Paper 3142-2015

Fuzzy Matching

Arti Virkud, Data Analyst in NYC

ABSTRACT

Quality measurement is increasingly important in the health-care sphere for both performance optimization and reimbursement. Treatment of chronic conditions is a key area of quality measurement. However, medication compendiums change frequently, and health-care providers often “free text” medications into a patient’s record. Manually reviewing a complete medications database is time consuming. In order to build a robust medications list, we matched a pharmacist-generated list of categorized medications to a raw medications database that contained names, name-dose combinations, and misspellings.

The matching procedure used is called COMPGED. We were able to combine a truncation function and an upcase function to optimize the output of COMPGED. Using these combinations and manipulating the scoring metric of COMPGED enabled us to narrow the database list to medications that were relevant to our categories. This process transformed a tedious task for PROC COMPARE or an Excel macro into a quick and efficient method of matching. The task of sorting through relevant matches was still conducted manually, but the time required to do so was significantly decreased by the fuzzy match in our application of COMPGED.

INTRODUCTION

In this paper, the reader will gain a broad understanding of how COMPGED functions can be optimized to facilitate a “fuzzy matching” process. Fuzzy matches are methods where approximation functions are used to “closely” connect two lists of information. In the case of Compged, an exact match can be obtained if the threshold is restricted low enough.

While the COMPGED function can be run in a data step, this paper will use Compged in PROC SQL. Understanding PROC SQL will be helpful to understanding this particular matching procedure, but the code can be easily transferred to a data step.

FIRST MAIN TOPIC FUZZY MATCHING

To optimize use of COMPGED to fuzzy match, you can use variable modification functions (e.g. truncation functions, upcase functions, etc.) to enhance the method of matching.

Here is an example of a COMPGED function in PROC SQL:

```
Proc SQL;
Create Table work.merge1 as
  Select Distinct
    a.med as FirstMed,
    b.med as SndMed,
    Compged(a.med,b.med) as Score
  From Work.Firstlist as a Inner Join Work.Secondlist as b on
  Compged(a.med,b.med) LT 50;
Quit;
```

Standard COMPGED Fuzzy Matching

If you have a list of medications (Firstlist) that contains generic names of medications, and you want to match it to a list of medications (Secondlist) that contains specific names of medications or spelling errors, you will not be able to match the two lists using standard exact match methods. You may find, however, that portions of the names match exactly. If you truncate your variables to those strings, COMPGED can serve as an exact match.

Here is an example of truncating a variable prior to using COMPGED function to generate a match list:

```
Data Work.TFirstlist;
  Set Work.Firstlist;
    FirstMed_trunc = upcase(substr(med,1,6));
Run;

Data Work.TSecondlist;
  Set Work.Secondlist;
    SndMed_trunc = upcase(substr(med,1,6));
Run;

Proc SQL;
Create Table work.Merge_trunc6&suffix as
  Select Distinct
    a.med as FirstMed,
    a.FirstMed_trunc,
    b.med as SecondMed,
    b.SndMed_trunc,
    Compged(a.FirstMed_trunc,b.SndMed_trunc) as Score
  From work.TFirstlist as a Inner Join work.TSecondlist as b on
  Compged(a.FirstMed_trunc,b.SndMed_trunc) LT 50;
Quit;
```

Modified COMPGED Fuzzy Matching

Table 1 is a table describing the algorithm that COMPGED uses to assign values ranking the match.

Operation	Cost	Description of Operation
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.**

Table 1. Algorithm for COMPGED matching

CONCLUSION

COMPGED is a useful tool in any type of matching process. You should review alternatives to COMPGED prior to moving forward in analyses to ensure the best function is used for the matching process.

CONTACT INFORMATION <HEADING 1>

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

Arti Virkud
avirkud@gmail.com