Matching without Unique Identifiers – Fuzzy Matching Applications with SAS® Software

Wade Bannister, Arizona State University, Tempe, AZ

ABSTRACT

As the sheer volume of data increases, an increase in the need to link people or places or products from one data source to another has followed, particularly in the healthcare field. Unfortunately, as the number of available data sources has sharply increased, the use of “unique” identifiers has not. Further, what constitutes a unique identifier in one data set may have no relationship with a unique identifier in another. This paper introduces the concept of fuzzy matching, which is a technique to link people (or other entities) across data sources when there are no unique identifiers available. Fuzzy matching typically utilizes numerous fields which combine to create approximate matches which then must be evaluated. Issues and examples related to the techniques and difficulties in implementing a fuzzy matching algorithm using SAS® software are presented, including data integrity issues, processing efficiency, blocking, and distance measures.

INTRODUCTION

Developing an effective and efficient fuzzy matching process can appear to be a daunting task. The very nature of the problem makes its development, for lack of a better word, fuzzy. Additionally, the sheer size of the data involved in most fuzzy matching applications tend to further confound the issue. While a paper such as this cannot hope to provide step-by-step directions for developing your own fuzzy matching process, it does outline some of the important issues to consider before building as well as provide tips and useful concepts for the SAS® programming itself.

WHY FUZZY MATCHING?

In today’s data driven frenzy, it is often necessary to link people from one database to people in another database where there are no unique identifiers to link the people with. For example, one may wish to link the patient records from Hospital A to the patient records from Hospital B. This seems simple enough until you notice that Hospital A’s patient ID number has no relation to the ID numbers that Hospital B uses. Fortunately, there are lots of other data fields available for each person that do not depend on the hospital, such as patient name, date of birth, social security number, etc. However, these fields do not necessarily construe a unique patient identifier – social security numbers are often not reported, names can be changed, abbreviated, or spelled differently, and there are typically a host of data entry errors. Data quality issues aside, it is entirely possible for someone to have identical values in each of these fields and still not be the same person.

Fuzzy matching techniques help to automate the process of linking people based on some measure of “closeness” when unique identifiers are not available. One of the most difficult aspects of developing a fuzzy matching algorithm is finding a way to make the machine “think” in a way that mimics the human decision making process. Consider the following example:

Record	First Name	Last Name	Date of Birth	SSN
1	Mary	Hernandez	12/24/1957	123-45-6789
2	Mary	Hernandes	12/04/1957	
3	Ixbar	Zhyputin	07/19/1991	555-12-1212
4	Ixbar	Zhyputini	07/09/1991	

We see that the two pairs of potential matches have similar characteristics, as the last names differ by only one letter, the dates of birth differ by just one number, and each pair of matches is missing a social security number. However, a person reviewing these records manually would tend to match the two “Ixbar”s but not the two “Mary”s, based solely on the fact that the name Ixbar Zyputin is far less common than Mary Hernandez. There are many other similar issues to consider when automating these types of decisions.

DESIGN CONSIDERATIONS

As with any project of this scale, it is important to make some assessments and do some planning before jumping into the code writing.

There are several things to consider when contemplating how you should develop your fuzzy matching application. There of course exist software products that one can purchase “off-the-shelf”, but these tend to be inflexible and may not be well suited for your particular application. Outside of this, a custom fuzzy matching application that matches your needs can be as elaborate or non-elaborate as you like. With sufficient time and SAS programming skill available, you can develop a fully automated and nearly perfectly accurate application that requires merely pushing a button and watching the programs run. On the other hand, a rudimentary system that merely matches the “obvious” records and outputs the rest for manual review are much simpler to develop, but of course require far more time and effort to use. Typically, some sort of compromise is reached, and I have found that developing a system that is almost fully automated but that still requires some manual review of a very small percentage of the matches is the optimal approach.

TIPS FOR BUILDING A FUZZY MATCHING SYSTEM

While the actual implementation can vary greatly, there are three primary steps of any fuzzy matching system. The first step involves linking each record in data set A to each record (or each reasonable record) in data set B. Within each of these comparisons, the second step is then to compare the values of each of the relevant fields between the two. The third and final step is actually making the decision as to whether the two records should be considered a match based on the evidence available. There are many ways to accomplish these steps, and it would be impossible to provide a step-by-step guide; however, the following tips should greatly assist you in the development of a fuzzy matching system that will meet your specific needs.

DATA QUALITY

Anybody who has attempted to develop a fuzzy matching system can attest to the importance of data quality. The more complete and accurate your source data is, the easier it is to match with. While there is little you can do about missing or erroneous data, there are steps one can take to take to improve the data quality. Ensuring consistent formatting, such as capitalization, removal of extraneous characters, and standardization of things such as addresses and titles will vastly improve the accuracy of the matching process. SAS® in fact offers their Data Quality Solution package that can automate much of this type of data cleanup, or you can program these rules yourself.

There are, unfortunately, data issues that are not very simple to correct, particularly with names and addresses. “Mike” can become “Michael”, and “123 E First Street” can easily be listed as “123 East 1st St.” While the human eye can quickly identify these as matches, it is more difficult to automate such a process. One approach, recommended by Charles Partridge, is to “tokenize” such fields to improve consistency. Such an approach would, for example, transform all variants of the name “William” (Will, Bill, William, Billy, etc.) into a single tokenized name. It is also highly recommended that you reduce your data elements into the most distinct fields as possible. For example, it is far more effective to separate a name field into a first name, middle name, and last name field. Otherwise it becomes very difficult to compare the two records, as illustrated below:

Name	Address
John Q Public	111 E 10th St Phoenix AZ 85000
Johnny Quincy Public	111 E Tenth Street, Phx, AZ 85001
JQ Public	Phoenix, AZ 85000

First Name	Middle Initial	Last Name	Street	City	State	Zip
John	Q	Public	111 E 10th St	Phoenix	AZ	85000
Johnny	Q	Public	111 E Tenth Street	Phx	AZ	85001
J	Q	Public		Phoenix	AZ	85000

We see that the act of comparing the records in the first table is extremely cumbersome – to SAS, the string “John Q Public” looks much different than the string “JQ Public”. But, separating this into first, middle, and last names makes such comparisons much simpler – it is easy to see that the Last Name and Middle Initial fields match exactly, and that the first name is different by only three characters. Thus SAS would be much more likely to match these records when the fields have been separated in this manner. The address fields are similarly much easier to compare when separated into their distinct elements. This also serves as an excellent way to both identify and work around missing values.

Also to be contended with are null entries that have been instead pre-filled with things such as “JOHN DOE” and “999-99-9999”. These can be very difficult to catch, particularly in large datasets, and can lead to grave matching errors. Therefore it is recommended that you create frequency distributions on these fields to help identify over-used values. Again, this is an example of how knowing your data and building your matching process to meet your particular needs can vastly improve the accuracy of the resulting matches.

All of these data quality issues can have drastic impacts on the matching results, and thus time invested in correcting these before beginning the matching is always time well spent.

CHOOSING FIELDS TO COMPARE

In nearly all fuzzy matching applications, there are multiple fields available to help identify potential matches. One of the difficult design questions that must be addressed is choosing which fields are the most useful to use in your comparison. There is a tendency to want to use every available field under the belief that the more information, the better. If we were dealing with perfect data, this would be true, but one must be wary of fields that can contain erroneous data or data elements that can be reported differently at different times. Two important criteria to consider are the field's discriminatory power and consistency. A field's discriminatory power has to do with how well the field can discriminate between two records, and consistency refers to how consistently the data is reported across time and data sets. Suppose, for example, that your data sets have the following variables available:

First Name
Middle Name
Last Name
Social Security Number
Age
Gender
Race
Address
Zip Code

In this example, the first name, last name, and social security number fields are clearly useful fields. The middle name, while potentially very helpful in identifying a person, is useful only if it is consistently captured in both data sets – if this field is null most of the time in one dataset, than it will not prove useful. Age is generally not very useful, unless both data sets were collected during almost identical time periods. Gender, while not providing highly identifiable information, is almost always reported correctly, rarely changes, and thus can provide some discriminatory information -- if the gender does not match, it is almost certainly not the same person. Race, on the other hand, can be reported differently by the same person at different times, and is often omitted, and thus is usually less useful. Fields such as address and zip code can be very useful if the fields are accurately reported, have been well scrubbed, and standardized, and the data sets you are matching are from at least similar time periods.

BLOCKING

Once the data have been appropriately scrubbed and the data fields have been chosen, one can begin the task of comparing all of the records to each other. The concept is fairly straightforward – compare each record in dataset A to each record in dataset B, and then flag all the records that look like reasonable matches. Unfortunately, such a process requires a full Cartesian join, which become of astronomical size with even moderately sized tables (e.g., joining just 10,000 records in dataset A to 10,000 records in dataset B would result in a table with 100,000,000 records). The technique known as blocking allows one to only attempt to match records that have any reasonable chance of being a match. That is, only those records which have some basic elements in common are compared to each other. A very simple example would be to block on gender and thus only attempt to join records whose gender already match. Using the previous example, blocking on gender would potentially result in matching 5,000 males in dataset A to the 5,000 males in dataset B and then the 5,000 females in dataset A to the 5,000 females in dataset B, resulting in two sets of 250,000 record tables, for a total of 500,000. Employing additional blocks can more even more drastically reduce the number of matching attempts that are made. However, employing blocks that are restrictive will ultimately reduce the accuracy of your matching, as matches that violate your blocking assumptions will never even be attempted.

DISTANCE MEASURES

Methods for assessing how "close" two potential matches are is an area that is still under some research. Unless one can be assured of perfectly cleaned and tokenized data, one must allow for some amount of variation in two fields that could still be considered a match. For example, names can be abbreviated or spelled incorrectly; digits can be transposed on social security numbers and dates of birth, etc. One must calculate a distance measure that can take into account misspellings and transpositions effectively. A good distance measure should numerically compute how different one string is from another by accounting for these spelling errors, transpositions, and truncations. There is a variety of distance measures available, including the SPEDIS function in SAS, as well as more advanced measures such as Euclidian or Hamming distances. Also invaluable are tools which can transform proper names into their phonetic equivalent, such as the SOUNDEX function available in SAS® as well as more sophisticated phonetic algorithms that can be programmed.

ACCURACY AND DECISION MAKING

Improving and tuning the accuracy of the resultant matches is often a matter of trial and error as well as just continued process improvement. When performing fuzzy matching using several fields at once, it is usually desirable to assign greater weight to certain fields matching than others. For example, a matching zip code and date of birth between two people is not very strong evidence of a match, while a matching social security number is very strong evidence. Thus, a certain amount of tinkering is required to determine the optimal weights for each field comparison that suits your particular data and application. Also of importance is developing a way to weed out false matches resulting from either bad data or common names. There are a variety of resources available that discuss these approaches in greater detail. Above all, it is strongly recommended that you test your algorithm thoroughly using test data and subsets before putting it into production.

CONCLUSIONS

Developing your own fuzzy matching system can be a trying and difficult endeavor, but the resultant tool is often invaluable for tasks that are becoming increasingly necessary. Building an accurate, efficient, and easy to use fuzzy matching application can vastly improve the way an organization can assimilate and store information.

REFERENCES

The following are excellent references for anyone developing a fuzzy matching routine:

Foley, Malachy (1999), "Fuzzy Merges: Examples and Techniques" *Proceedings of the Twenty Fourth Annual SAS Users Group International Conference,* 46*.*

Partridge, Charles (1998), "The Fuzzy Feeling SAS Provides: Electronic Matching of Records without Common Keys," *Observations, The Technical Journal for SAS Users*, www 15.

CONTACT INFORMATION

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

Wade Bannister
School of Health Management and Policy
Arizona State University
PO Box 874506
Tempe, AZ 85287-4506
wade.bannister@asu.edu