Paper 7760-2016

Fuzzy Matching: Where Is It Appropriate and How Is It Done? SAS® Can Help.

Stephen Sloan, Accenture

Dan Hoicowitz, Accenture Federal Services

ABSTRACT

When attempting to match names and addresses from different files, we often run into a situation where the names are similar, but not exactly the same. Sometimes there are additional words in the names, sometimes there are different spellings, and sometimes the businesses have the same name but are located thousands of miles apart. The files that contain the names might have numeric keys that cannot be matched. Therefore, we need to use a process called "fuzzy matching" to match the names from different files. The SAS® function COMPGED, combined with SAS® character-handling functions, provides a straightforward method of applying business rules and testing for similarity.

INTRODUCTION

Our challenge

Match customer name-and-address databases that might have thousands or millions of rows and choose likely matches without having key fields to use to join the databases.

Our tool

We used character-handling functions and the COMPGED word-comparison function available in SAS®.   Sample code will be supplied on request.

Six Step Approach to Cleansing the Data and Using Fuzzy Logic

Step 1 – Remove extraneous characters

As a general rule, punctuation can differ while the names are the same. For example, John's "super" pizza and John's super pizza refer to the same restaurant. Therefore, we removed the following characters from all names: ' " & ? -

Step 2 – Put all characters in upper-case notation and remove leading blanks

Step 3 – Remove words that might or might not appear in the same company name.

Some examples are The, .com, Inc, LTD, LLC, DIVISION, CORP, CORPORATION, CO., and COMPANY.

Step 4 – Rationalize the zip codes when matching addresses

We found it useful to remove the last 4 digits of 9-digit zip codes, because some files might only have 5-digit zip codes. Since some files might have zip codes as numeric fields, and other files might have zip codes as character fields, make sure to include leading zeroes. For example, my home zip code is 08514. As a numeric field, it would appear as 8514 and the zero would need to be appended.

If working with US zip codes, make sure they are all numeric. This does not always apply for other countries. One common mistake to watch for is that sometimes Canada, with abbreviation CA, is put in as the state CA (California) instead of the country CA. Since Canada has an alphanumeric 6-character zip code, this will be caught when checking for numeric zip codes.

Step 5 – Choose a standard for addresses

Decide whether to use Avenue or Ave, Road or Rd, etc, and then convert the address fields to match the standard.

Step 6 – Match the names

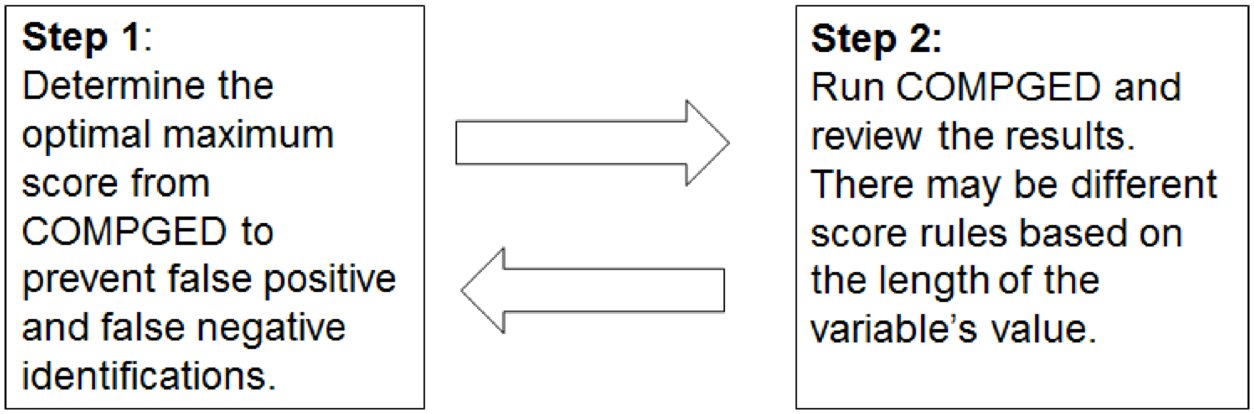
SAS® COMPGED Function

The COMPGED function provides a score when comparing two text strings. The score is called the Generalized Edit Distance (GED). The higher the computed GED, the less likely the two strings match. Zero = perfect match.

COMPGED enables the matching records from different data sets to be compared without a unique identifier, by computing the generalized edit distance or score between the two character values. In words, COMPGED computes a measure of dissimilarity between two strings. We followed an iterative process when using COMPGED, looking at samples of matches and non-matches and minimizing both false positives and false negatives.

Here is how it works:

Take a Cartesian join of the names in the two (or more) files being compared and use the SAS® COMPGED function to compute a score. Then determine how high a cutoff you will allow. We chose to allow higher scores for longer names (i.e. if two three-character names differed by one character, that would be more significant than if two fifteen-character names differed by one character). We also reduced the length used for this calculation by the length of certain words that were generic to the line of business we were evaluating. Some examples are PRINTING, SYSTEMS, and SOLUTIONS.



Step 1: Determine the optimal maximum score from COMPGED to prevent false positive and false negative identifications.

Step 2: Run COMPGED and review the results. There may be different score rules based on the length of the variable's value.

SAS® COMPGED Approach

The following is a list of musicians and their names, birth dates, and ID numbers.

Match both data sets on Name and National Identification Number (NINO). Objective is to determine the optimal maximum score from COMPGED .

Data Set 1

Name	**NINO**	**DOB**
John Baldwin	DSFAS	1/3/1946
Robert Plant	ASFF	1/4/1947
James Page	DFA	1/5/1948
John Bonham	DFAD	1/6/1949
Ray Davies	HKK	1/7/1947
Dave Davies	HUYF	1/8/1947
Peter Quaife	DSASF	1/8/1950
Mick Ivory	MOIUA	1/9/1950

Data Set 2

Name	**NINO**	**DOB**
John Baldwin	DSFAS	1/3/1946
Robert Plant	ASFF	1/4/1947
Jimmy Page	DFA	1/5/1948
John Bonham	DFAD	1/6/1949
Ray Davies	HKK	1/7/1947
Dave Davies	HUYF	1/8/1947
Peter Quaife	DSASF	1/8/1950
Mick Ivory	MOIUA	1/9/1950

To calculate the edit distance between a set of two strings, COMPGED works to convert the first string into the second string.   Each operation (insertion, replacement, deletion) is added up step by step with the lowest cost method determining the score.

Data Set 1			
First Name	Last Name	NINO	DOB
James	Page	DFA	1/5/1948
Data Set 2			
First Name	Last Name	NINO	DOB
JIMMY	Page	DFA	1/5/1948

Comparing First Name from Data Set 1 (variable name FIRSTNAME) to First Name in Data Set 2 (variable name FIRSTNAME2) requires the replacement of three characters, the 2nd, 4th, and 5th, as James goes to Jimmy.

```
SCORE=COMPGED(FIRSTNAME,FIRSTNAME2,'INL');
```

The score came to 300, because SAS® would conduct three replace operations. Please see below for the definition of the code and how the score is computed.

First Name	Last Name	NINO	DOB	Score
James	Page	DFA	1/5/1948	**300**

If we did not specify that COMPGED would ignore the case, the score would be 400 because the letter m in James would not match the letter M in JIMMY. See the syntax description below for how the parameter INL influenced the treatment of upper and lower case.

```
SCORE=COMPGED(FIRSTNAME,FIRSTNAME2);
```

First Name	Last Name	NINO	DOB	Score
James	Page	DFA	1/5/1948	**400**

The syntax for the COMPGED function, extracted from SAS® the Help screen for the COMPGED function, is below:

COMPGED(*string-1*, *string-2* <,*cutoff*> <,*modifiers*>)

Required Arguments

string–1

specifies a character constant, variable, or expression.

string-2

specifies a character constant, variable, or expression.

Optional Arguments

cutoff

is a numeric constant, variable, or expression. If the actual generalized edit distance is greater than the value of *cutoff*, the value that is returned is equal to the value of *cutoff*.

Tip

Using a small value of *cutoff* improves the efficiency of COMPGED if the values of *string–1* and *string–2* are long

modifiers

specifies a character string that can modify the action of the COMPGED function. You can use one or more of the following characters as a valid modifier:

i or I	ignores the case in *string–1* and *string–2*.
l or L	removes leading blanks in *string–1* and *string–2* before comparing the values.
n or N	removes quotation marks from any argument that is an n-literal and ignores the case of *string–1* and *string–2*.
: (colon)	truncates the longer of *string–1* or *string–2* to the length of the shorter string, or to one, whichever is greater.

The scoring for the SAS® COMPGED function is below, followed by examples of how scores would be computed. Both are available in the Help screen for the COMPGED function.

APPEND	50	When the output string is longer than the input string, add any one character to the end of the output string without moving the pointer.
BLANK	10	Do any of the following: Add one space character to the end of the output string without moving the pointer. When the character at the pointer is a space character, advance the pointer by one position without changing the output string. When the character at the pointer is a space character, add one space character to the end of the output string, and advance the pointer by one position. If the cost for BLANK is set to zero by the COMPCOST function, the COMPGED function removes all space characters from both strings before doing the comparison.
DELETE	100	Advance the pointer by one position without changing the output string.
DOUBLE	20	Add the character at the pointer to the end of the output string without moving the pointer.
FDELETE	200	When the output string is empty, advance the pointer by one position without changing the output string.
FINSERT	200	When the pointer is in position one, add any one character to the end of the output string without moving the pointer.
FREPLACE	200	When the pointer is in position one and the output string is empty, add any one character to the end of the output string, and advance the pointer by one position.
INSERT	100	Add any one character to the end of the output string without moving the pointer.
MATCH	0	Copy the character at the pointer from

		the input string to the end of the output string, and advance the pointer by one position.
PUNCTUATION	30	Do any of the following: Add one punctuation character to the end of the output string without moving the pointer. When the character at the pointer is a punctuation character, advance the pointer by one position without changing the output string. When the character at the pointer is a punctuation character, add one punctuation character to the end of the output string, and advance the pointer by one position. If the cost for PUNCTUATION is set to zero by the COMPCOST function, the COMPGED function removes all punctuation characters from both strings before doing the comparison.
REPLACE	100	Add any one character to the end of the output string, and advance the pointer by one position.
SINGLE	20	When the character at the pointer is the same as the character that follows in the input string, advance the pointer by one position without changing the output string.
SWAP	20	Copy the character that follows the pointer from the input string to the output string. Then copy the character at the pointer from the input string to the output string. Advance the pointer two positions.
TRUNCATE	10	When the output string is shorter than the input string, advance the pointer by one position without changing the output string.

Examples of the scoring in the SAS® COMPGED function, re-sorted from an example available in the Help screen for the COMPGED function.

Obs	String1	String2	Generalized Edit Distance	Operation
1	baboon	baboon	0	match
2	baboo	baboon	10	truncate
3	bab oon	baboon	10	blank
4	babboon	baboon	20	double
5	babon	baboon	20	single
6	baobon	baboon	20	swap
7	bab,oon	baboon	30	punctuation
8	baboonX	baboon	50	append
9	baXboon	baboon	100	insert
10	baoon	baboon	100	delete
11	baXoon	baboon	100	replace
12	aboon	baboon	120	trick question: swap+delete
13	baby	baboon	120	replace+truncate*2
14	bXaoon	baboon	200	insert+delete
15	bXaYoon	baboon	200	insert+replace
16	bXoon	baboon	200	delete+replace
17	Xbaboon	baboon	200	finsert
18	Xaboon	baboon	200	freplace
19	balloon	baboon	200	replace+insert
20	axoon	baboon	300	fdelete+replace
21	axoo	baboon	310	fdelete+replace+truncate
22	axon	baboon	320	fdelete+replace+single

CONCLUSION

When confronted with the need to match text fields without the benefit of identifying codes, the SAS® COMPGED function can achieve positive results for the organization when combined with judicious editing of the data and appropriate business rules about the required tightness of the match

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Stephen Sloan
Accenture
Stephen.b.sloan@accenture.com

Daniel Hoicowitz

Accenture Federal Services
Daniel.s.hoicowitz@acceturefederal.com