Paper 073-29

Using Edit-Distance Functions to Identify "Similar" E-Mail Addresses

Howard Schreier, U.S. Dept. of Commerce, Washington DC

ABSTRACT

Version 9 of SAS® software has added functions which can efficiently help to assess the degree to which different character strings are similar or have elements in common. These do not necessarily supersede the simpler functions provided in earlier versions. In some cases it makes sense to use old and new together.

INTRODUCTION

We want to summarize the archives of the SAS-L e-mail list by enumerating the message authors and the number of messages written by each. So we extract the e-mail addresses and use PROC FREQ or the equivalent to come up with the counts.

THE PROBLEM IN MINIATURE

The output, in part, looks something like this:

SendingAddress	Count
susan.sugigoer@anewjob.com	101
sasmaniac23@aol.com	3
samuel.sasmaniac@uvw.edu	55
sarah.sowhat@uvw.edu	44
sasmanis@vulture.uvw.edu	14
susan.sugigoer@uvw.edu	17
suzy1342@yahoo.com	14

The problem is that the observations represent distinct e-mail addresses rather than distinct individuals. People's addresses change and vary, for a number of reasons: new jobs; new internet service providers; "transparent" changes introduced by systems administrators. It's also common for individuals to have multiple addresses in simultaneous use.

So, accurate counts cannot be developed unless we can aggregate over multiple addresses which actually belong to the same individual. This is a variant of the oft-discussed "fuzzy merge" problem.

In this example, what we really want is something like:

SendingAddress	Count
susan.sugigoer@anewjob.com	132
samuel.sasmaniac@uvw.edu	72
sarah.sowhat@uvw.edu	44

where we have arbitrarily kept the address with the highest count to identify each individual. Since e-mail addresses can be cryptic, even better is a list identifying people by name:

LastName	FirstName	Count
Sugigoer	Susan	132
Sasmaniac	Samuel	72
Sowhat	Sarah	44

THE ACTUAL PROBLEM

The SAS-L e-mail list has been around since 1986, and more than twenty thousand distinct e-mail addresses have been used to post questions, answers, and comments over the years. The total number of postings is in the hundreds of thousands.

It is only realistic to accept that the aggregation process (1) cannot be completely automated; (2) cannot be done with complete accuracy; and (3) cannot, as a practical matter, be completed for all individuals represented in the data. Since the primary purpose is to identify the leading contributors in approximate rank order, we can accept (1) relatively small errors in the end results and (2) omission of any individual who posted only a small number of times.

Our goal is to develop a computer-supported manual research system. The computer will be expected to identify relatively small subsets of observations which seem like they might be e-mail "aliases" for various individuals.

THE FUNCTIONS

SAS now offers several functions (SPEDIS, COMPLEV, and COMPGED) which can aid in solving such problems by quantifying the degree to which two character values are similar. These supplement simpler functions (such as INDEX and SCAN) which can reveal whether or not substrings match.

COMPGED EXAMPLE

Here's an example:

```
data _null_;
ged = compged('baby','baboon'); put ged=;
ged = compged('balloon','baboon'); put ged=;
run;
```

When this code is run, the log displays:

```
ged=120
ged=200
```

In the first case, SAS transformed "baboon" to "babyon" and assessed a "cost" of 100 for the replacement of one character with another; then it truncated the string, first to "babyo" and finally to "baby", with a cost of 10 for each truncation. The total cost, 120, is the result returned by the function. In the second case, a replacement changed "baboon" to "baloon", again at a cost of 100; then insertion of a letter ("l") at the cost of 100 completed the process and brought the result to 200. The implication is that "baby" is more similar to "baboon" than is "balloon" to "baboon".

MORE ABOUT COMPGED

The COMPGED function has in its repertoire a number of other operations (for example, transposition of adjacent characters), each of which has a cost coefficient. The details, and more, are presented in the documentation. COMPGED also has an algorithm to guide it to the least-cost sequence of operations for any transformation.

Associated with the COMPGED function is the CALL COMPCOST routine, which allows you to change cost coefficients and even selectively disable operations. Use of this routine is optional. It does permit you to tune the COMPGED function for particular purposes.

COMPLEV

The COMPLEV function computes the so-called Levenshtein edit distance to compare two character strings. It counts the minimum number of single-character insert, delete, or replace operations needed to accomplish the transformation, and thus is essentially a streamlined special case of the COMPGED function. It is less versatile than COMPGED, but it executes faster.

SPEDIS

The SPEDIS (spelling distance) function has been part of SAS longer than the other two functions in this category. It is similar in nature to COMPGED, but less efficient in execution.

THE PROCESS IN GENERAL

We'll now turn to the process which I developed to aggregate, by author, message counts from the archives of SAS-L. This method for identifying groups of e-mail addresses belonging to individual SAS-L contributors involves a combination of computer processing and human research and analysis.

There are two elements to the human research and analysis. The first is to look at possible "aliases" as identified by the computer and select those which appear most plausible. The other is to use the Advanced Search screen of the Google™ Groups web site to find the real names (or in some cases partial names or screen names) corresponding to these selected addresses.

Triage is essential, to reduce both the human and computer processing burdens. Addresses used very few times are completely excluded, and among the remainder, priority is given to those addresses used the most. This vastly reduces the scale of the problem with little sacrifice in the scope or accuracy of the final results.

The process is iterative. The results of one cycle provide input for the next.

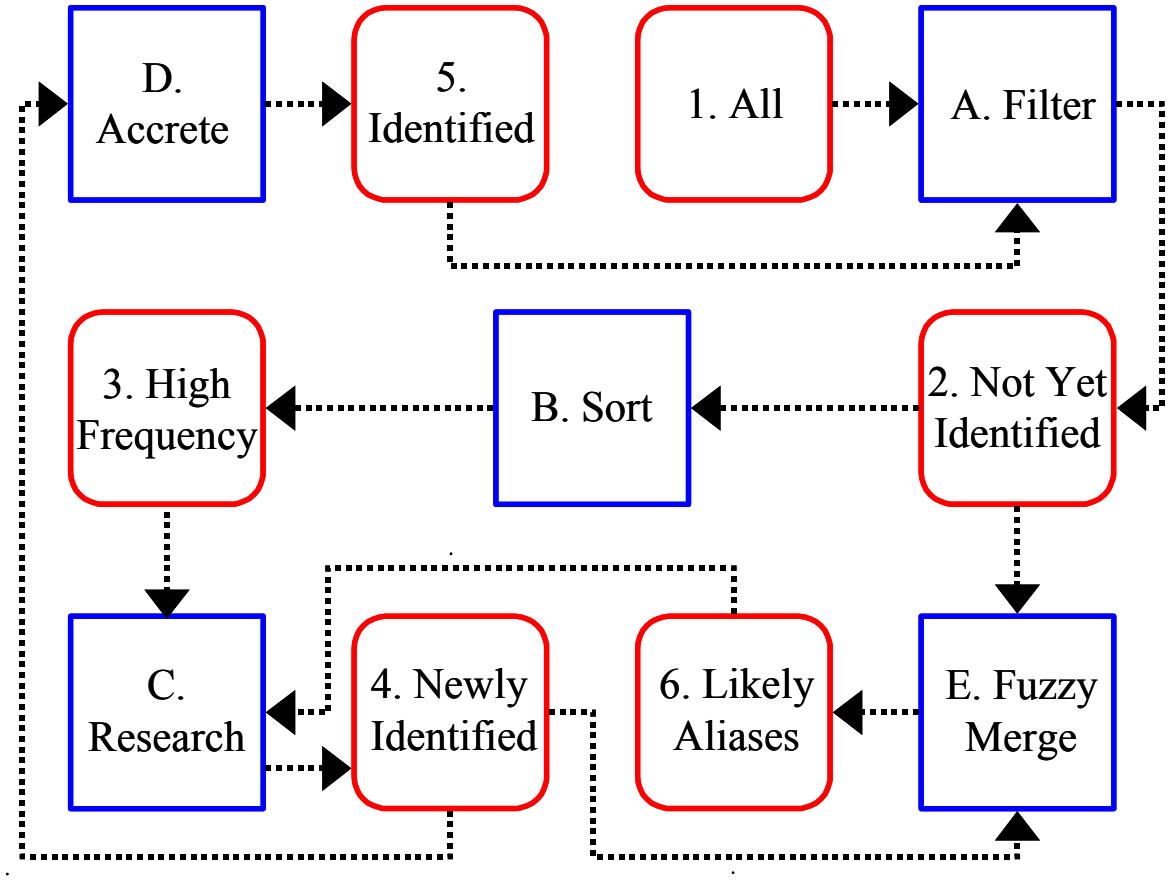
These techniques are hardly perfect. The mechanical matching generates many false positives and, presumably, false negatives. Even without such errors, the strategy would miss a person who posted a large number of messages by using many addresses a relatively few times each. But that seems an unlikely scenario, so barring major errors in the manual analysis, the results should be quite satisfactory.

THE PROCESS IN DETAIL

With the exception of the step which performs the "fuzzy merge", the SAS code is very straightforward and does not merit any special explanation. So we will begin with a schematic view of the entire process, then examine the code which is of particular interest.

FLOW CHART

Below is a flow chart depicting the process. The red, round-cornered boxes with labels identified by numerals are SAS datasets. The blue, square-cornered boxes with labels identified by letters of the alphabet are processing steps.



Several terms will be used repeatedly in the notes which follow:

- address: an e-mail address used to send messages to SAS-L between 1986 and 2003
- identification: the process of associating a person with an address, usually in terms of first name and last name, but occasionally in terms of just one or the other or a screen name
- alias: one of multiple addresses belonging to the same person
- frequency: number of SAS-L messages originated from an address.

Notes:

(1) This file (derived from the LISTSERV archives at Marist University) contains one observation for each distinct originating address (of which there are around 23 thousand), along with a count indicating the number of SAS-L messages from that address (November 1986 through December 2003). As might be expected, a small number of addresses are associated with a large number of postings. More than 10 thousand (or nearly half) of the addresses have been used exactly once, and almost 18 thousand have been used five or fewer times. On the other hand, there are perhaps 250 addresses associated with 100 or more postings each.

(A) This process removes the already-identified addresses (5) from the file of all addresses (1) to yield a file of not-yet-identified addresses. For practical reasons, it also removes addresses with very low frequencies. This greatly expedites other processes (C, E) at little cost in terms of the end results. On the first pass, there

are no identified addresses, so (5) is empty and only low-frequency addresses are removed.

(2) This file contains all addresses not yet identified except for those with very low frequencies, which theoretically should be included, but are excluded for practical reasons.

(B) This process sorts the list of not-yet-identified addresses by frequency, to isolate those with the highest frequencies. These higher-frequency addresses are generally more worth researching.

(3) This file exists to present addresses which are not yet identified but which have relatively high frequencies.

(C) This process entails manual research to identify the person to which a particular address belongs or belonged. Usually the identification is in terms of a first name and a last name, but in some cases it is just one or the other, or a screen name.

In some cases, identification may be obvious (eg, "samuel.sasmaniac@…"). In general however, it is not (eg, "suzy1342@…"), and the research is done using the Advanced Search screen of the Google Groups site.

It is not practical to research all addresses. Since the objective is to get reasonably accurate counts for the most active posters, two selection criteria are used to pick addresses to research

- high frequency
- similarity to previously identified addresses

Two inputs are available to guide the process. The "High Frequency" list (3) shows the not yet identified addresses with the highest frequencies; this is the only guidance available during the initial iteration. The "Likely Aliases" file (6) identifies addresses which seem (to the computer) similar to those already identified. It also includes frequencies for those addresses. The research task includes weeding out obvious false positives. Frequencies of the likely aliases may also be considered, but with a different threshold (since, for the ultimate purpose of the exercise, identifying an address used 30 times by somebody who has made 500 other postings using other addresses is more important than identifying an address used 60 times by somebody who has not used other addresses).

(4) This file lists only those addresses identified in the most recent cycle. This shrinks both the join in (E), saving computer processing time, and the results of the join (6), sparing the researcher the trouble of reviewing the same possible matches again and again.

(D) This process adds the most recent set of identified addresses to the list of those identified in previous cycles.

(5) This file contains the cumulative list of identified addresses. It is the end result of the whole exercise.

(E) This process performs a cartesian join pairing each newly identified address (4) with each unidentified one (2), then uses a formula employing the COMPGED function to quantify similarity. Thresholds, based on three scenarios, are applied to keep pairs with low "distances" in a list of likely aliases (6).

In making the comparisons, each e-mail address is broken into two components: the "box" (characters preceding the "@") and the "system" (characters following the "@"), after removing most non-alphabetic characters (on the theory that they reveal little about similarity among addresses).

(6) This file is the list of likely aliases, enumerating addresses which seem like they might belong to people who are already identified with other addresses. To the extent that it is complete (few false negatives), the ultimate results should be rather accurate.

CODE

These are the tables and columns mentioned in the SQL statements which perform the fuzzy merge.

Tables:

- NEWNAMES: newly identified addresses (4)
- NOTYETID: not yet identified addresses (2)
- POSSIBLE: likely aliases for the newly identified addresses, drawn from the not yet identified addresses; some repetition
- DISTINCT: likely aliases (6) for the newly identified addresses, drawn from the not yet identified addresses; no repetition

Columns:

- LAST: last name
- FIRST: first name
- EMAIL: e-mail address
- BOX: left portion of e-mail address, with digits and most special characters removed
- SYSTEM: right portion of e-mail address, with digits and most special characters removed
- BIG5: flag indicating that e-mail address is in one of five widely used domains (AOL, MSN, Compuserve, Deja, Yahoo)
- MANY: frequency
- BOX_BOX: COMPGED score for BOX values
- LAST_BOX: COMPGED score for a LAST value (from NEWNAMES) and a BOX value (from NOTYETID)
- SYSTEM_SYSTEM: COMPGED score for SYSTEM values

Here is the PROC SQL code used. First, the cartesian join is formed, and filtered:

```
create table possible as select
  last,
  first,
  notyetid.email,
  notyetid.box,
  notyetid.system,
  big5,
  notyetid.many,
  compged(newnames.box   ,notyetid.box   ,400)
    as box_box,
  compged(newnames.last  ,notyetid.box   ,300)
    as last_box,
  compged(newnames.system,notyetid.system,300)
    as system_system
 from notyetid, newnames
 where newnames.email ne notyetid.email
   and
       (
       /* Scenario: corporate change */
       calculated system_system < 300 and
       calculated box_box       < 400 and
       newnames.box ne '_'            and
       notyetid.box ne '_'            and
       not big5
       or
       /* Scenario: address resembles surname */
       calculated last_box < 300          and
       substr(newnames.last,1,3) ne '[?]' and
       notyetid.box ne '_'
       or
       /* Scenario: new job or ISP */
       calculated box_box < 200 and
       newnames.box ne '_'      and
       notyetid.box ne '_'
       )
;
```

These are the three scenarios underlying the filters which appear in the WHERE clause.

- A company (or university, or government agency) refines or upgrades its e-mail system. The result is that an address like sasmanis@vulture.uvw.edu is replaced with one like samuel.sasmaniac@uvw.edu. Since both components (left and right of the "@") ought to be similar, both BOX_BOX and SYSTEM_SYSTEM are in the filter.
- An individual at some times uses an address which does not reflect his or her surname, but at other times uses one which does include at least an approximation of the surname. Example: suzy1342@yahoo.com belongs to Susan Sugigoer, as does susan.sugigoer@uvw.edu. The test for this scenario is to compare NEWNAMES.LAST with NOTYETID.BOX. This approach is not symmetric (identifying susan.sugigoer@uvw.edu with Susan Sugigoer will not help with identification of suzy1342@yahoo.com).

- An individual gets a job, or a new job, or a new internet service provider. Example: susan.sugigoer@uvw.edu becomes susan.sugigoer@anewjob.com. To model this, only the left component of the address is considered, so BOX_BOX is in the filter.

The underscore character and the bracketed question mark are conventions used for missing values. Testing for them in the WHERE clause helps to reduce false positives, as does including the BIG5 flag for widely used e-mail domains such as yahoo.com.

The (numeric and optional) third argument to the COMPGED function is a cutoff which can avoid needless computation by terminating evaluation once the "cost" reaches the specified value. It is explained in the documentation.

The code as shown is fairly arbitrary. It was developed through trial and error. Additional tuning and elaboration could undoubtedly improve its efficiency and accuracy.

The table POSSIBLE includes duplicate address pairs. These are eliminated by means of the DISTINCT keyword:

```
create table distinct as select distinct
 last, first, email, many from possible
 order by last, first, many descending;
```

The table DISTINCT is the list of likely aliases (6).

CONCLUSION

COMPGED and similar functions for quantifying the degree of similarity between character values can be quite useful in semi-automated "fuzzy merge" processes.

REFERENCES

Google, *Advanced Groups Search*, http://www.google.com/advanced_group_search

Marist University, *Listserv logs for SAS-L*, http://vm.marist.edu/htbin/wa?LIST=SAS-L

SAS Institute Inc., *SAS OnlineDoc® 9*, http://v9doc.sas.com/sasdoc/

Schreier, Howard, *Ask and Ye Shall Receive: Getting the Most from SAS-L*, http://www.nesug.org/Proceedings/nesug03/gv/gv006.pdf

CONTACT INFORMATION

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

Howard Schreier
U.S. Dept. of Commerce
Mail Stop H-2015
Washington DC 20230

202-482-4180

Howard_Schreier@ita.doc.gov

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.