Caverphone : Phonetic Matching algorithm

Caversham Project Occasional Technical Paper **Code Number:**CTP060902
Author: David Hood
Date: 2nd September 2002
Keywords: phonetic encoding, data matching, history, names
E-mail: caversham@otago.ac.nz

The Technical papers series is intended to document specific technical solutions to problems encountered within the project. It is being made available to the public in the event the techniques outlined may be of use to others.

Synopsis: The linking of different instances of the same person through multiple sources is partially based on name identification. Some evidence exists that particular social classes are more formal about the use of their names. This potentially leads to a class based bias in record linkage. By broadening the linkage criteria to include variation in names, this bias is ameliorated.

Note: The caverphone algorithm is available at http://caversham.otago.ac.nz/cavphone.php

Description of Problem:

A key part of the Caversham Project research involves linking between data sources[1]. A major component of such linkages is the matching of names of people between the different sources. Since the matching by name is so important, we need to have confidence in the matching criteria,

Since the Caversham database is available to the public through web-access, those interested in family history make regular use of it. Among the feedback that we receive are messages noting when the database is incorrect. Some of this error is due to scanning/checking error, some due to compilation errors in the original rolls, but some is due to variations in name with the electoral rolls over time.

In addressing difficulties caused by the variations in spelling of names, we must decide:
Is this a problem?
If it is a problem, can it be ameliorated?
If it can be ameliorated, how can it be so?

Is it a problem?

Because the project deal principally with broad statistical analysis, rather than the tracing of individual life stories, variation in name is only a problem if there is any suggestion that such variation is linked to socio-economic factors. If the variation is the result of independent random chance, then it is not going to affect the overall picture of the past.

In examining the frequency of occurrence of middle names among males in the electoral roll database, it was found that the class distribution of those with one or more middle names was different to those without a middle name (Table 1).

1. See Hood, David 'Matching multiple data sources from New Zealand: the experience of the Caversham Project' History and Computing, 12 (2), 2000

Table 1: Class distribution by presence of middle names of electoral roll entries

Class	No middle Name%	Middle name(s)%
Large Employer	0.99	1.50
Professional	1.57	3.24
Semi-Professional	1.56	2.81
Petty Proprietor	10.97	8.72
Petty Official	4.12	4.96
White collar	11.76	21.51
Skilled	32.75	31.50
Semi-Skilled	6.43	6.57
Unskilled	29.85	19.20
Total	100.00	100.00

From the over-representation of middle names among white collar (and to some extent upper class) workers, and the underrepresentation among unskilled workers, we can could that name usage is conditioned by societal factors. As such, we can assume that by making the databases ability to match entries by name more flexible we will be improving the representativeness of the information held.

Linking on the basis of phonetic encoding also aids in identifying individuals that were less than consistent in the spelling of their names. While the identification of records about individuals has not been the primary goal of the project, improving the linkages does assist in this when required.

If it is a problem, can it be ameliorated?

Having concluded that the database and the project's goals would benefit from phonetic linking (and benefit to an extent justifying the effort) various phonetic linkage strategies were investigated:

Soundex:
Reference : http://www.archives.gov/research_room/genealogy/census/soundex.html

Soundex dates back to the late 19th century. Possibly the best known phonetic encoding system, it has been used by many institutions and individuals, including the United States census.

Algorithm (excerpted from reference site until the discussion section):

A soundex code consists of a letter and three numbers. The letter is always the first letter of the surname. The numbers are assigned to the remaining letters of the surname according to the soundex guide shown below. Zeroes are added at the end if necessary to produce a four-character code. Additional letters are disregarded. Examples:
Washington is coded W-252 (W, 2 for the S, 5 for the N, 2 for the G, remaining letters disregarded).
Lee is coded L-000 (L, 000 added).

Number	Represents the Letters
1	B, F, P, V
2	C, G, J, K, Q, S, X, Z
3	D, T
4	L
5	M, N
6	R

Disregard the letters A, E, I, O, U, H, W, and Y.

Additional Soundex Coding Rules
Names With Double Letters

If the surname has any double letters, they should be treated as one letter. For example:
Gutierrez is coded G-362 (G, 3 for the T, 6 for the first R, second R ignored, 2 for the Z).

Names with Letters Side-by-Side that have the Same Soundex Code Number

If the surname has different letters side-by-side that have the same number in the soundex coding guide, they should be treated as one letter. Examples:
Pfister is coded as P-236 (P, F ignored, 2 for the S, 3 for the T, 6 for the R).

Jackson is coded as J-250 (J, 2 for the C, K ignored, S ignored, 5 for the N, 0 added).

Tymczak is coded as T-522 (T, 5 for the M, 2 for the C, Z ignored, 2 for the K). Since the vowel "A" separates the Z and K, the K is coded.

Names with Prefixes

If a surname has a prefix, such as Van, Con, De, Di, La, or Le, code both with and without the prefix because the surname might be listed under either code. Note, however, that Mc and Mac are not considered prefixes.
For example, VanDeusen might be coded two ways:

V-532 (V, 5 for N, 3 for D, 2 for S)
or

D-250 (D, 2 for the S, 5 for the N, 0 added).

Consonant Separators

If a vowel (A, E, I, O, U) separates two consonants that have the same soundex code, the consonant to the right of the vowel is coded. Example:
Tymczak is coded as T-522 (T, 5 for the M, 2 for the C, Z ignored (see "Side-by-Side" rule above), 2 for the K). Since the vowel "A" separates the Z and K, the K is coded.
If "H" or "W" separate two consonants that have the same soundex code, the consonant to the right of the vowel is not coded. Example:
Ashcraft is coded A-261 (A, 2 for the S, C ignored, 6 for the R, 1 for the F). It is not coded A-226.

Soundex Discussion:

Soundex is used by creating a phonetic version of the name in question. This is stored as part of the record in the database, then used as a basis for comparison.

One common comment about using phonetic matching is that it increases the poll of potential matches for a particular record. As such it work well when locating potential matches manually for small numbers of searches, but it complicates automatic linking. This difficulty is true for any phonetic technique however, and can be remedied in way in which the linking is carried out.

Turning to more specific comments on Soundex, the following points should be made:
- Soundex is popular, many arcives exist offering soundex in a searchable format.
- Soundex is not sensitive to the changes in pronunciation that occur when letters are in slightly different positions. Bing has a soundex code of B-520, as does Benig. Extra vowels can significantly effect the pronunciation of words.
- By not encoding the initial letter into a phonetic format, potential matches are missed. An example would be Bing versus Ping.
- Soundex's preservation of the post vocalic L (hall) and R (door) are only useful in those languages and dialects

in which such features are distinctive.

Soundex was rejected on the following grounds:
- Soundex misses potential matches by not coding the intial consonant and by coding the L and R.

Metaphone

Metaphone is a more recent phoentic encoding. This discussion focuses on the version modified by Michael Kuhn from Lawrence Philips' original

reference:http://aspell.sourceforge.net/metaphone/metaphone-kuhn.txt

Algorithm (excerpted from reference site until the discussion section):

The 16 consonant sounds:

```
                              |--- ZERO represents "th"
                              |
   B  X  S  K  J  T  F  H  L  M  N  P  R  0  W  Y
```

Exceptions:

Beginning of word: "ae-", "gn", "kn-", "pn-", "wr-" ----> drop first letter
"Aebersold", "Gnagy", "Knuth", "Pniewski", "Wright"

Beginning of word: "x" ----> change to "s"
as in "Deng Xiaopeng"

Beginning of word: "wh-" ----> change to "w"
as in "Whalen"

Transformations:

B ----> B unless at the end of word after "m", as in "dumb", "McComb"

C ----> X (sh) if "-cia-" or "-ch-"
S if "-ci-", "-ce-", or "-cy-"
SILENT if "-sci-", "-sce-", or "-scy-"
K otherwise, including in "-sch-"

D ----> J if in "-dge-", "-dgy-", or "-dgi-"
T otherwise

F ----> F

G ----> SILENT if in "-gh-" and not at end or before a vowel
in "-gn" or "-gned"
in "-dge-" etc., as in above rule
J if before "i", or "e", or "y" if not double "gg"
K otherwise

H ----> SILENT if after vowel and no vowel follows
or after "-ch-", "-sh-", "-ph-", "-th-", "-gh-"
H otherwise

J ----> J

K ----> SILENT if after "c"
K otherwise

L ----> L

M ----> M

N ----> N

P ----> F if before "h"
P otherwise

Q ----> K

R ----> R

S ----> X (sh) if before "h" or in "-sio-" or "-sia-"
S otherwise

T ----> X (sh) if "-tia-" or "-tio-"
0 (th) if before "h"
silent if in "-tch-"
T otherwise

V ----> F

W ----> SILENT if not followed by a vowel
W if followed by a vowel

X ----> KS

Y ----> SILENT if not followed by a vowel
Y if followed by a vowel

Z ----> S

Metaphone Discussion:

Metaphone was developed to address many of the problems in Soundex. It is sensitive to changes in position of letters and combinations like TH.

While an improvement on soundex, I still found it not to be sensitive enough with the interaction of vowels (particularly the post vocalic L and R). The total exclusion of the vowels also result in some matches that would be unlikely to be made (particularly the initial vowel).

Double Metaphone:
An improved version of metaphone (http://www.cuj.com/documents/s=8038/cuj0006philips/) this was rejected as I could not find a version I could implement.

New York State Identification and Intelligence Algorithm (NYSIIS)

In 1970 the New York State Identification and Intelligence project headed by Robert L. Taft published the paper "Name Search Techniques". In this paper he compared Soundex with a new phonetic routine (NYSIIS) that was

designed through rigorous empirical analysis.

Algorithm (Taft, 1970):

1) Translate first characters of name:
MAC => MCC, PH => FF, KN => NN, PF => FF, K => C, SCH => SSS

(2) Translate last characters of name
EE => Y, IE => Y, DT,RT,RD,NT,ND => D

(3) First character of key = first character of name.

(4) Translate remaining characters by following rules, incrementing by one character each time:
a. EV => AF else A,E,I,O,U => A
b. Q => G Z => S M => N
c. KN => N else K => C
d. SCH => SSS PH => FF
e. H => If previous or next is nonvowel, previous
f. W => If previous is vowel, previous

(5) Check last character
1. If last character is S, remove it
2. If last characters are AY, replace with Y
3. If last character is A, remove it

NYSIIS discussion:

Falling between Soundex and Metaphone in terms of sophistication, NYSIIS suffers the same problems as the other algorithms in addressing letter position.

Other phonetic linking techniques.

Other techniques were also investigated. One particularly promising one involved assessing the likelihood that two names were the same by counting the number of transformation required to turn one name into the other. The major difficultly of this "Edit Distance" (Ristad and Yianilos, Learning String Edit Distance, IEEE Trans. PAMI, 20(5):522-532, 1998) technique is that, because it comparison individual names (rather than providing a coded form) it becomes very inefficient when used with large numbers of entries. As the project databases can involve large numbers of entries, this approach was rejected.

Caverphone

As the existing models for phonetic matching were found to be unsuitable for the caversham data set, a customised model of phonetic encoding was developed. This was designed to be similar to metaphone in the sensitivity to letter placement, but with an encoding scheme intended to allow for the range of accents present in the study area (southern part of the city of Dunedin, New Zealand) in the years 1893-1938. The increased presence of computers in society means that modern phonetic encoding systems (metaphone, caverphone) can take advantage of many more rules and exceptions than in the more traditional manually compiled forms.

The rules are applied consecutively, in a cascading fashion. An individual letter can thus have several transformations applied to it. In the case of rules that are of the form replace x with y, then move from left to right through the name, replacing occurances of the letter in the name as you go.

Caverphone Algorithm:

- Start with a Surname or Firstname
- Convert to lowercase
  This coding system is case sensitive, implementations should acknowledge that a is not the same as A
- Remove anything not A-Z
  The main intention of this is to remove spaces, hyphens, and apostrophes.
  example: o'brian becomes obrian
- If the name starts with cough make it cou2f
  2 is being used as a temporary placeholder to indicate a consonant which we are no longer interested in.
- If the name starts with rough make it rou2f
- If the name starts with tough make it tou2f
- If the name starts with enough make it enou2f
- If the name starts with gn make it 2n
- If the name ends with mb make it m2
- replace cq with 2q
- replace ci with si
- replace ce with se
- replace cy with sy
- replace tch with 2ch
- replace c with k
- replace q with k
- replace x with k
- replace v with f
- replace dg with 2g
- replace tio with sio
- replace tia with sia
- replace d with t
- replace ph with fh
- replace b with p
- replace sh with s2
- replace z with s
- replace and initial vowel with an A
- replace all other vowels with a 3
  3 is a temporary placeholder marking a vowel
- replace 3gh3 with 3kh3
  Exceptions are dealt with before the general case. gh between vowels is an except of the more general gh rule.
- replace gh with 22
- replace g with k
- replace groups of the letter s with a S
  Continuous strings of s are replace by a single S
- replace groups of the letter t with a T
- replace groups of the letter p with a P
- replace groups of the letter k with a K
- replace groups of the letter f with a F
- replace groups of the letter m with a M
- replace groups of the letter n with a N
- replace w3 with W3
- replace wy with Wy
- replace wh3 with Wh3
- replace why with Why
- replace w with 2
- replace and initial h with an A
- replace all other occurrences of h with a 2
- replace r3 with R3

- replace ry with Ry
- replace r with 2
- replace l3 with L3
- replace ly with Ly
- replace l with 2
- replace j with y
- replace y3 with Y3
- replace y with 2
- remove all 2s
- remove all 3s
- put six 1s on the end
- take the first six characters as the code

As this is intended for computer implementation it is easier to tell the computer to add six 1s to the end, then take the first six characters. This is easier than implementing a rule of add 1 at the end to make the number of places up to six.

Example 1:
David
david
dafid
tafit
t3f3t
T3f3T
T3F3T
TFT
TFT111111
TFT111

Example 2:
Whittle
whittle
wh3ttl3
wh3Tl3
Wh3Tl3
W23Tl3
W23TL3
W3TL3
WTL
WTL1111111
WTL111

Example implementation (javascript):

```
function caverphonise(str) {
var rep=/[^a-zA-Z]/g ;
aragorn = str.replace(rep,"");
aragorn = aragorn.toLowerCase();
if(aragorn.substring(0, 5)=="cough"){
aragorn = "cou2f" + aragorn.substring(5, aragorn.length);
}
if(aragorn.substring(0, 5)=="rough"){
aragorn = "rou2f" + aragorn.substring(5, aragorn.length);
}
if(aragorn.substring(0, 5)=="tough"){
aragorn = "tou2f" + aragorn.substring(5, aragorn.length);
```

```
}
if(aragorn.substring(0, 5)=="enough"){
aragorn = "enou2f" + aragorn.substring(5, aragorn.length);
}
if(aragorn.substring(0, 2)=="gn"){
aragorn = "2n" + aragorn.substring(2, aragorn.length);
}
if(aragorn.substring((aragorn.length-3), aragorn.length)=="mb"){
aragorn = aragorn.substring(0, aragorn.length-2)+"m2";
}
aragorn = aragorn.replace(/cq/g,"2q");
aragorn = aragorn.replace(/ci/g,"si");
aragorn = aragorn.replace(/ce/g,"se");
aragorn = aragorn.replace(/cy/g,"sy");
aragorn = aragorn.replace(/tch/g,"2ch");
aragorn = aragorn.replace(/c/g,"k");
aragorn = aragorn.replace(/q/g,"k");
aragorn = aragorn.replace(/x/g,"k");
aragorn = aragorn.replace(/v/g,"f");
aragorn = aragorn.replace(/dg/g,"2g");
aragorn = aragorn.replace(/tio/g,"sio");
aragorn = aragorn.replace(/tia/g,"sia");
aragorn = aragorn.replace(/d/g,"t");
aragorn = aragorn.replace(/ph/g,"fh");
aragorn = aragorn.replace(/b/g,"p");
aragorn = aragorn.replace(/sh/g,"s2");
aragorn = aragorn.replace(/z/g,"s");
if(aragorn.substring(0, 1)=="a"){
aragorn = "A" + aragorn.substring(1, aragorn.length);
}
aragorn = aragorn.replace(/a/g,"3");
if(aragorn.substring(0, 1)=="e"){
aragorn = "A" + aragorn.substring(1, aragorn.length);
}
aragorn = aragorn.replace(/e/g,"3");
if(aragorn.substring(0, 1)=="i"){
aragorn = "A" + aragorn.substring(1, aragorn.length);
}
aragorn = aragorn.replace(/i/g,"3");
if(aragorn.substring(0, 1)=="o"){
aragorn = "A" + aragorn.substring(1, aragorn.length);
}
aragorn = aragorn.replace(/o/g,"3");
if(aragorn.substring(0, 1)=="u"){
aragorn = "A" + aragorn.substring(1, aragorn.length);
}
aragorn = aragorn.replace(/u/g,"3");
if(aragorn.substring((aragorn.length-1), aragorn.length)=="j"){
aragorn = aragorn.substring(0, aragorn.length-1)+"g";
}
aragorn = aragorn.replace(/3gh3/g,"3kh3");
aragorn = aragorn.replace(/gh/g,"22");
aragorn = aragorn.replace(/g/g,"k");
aragorn = aragorn.replace(/ss*/g,"S");
aragorn = aragorn.replace(/tt*/g,"T");
```

```
aragorn = aragorn.replace(/pp*/g,"P");
aragorn = aragorn.replace(/kk*/g,"K");
aragorn = aragorn.replace(/ff*/g,"F");
aragorn = aragorn.replace(/mm*/g,"M");
aragorn = aragorn.replace(/nn*/g,"N");
aragorn = aragorn.replace(/w3/g,"W3");
aragorn = aragorn.replace(/wy/g,"Wy");
aragorn = aragorn.replace(/wh3/g,"Wh3");
aragorn = aragorn.replace(/why/g,"Why");
aragorn = aragorn.replace(/w/g,"2");
if(aragorn.substring(0, 1)=="h"){
aragorn = "A" + aragorn.substring(1, aragorn.length);
}
aragorn = aragorn.replace(/h/g,"2");
aragorn = aragorn.replace(/r3/g,"R3");
aragorn = aragorn.replace(/ry/g,"Ry");
aragorn = aragorn.replace(/r/g,"2");
aragorn = aragorn.replace(/l3/g,"L3");
aragorn = aragorn.replace(/ly/g,"Ly");
aragorn = aragorn.replace(/l/g,"2");
aragorn = aragorn.replace(/j/g,"y");
aragorn = aragorn.replace(/y3/g,"Y3");
aragorn = aragorn.replace(/y/g,"2");
aragorn = aragorn.replace(/2/g,"");
aragorn = aragorn.replace(/3/g,"");
aragorn = aragorn +"111111"
aragorn = aragorn.substring(0, 6)
alert(aragorn);
}
```

Note: Aragorn is simply a variable reference for the string in progress. As this is a series of find/replace actions on a string, it is easily suited to implementation using scripting languages with good support for string manipulation (javascript, perl, python).

If it can be ameliorated, how can it be so?

The easiest way to apply the phonetic encoding is to take a column of names and create an additional column of phonetically encoded forms beside the originals. This can be extracting the unique names from a database as tab delimited text, which are then used to create a table that can be linked back to the original source table. It could also be taking a spreadsheet of records and save the name column as tab delimited text and creating a phonetic entry for each record.

The advantage of having the list of names to be encoded as tab delimited text is that it is a trivial operation in any scripting language to read each line of the file, and write a version of that line with the added encoding to a new file.

In most tests of phonetic linking the process is compared to either other phonetic linking techniques or no phonetic linking. However a sensible linking strategy uses both. When linking between records in two sources (from A to B) is attempted, their are four possible outcomes:

- One record in source A matches to a single record in source B (success)
- One record in A matches to many records in B (failure)
- Many records in A match to a single record in B (failure)
- One record in A matches to no records in B (failure)

Phonetic matching, when compared to matching by actual name, would decrease the number of successes by creating more potential matches, so it does not help in the first case. Phonetic matching should thus be

employed to decrease the failures in the second, third, and fourth cases. If multiple linkage attempts using different sets of criteria are being made, a failure due to too many matches in B may make a tight match with a single record in B on the basis of phonetic links. In the third case it may provide a unique match on the basis of phonetic linking and other additional information. And example would be two Robert Smiths in source A and one in source B, if one of the Smiths is phonetically matched to to a Robert Symthe at the same address, then that becomes a success (and by a process of elimination the other Robert Smith can be matched as well). In the forth case phonetic matching can produce links between entries in A and B that have no other suggested links.

The suggested steps for using the phonetic encoding to best effect are:

1) Use multiple criteria (starting with the most stringent) to link A and B on the basis of actual names.

2) Where a unique match exists, record the ID number of A in B and B in A, and remove both records from the pools of records being linked.

3) Once all combinations of linking criteria have been tested using actual names, repeat them using phonetic matches.

4) When using phonetic matches test phonetic surname and actual firstname, then repeat all the criteria using actual surname and phonetic firstname, then phonetic representations of both. Surname is tested first as a qualitative assessment suggests that most phonetic variation occurred in the surname, firstnames were more prone to abbreviation or substitution of middle names (a separate problem).

5) Repeat the entire process using actual names on the unmatched pools of records, as some failure due to ties (to many entries) may have been resolved.

When conducting any particular test of A against B, A should first be tested against A. This will identify where there are going to be many entries in A against a record in B (the second case of result).

Given contemporary computing power, a typical modern computer could be set-up to run all these tests over a 48 hour period and return with a set of results. For an example of such a test see Hood, David 'Matching multiple data sources from New Zealand: the experience of the Caversham Project' History and Computing, 12 (2), 2000.