 – Combined measures for non-match first name matches

	Non-match first name				
	Possible non-match		Probable non-match		
First name match score	**1.00**	**2.00**	**3.00**	**4.00**	**5.00**
0 (not a match)	62.6%	89.9%	98.4%	98.9%	99.5%
20s (probable match)	21.2%	3.4%	1.0%	0%	0%
30s (possible match)	16.2%	6.7%	0.5%	1.1%	0.5%
N	99	179	191	87	217

Confirmation of matches

As we did with the last names, we confirmed whether these measures worked with a second dataset. The overall error rates for first names were: 23% for predicted true matches (false positives) and 8.7% for predicted non-matches (false negatives). However, none of the "probable" matches for either good or non-matches had known false positives or false negatives (the error rate was 0%). Further, most of the probable true matches fell into the manual match 20's category (likely match) rather than the 30s category (possible match).

Table 13 – Confirmation of first name match algorithms

First name match score	Predicted true match overall	Probable true match	Possible true match	Predicted non-match overall	Probable non-match	Possible non-match	No match made
0	23.5%	0%	42.9%	91.3%	100%	87.1%	64.1%
20s	51.0%	73.9%	32.1%	0%	0%	0%	10.3%
30s	25.5%	26.1%	25.0%	8.7%	0%	12.9%	25.6%
N	51	23	28	46	48	31	39 (28.7% no match made)

Conclusion

We found multiple measures and combinations of measures best identify likely matches and likely non-matches. While the "possible" measures worked less reliably for first names when tested on a small confirmation dataset, the "probable" measures worked very well. This indicates we would have to be very careful with the possible matches. Typically, we use a combination of criteria to decide whether

there is a true match (names, DOB, SSN). We would flag “probable” matches to check if any of the other variables (DOB, SSN) do not match.

There are likely other combinations of measures that we did not identify here that could be used. Our goal was to create measures that would identify matches and non-matches while minimizing false positives and false negatives while identifying as many cases as possible as a match or non-match. The proportion of cases that did not result in a likely positive or negative choice ranged from 15% in the first name dataset to 20% of last names. As we work more with string distance measures, we may discover reliable cutoffs that will minimize the unknown rate and improve the error rate. Despite this, calculating the string distance measures is a relatively simple process, and by using SPSS syntax, we can create variables to categorize our decisions based on these measures. This process will save us many hours of manual checking of data.

Appendices

Appendix A – Sample Soundex Matching SPSS Code

```
*This set of code creates a last name variable that is all uppercase*
STRING  lastnamefixed (A23).
COMPUTE lastnamefixed= (ltrim(rtrim(upcase(lname)))).
EXECUTE.
STRING  LN1 (A1).
COMPUTE LN1=CHAR.SUBSTR(lastnamefixed,1).
EXECUTE.

*this next set of commands is to get rid of any leading non-alphabet characters*
STRING  lastnamefixed2 (A23).
compute lastnamefixed2=Char.Substr(lastnamefixed,2).
execute.
do if (LN1='A').
compute lastnamefixed2=lastnamefixed.
else if (LN1='B').
compute lastnamefixed2=lastnamefixed.
else if (LN1='C').
compute lastnamefixed2=lastnamefixed.
else if (LN1='D').
compute lastnamefixed2=lastnamefixed.
else if (LN1='E').
compute lastnamefixed2=lastnamefixed.
else if (LN1='F').
compute lastnamefixed2=lastnamefixed.
else if (LN1='G').
compute lastnamefixed2=lastnamefixed.
else if (LN1='H').
compute lastnamefixed2=lastnamefixed.
else if (LN1='I').
compute lastnamefixed2=lastnamefixed.
else if (LN1='J').
compute lastnamefixed2=lastnamefixed.
else if (LN1='K').
compute lastnamefixed2=lastnamefixed.
else if (LN1='L').
compute lastnamefixed2=lastnamefixed.
else if (LN1='M').
compute lastnamefixed2=lastnamefixed.
else if (LN1='N').
compute lastnamefixed2=lastnamefixed.
else if (LN1='O').
compute lastnamefixed2=lastnamefixed.
```

```
else if (LN1='P').
compute lastnamefixed2=lastnamefixed.
else if (LN1='Q').
compute lastnamefixed2=lastnamefixed.
else if (LN1='R').
compute lastnamefixed2=lastnamefixed.
else if (LN1='S').
compute lastnamefixed2=lastnamefixed.
else if (LN1='T').
compute lastnamefixed2=lastnamefixed.
else if (LN1='U').
compute lastnamefixed2=lastnamefixed.
else if (LN1='V').
compute lastnamefixed2=lastnamefixed.
else if (LN1='W').
compute lastnamefixed2=lastnamefixed.
else if (LN1='X').
compute lastnamefixed2=lastnamefixed.
else if (LN1='Y').
compute lastnamefixed2=lastnamefixed.
else if (LN1='Z').
compute lastnamefixed2=lastnamefixed.
end if.
execute.

* break the name into characters/individual variables, make the first letter the first character of soundex string.*
string a1 to a23 (a1) soundex1 (a23).
do repeat a=a1 to a23/b=1 to 23.
compute a=substr(lastnameFixed2,b,1).
end repeat.
execute.

* add numbers to soundex string.*
* drop spaces, H, W and non-alpha characters which were recoded to '' .*
compute soundex1=a1.
recode a2 to a23  ('A', 'E', 'I', 'O', 'U', 'Y' = '0')('B', 'F', 'P', 'V' ='1')
  ('C', 'G', 'J', 'K', 'Q', 'S', 'X', 'Z' = '2')
  ('D', 'T' = '3')('L' = '4')('M', 'N' = '5')('R' = '6')(else='').
execute.
do repeat a=a2 to a23.
if a ~= '' soundex1=concat(ltrim(rtrim(soundex1)),a).
end repeat.
execute.
```

```
* Now, combine any double numbers into a single instance of that number.*
string pl cl (a1) soundex2 (a23).
loop x=1 to 23.
compute cl=substr(soundex1,x,1).
if cl ~= pl soundex2=concat(ltrim(rtrim(soundex2)),cl).
compute pl=cl.
end loop.
execute.

* Further, if the first number in the Soundex value is the same as the code number for
* the initial letter, delete the first number*
string codea1 (a1).
compute codea1 = A1.
Recode codea1 ('A', 'E', 'I', 'O', 'U', 'Y' = '0')('B', 'F', 'P', 'V' ='1')
  ('C', 'G', 'J', 'K', 'Q', 'S', 'X', 'Z' = '2')
  ('D', 'T' = '3')('L' = '4')('M', 'N' = '5')('R' = '6')(else='').
execute.

string soundex3 (a20).
compute soundex3=soundex2.
if CODEA1=a2
soundex3=concat(substr(soundex2,1,1),substr(soundex2,3)).
execute.

* Now, remove all zeros from the Soundex string.*
string soundex4 (a20).
loop x=1 to 20.
compute cl=substr(soundex3,x,1).
if cl ~= '0' soundex4=concat(ltrim(rtrim(soundex4)),cl).
end loop.
EXECUTE.

* Finally, return the first four characters of the end product as the Soundex encoding.
* If there are less than four characters to be returned, concatenate enough zeros to make the length
four.*
string soundexlast (a4).
compute soundexlast=soundex4.
if length(ltrim(rtrim(soundexlast)))=3 soundexlast=concat(ltrim(rtrim(soundexlast)),'0').
if length(ltrim(rtrim(soundexlast)))=2 soundexlast=concat(ltrim(rtrim(soundexlast)),'00').
if length(ltrim(rtrim(soundexlast)))=1 soundexlast=concat(ltrim(rtrim(soundexlast)),'000').
execute.
*drops extra variables from file*
match files file=*/drop=lastnamefixed
```

```
LN1 a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13
       a14 a15 a16 a17 a18 a19 a20 a21 a22 a23 soundex1
       pl cl soundex2 x codea1 soundex3 soundex4.
execute.
```

Appendix B – Match Quality Manual Review Tables

Table B.1 – Manual first name match scores

First name	Perfect - 1	1 - Perfect match, 1-to-1 correspondence between characters
	Probable - 2	20 - Probable match: minor misspelling (John/Jon), one letter off or very common misspellings (Stephen/Steven). Don't include names that are distinct names in their own right (Andres/ Andrea; Adam/Adan).
		21 - Probable match: names that are very similar in Spanish and English (Christine vs. Christina; Julie vs. Julia). Include only those that suggest a spelling error. Do not include Spanish versions of names that are significantly different from English (Juan/John), which do not suggest a spelling error
		22 - Probable match: suffix (one or the other has jr., sr., III, etc.)
		23 - Probable match: middle name included in the first name variable and but not the other first name variable (ex: Richard Joe vs. Richard)
		24- Probable match: names reversed (middle name and first name reversed; last and first name reversed)
		25 - Probable match: nicknames (e.g., Johnny vs. John); diminutive version of names (Isabel, Isabella)
	Possible - 3	30- Possible match: same name has very different spelling (EG, Lewis/Louis, Damian/Damien, Kaytlin/Caitlin), not just a spelling error
		31 -Possible match - Spanish version of English name or vice versa (Juan/John; Esteban/Steven) that are clearly not a spelling error
		32- Possible match- similar names but not the same (Lee versus Leo; Adam vs. Adan, could be misspelling but if these could be a name in their own right, code here
		33- Possible match- Names that are similar, but correspond to a different gender (could be a misspelling, but could be a completely different name). E.g., Angela vs. Angelo or Angel/Angelo
	Not good - 0	0 - Not a true match. Include names that are clearly different (Joseph/Jolene; Clarence/Clarise; Joseph/Jordan), names more than three characters apart; names that are clearly not a spelling error
	Missing - 88	88 – Info missing from one or both fields

Table B.2 – Manual last name match scores

Last name	1 - Perfect match	1 - Perfect match. 1-to-1 correspondence. Include here differences in punctuation (such as Smith-Jones and Smith Jones) as the name is the same.
	2 - Probable match	21 - Probable match (e.g., minor misspelling, hyphenated names in which the order is switched: jones-smith vs. smith-jones)
		22 - Probable match (hyphenated name/one part of name matches, Jones-Smith vs. Jones; suffix in one and not the other, e.g., Jones III vs. Jones)
	3 - Possible match	3 - Possible match (e.g., could be a true match, but not entirely sure (ex: Martin vs. Martinez)
	0 - Not a true match	0 - not a true match (clearly different names, names more than three characters off)
	88 - missing information	88 - info missing from one or both fields

Appendix C– String Distance Algorithm Procedure

Part 1 – Running the R Script

1. Export matched records from SPSS as .csv file. Since the original datasets were matched by Soundex first and last names, we will want the original first and last name variables.
2. Read into RStudio (Under "Environment" tab in upper-right quadrant, click "import dataset" -> to read in a .csv, click "From Text (base)" -> select the file -> if first row of .csv is column names, select "Heading: Yes" -> shouldn't have to change any of the defaults options for the other settings, so click "Import").
3. Open the R script.
   A sample script for last names is as follows:

```
lastsamp <- new.name.set.to.verify.algrthms$last1
lastkey <- new.name.set.to.verify.algrthms$last2
getwd()
setwd("")
#####Comparing string similarities for last names
M <- data.frame(
  m = c("osa", "lv", "dl", "lcs", "qgram", "qgram", "qgram",
       "cosine", "cosine", "cosine", "jaccard", "jaccard", "jaccard",
       "jw", "jw", "jw"),
  q = c(0,0,0,0,1,2,3,1,2,3,1,2,3,0,0,0),
  p = c(0,0,0,0,0,0,0,0,0,0,0,0,0,0.1,0.2)
)

R <- apply(M, 1,
         function(m) stringdist(lastsamp, lastkey, method=m["m"], q=m["q"], p=m["p"]))

R2 <- round(R,3)

rownames(R2) <- paste(format(paste("'", lastsamp, "'", sep=""), width=14), " - ",
                  format(paste("'", lastkey, "'", sep=""), width=17), sep=""
)

colnames(R2) <- M$m
write.csv(R2, "lastnamesnew.csv")
```

Part 2 – Getting the String Distances Back into SPSS

1. The .csv files created by R in the previous part will have the records all in the same order as they were in the original SPSS dataset, assuming that order was preserved in the .csv file exported from SPSS. (If you didn't sort the files at any point in this process, the records will be in the same order throughout.) As such, the first record in "lastnames" corresponds to the first record in "firstnames" corresponds to the first record in the original SPSS dataset; the second record in "lastnames" corresponds to the second record in "firstnames" corresponds to the second record

in the original SPSS dataset; … ; the nth record in “lastnames” corresponds to the nth record in “firstnames” corresponds to the nth record in the original SPSS dataset.

Because of this, it’s easy to assemble all the data in one dataset via a number of techniques. Depending on preference and intent, you could try one of the following examples, though there are plenty more ways to do it:

Example 1. Read “lastnames.csv” and “firstnames.csv” into SPSS. Generate a variable in each of those datasets consisting of a sequence of increasing integers (ie, 1, 2, …, n), such that corresponding rows in both datasets have the same unique integer value associated with them. Create this variable identically in the SPSS dataset containing the original records. Since each integer will be uniquely associated with the same record in each dataset, you can merge by this variable to create a dataset containing the data from the original SPSS file, from “lastnames.csv,” and from “firstnames.csv.”

Example 2. In the original SPSS file, insert new variables, one for each string distance algorithm, and simply copy the values from the corresponding column in the .csv into the SPSS file. SPSS won’t love this for large datasets, but it will comply. NOTE: if using this method with the .csv files open in Excel, do not copy the column name from the .csv file. Excel puts column names in the first row of data, whereas SPSS stores column names separately from the data.

2. Check a few cases to confirm that the correct string distances are associated with the corresponding records.
3. You now have an SPSS dataset containing all the original data, along with the string distance algorithms for each pair of first and last name variables in the dataset. If it looks good, click save.