Cross linguistic name matching in English and Arabic: A one to many mapping extension of the Levenshtein edit distance algorithm

Article · July 2006

DOI: 10.3115/1220835.1220895

CITATIONS
43

READS
641

4 authors, including:

Sherri Condon
MITRE
40 PUBLICATIONS **843** CITATIONS

SEE PROFILE

Christopher Ackerman
Google Inc.
6 PUBLICATIONS **92** CITATIONS

SEE PROFILE

Cross Linguistic Name Matching in English and Arabic: A "One to Many Mapping" Extension of the Levenshtein Edit Distance Algorithm

Dr. Andrew T. Freeman, Dr. Sherri L. Condon and Christopher M. Ackerman
The Mitre Corporation
7525 Colshire Dr
McLean, Va 22102-7505
{afreeman, scondon, cackerman}@mitre.org

Abstract

This paper presents a solution to the problem of matching personal names in English to the same names represented in Arabic script. Standard string comparison measures perform poorly on this task due to varying transliteration conventions in both languages and the fact that Arabic script does not usually represent short vowels. Significant improvement is achieved by augmenting the classic Levenshtein edit-distance algorithm with character equivalency classes.

1 Introduction to the problem

Personal names are problematic for all language technology that processes linguistic content, especially in applications such as information retrieval, document clustering, entity extraction, and translation. Name matching is not a trivial problem even within a language because names have more than one part, including titles, nicknames, and qualifiers such as *Jr*. or *II*. Across documents, instances of the name might not include the same name parts, and within documents, the second or third mention of a name will often have only one salient part. In multilingual applications, the problem is complicated by the fact that when a name is represented in a script different from its native script, there may be several alternative representations for each phoneme, leading to large number of potential variants for multi-part names.

A good example of the problem is the name of the current leader of Libya. In Arabic, there is only one way to write the consonants and long vowels of any person's name, and the current leader of Libya's name in un-vocalized Arabic text can only be written as معمر القذافي. In English, his name has many common representations. Table 1 documents the top five hits returned from a web search at www.google.com, using various English spellings of the name.

Version	Occurrences
Muammar Gaddafi	43,500
Muammar Qaddafi	35,900
Moammar Gadhafi	34,100
Muammar Qadhafi	15,000
Muammar al Qadhafi	11,500

Table 1. Qadhafy's names in English

Part of this variation is due to the lack of an English phoneme corresponding to the Standard Arabic phoneme /q/. The problem is further compounded by the fact that in many dialects spoken in the Arabic-speaking world, including Libya, this phoneme is pronounced as [g].

The engineering problem is how one reliably matches all versions of a particular name in language A to all possible versions of the same name in language B. Most solutions employ standard string similarity measures, which require the names to be represented in a common character set. The solution presented here exploits transliteration conventions in normalization procedures and equivalence mappings for the standard Levenshtein distance measure.

2 Fuzzy string matching

The term fuzzy matching is used to describe methods that match strings based on similarity rather than identity. Common fuzzy matching techniques include edit distance, n-gram matching, and normalization procedures such as Soundex.

Proceedings of the Human Language Technology Conference of the North American Chapter of the ACL, pages 471–478,
New York, June 2006.

This section surveys methods and tools currently used for fuzzy matching.

2.1 Soundex

Patented in 1918 by Odell and Russell the Soundex algorithm was designed to find spelling variations of names. Soundex represents classes of sounds that can be lumped together. The precise classes and algorithm are shown below in figures 1 and 2.

Code:	0	1	2	3	4	5	6
Letters:	aeiouy hw	bp fv	cgjkq sxz	dt	l	mn	r

Figure 1: Soundex phonetic codes

1. Replace all but the first letter of the string by its phonetic code.
2. Eliminate any adjacent repetitions of codes.
3. Eliminate all occurrences of code 0, i.e. eliminate all vowels.
4. Return the first four characters of the resulting string.
5. Examples: Patrick = P362, Peter = P36, Peterson = P3625

Figure 2: The Soundex algorithm

The examples in figure 2 demonstrate that many different names can appear to match each other when using the Soundex algorithm.

2.2 Levenshtein Edit Distance

The Levenshtein algorithm is a string edit-distance algorithm. A very comprehensive and accessible explanation of the Levenshtein algorithm is available on the web at http://www.merriampark.com/ld.htm.

The Levenshtein algorithm measures the edit distance where edit distance is defined as the number of insertions, deletions or substitutions required to make the two strings match. A score of zero represents a perfect match.

With two strings, string *s* of size *m* and string *t* of size *n*, the algorithm has O(*nm*) time and space complexity. A matrix is constructed with *n* rows and *m* columns. The function $e(s_i,t_j)$ where s_i is a character in the string *s,* and t_j is a character in string *t* returns a 0 if the two characters are equal and a 1 otherwise. The algorithm can be represented compactly with the recurrence relation shown in figure 3.

```
for each i from 0 to |s|
    for each j from 0 to |t|
        levenshtein(0; 0) = 0
        levenshtein(i; 0) = i
        levenshtein(0;j) = j
        levenshtein (i;j) =
                min[levenshtein (i − 1; j) + 1;
                levenshtein(i; j − 1) + 1;
                levenshtein(i − 1; j − 1) +
                e(s_i; t_j )]
```

Figure 3. Recurrence relation for Levenshtein edit distance

A simple "fuzzy-match" algorithm can be created by dividing the Levenshtein edit distance score by the length of the shortest (or longest) string, subtracting this number from one, and setting a threshold score that must be achieved in order for the strings to be considered a match. In this simple approach, longer pairs of strings are more likely to be matched than shorter pairs of strings with the same number of different characters.

2.3 Editex

The Editex algorithm is described by Zobel and Dart (1996). It combines a Soundex style algorithm with Levenshtein by replacing the $e(s_i,t_j)$ function of Levenshtein with a function $r(s_i,t_j)$. The function $r(s_i,t_j)$ returns 0 if the two letters are identical, 1 if they belong to the same letter group and 2 otherwise. The full algorithm with the letter groups is shown in figures 4 and 5. The Editex algorithm neutralizes the *h* and *w*. This shows up in the algorithm description as $d(s_{i-1},s_i)$. It is the same as $r(s_i,t_j)$, with two exceptions. It compares letters of the same string rather than letters from the different strings. The other difference is that if s_{i-1} is *h* or *w*, and $s_{i-1} \neq s_i$, then $d(s_{i-1},s_i)$ is one.

```
for each i from 0 to |s|
    for each j from 0 to |t|
    editex(0; 0) = 0
    editex(i; 0) = editex(i − 1; 0) + d(s_{i−1}; s_i)
    editex(0; j) = editex(0; j − 1) + d(t_{j−1}; t_j )
    editex(i; j) = min[editex (i − 1; j) +
                d(s_{i−1}; s_i);
                ediext(i; j − 1) + d(t_{j−1}; tj);
                editex(i − 1; j − 1) + r(s_i; t_j )]
```

Figure 4: Recurrence relation for Editex edit distance

0 1 2 3 4 5 6 7 8 9

aeiouy bp ckq dt lr mn gj fpv sxz csz

Figure 5: Editex letter groups

Zobel and Dart (1996) discuss several enhancements to the Soundex and Levenshtein string matching algorithms. One enhancement is what they call "tapering." Tapering involves weighting mismatches at the beginning of the word with a higher score than mismatches towards the end of the word. The other enhancement is what they call *phonometric methods*, in which the input strings are mapped to pronunciation based phonemic representations. The edit distance algorithm is then applied to the phonemic representations of the strings.

Zobel and Dart report that the Editex algorithm performed significantly better than alternatives they tested, including Soundex, Levenshtein edit distance, algorithms based on counting common n-gram sequences, and about ten permutations of tapering and phoneme based enhancements to assorted combinations of Soundex, n-gram counting and Levenshtein.

2.4 SecondString

SecondString, described by Cohen, Ravikumar and Fienberg (2003) is an open-source library of string-matching algorithms implemented in Java. It is freely available at the web site http://secondstring.sourceforge.net.

The SecondString library offers a wide assortment of string matching algorithms, both those based on the "edit distance" algorithm, and those based on other string matching algorithms. SecondString also provides tools for combining matching algorithms to produce hybrid-matching algorithms, tools for training on string matching metrics and tools for matching on tokens within strings for multi-token strings.

3 Baseline task

An initial set of identical names in English and Arabic script were obtained from 106 Arabic texts and 105 English texts in a corpus of newswire articles. We extracted 408 names from the English language articles and 255 names from the Arabic language articles. Manual cross-script matching identified 29 names common to both lists.

For a baseline measure, we matched the entire list of names from the Arabic language texts against the entire list of English language names using algorithms from the *SecondString* toolkit. The Arabic names were transliterated using the computer program *Artrans* produced by Basis (2004).

For each of these string matching metrics, the matching threshold was empirically set to a value that would return some matches, but minimized false matches. The Levenshtein "edit-distance" algorithm returns a simple integer indicating the number of edits required to make the two strings match. We normalized this number by using the formula $1-\left(\frac{Levenshtein(s,t)}{|s|+|t|}\right)$, where any pair of strings with a fuzzy match score less than 0.875 was not considered to be a match. The intent of dividing by the length of both names is to minimize the weight of a mismatched character in longer strings.

For the purposes of defining recall and precision, we ignored all issues dealing with the fact that many English names correctly matched more than one Arabic name, and that many Arabic names correctly matched more than one English name. The number of correct matches is the number of correct matches for each Arabic name, summed across all Arabic names having one or more matches. Recall *R* is defined as the number of correctly matched English names divided by the number of available correct English matches in the test set. Precision *P* is defined as the total number of correct names returned by the algorithm divided by the total number of names returned. The F-score is $2\cdot\frac{(PR)}{P+R}$.

Figure 5 shows the results obtained from the four algorithms that were tested. *Smith-Waterman* is based on Levenshtein edit-distance algorithm, with some parameterization of the gap score. *SLIM* is an iterative statistical learning algorithm based on a variety of estimation-maximization in which a Levenshtein edit-distance matrix is iteratively processed to find the statistical probabilities of the overlap between two strings. *Jaro* is a type n-gram algorithm which measures the number and the order of the common characters between two strings. *Needleman-Wunsch* from Cohen et al.'s (2003) *SecondString* Java code library is the Java implementation referred to as "Levenshtein edit

distance" in this report. The Levenshtein algorithms clearly out performed the other metrics.

Algorithm	Recall	Precision	F-score
Smith Waterman	14/29	14/18	0.5957
SLIM	3/29	3/8	0.1622
Jaro	8/29	8/11	0.4
NeedlemanWunsch	19/29	19/23	0.7308

Figure 5: Comparison of string similarity metrics

4 Motivation of enhancements

One insight is that each letter in an Arabic name has more than one possible letter in its English representation. For instance, the first letter of former Egyptian president Gamal Abd Al-Nasser's first name is written with the Arabic letter جـ, which in most other dialects of Arabic is pronounced either as [δZ] or [Z], most closely resembling the English pronunciation of the letter "j". As previously noted, قـ has the received pronunciation of [q], but in many dialects it is pronounced as [g], just like the Egyptian pronunciation of Nasser's first name Gamal. The conclusion is that there is no principled way to predict a single representation in English for an Arabic letter.

Similarly, Arabic representations of non-native names are not entirely predictable. Accented syllables will be given a long vowel, but in longer names, different writers will place the long vowels showing the accented syllables in different places. We observed six different ways to represent the name Milosevic in Arabic.

The full set of insights and "real-world" knowledge of the craft for representing foreign names in Arabic and English is summarized in figure 6. These rules are based on first author Dr. Andrew Freeman's[1] experience with reading and translating Arabic language texts for more than 16 years.

1)	The hamza (ء) and the 'ayn (ع) will often appear in English language texts as an apostrophe or as the vowel that follows.
2)	Names not native to Arabic will have a long vowel or diphthong for accented syllables represented by "w," "y" or "A.
3)	The high front un-rounded diphthong ("i," "ay", "igh") found in non-Arabic names will often be represented with an alif-yaa (ايـ) sequence in the Arabic script.
4)	The back rounded diphthongs, (ow, au, oo) will be represented with a single "waw" in Arabic.
5)	The Roman scripts letters "p" and "v" are represented by "b" and "f" in Arabic. The English letter "x" will appear as the sequence "ks" in Arabic
6)	Silent letters, such as final "e" and internal "gh" in English names will not appear in the Arabic script.
7)	Doubled English letters will not be represented in the Arabic script.
8)	Many Arabic names will not have any short vowels represented.
9)	The "ch" in the English name "Richard" will be represented with the two character sequence "t" (ت) and "sh" (ش). The name "Buchanan" will be represented in Arabic with the letter "k" (ك).

Figure 6: Rules for Arabic and English representations

[1] Dr. Freeman's PhD dissertation was on Arabic dialectology.

5 Implementation of the enhancements

5.1 Character Equivalence Classes (CEQ):

The implementation of the enhancements has six parts. We replaced the comparison for the character match in the Levenshtein algorithm with a function Ar(s_i, t_j) that returns zero if the character t_j from the English string is in the match set for the Arabic character s_i;, otherwise it returns a one.

```
for each i from 0 to |s|
    for each j from 0 to |t|
        levenshtein(0; 0) = 0
        levenshtein(i; 0) = i
        levenshtein(0;j) = j
        levenshtein (i;j) =
        min
          [levenshtein (i − 1; j) + 1;
          levenshtein(i; j − 1) + 1;
          levenshtein(i − 1; j − 1) + Ar(s_i; t_j )]
```

Figure 7: Cross linguistic Levenshtein

String similarity measures require the strings to have the same character set, and we chose to use transliterated Arabic so that investigators who could not read Arabic script could still view and understand the results. The full set of transliterated Arabic equivalence classes is shown in Figure 8. The set was intentionally designed to handle Arabic text transliterated into either the Buckwalter

transliteration (Buckwalter, 2002) or the default setting of the transliteration software developed by Basis Technology (Basis, 2004).

5.2 Normalizing the Arabic string

The settings used with the Basis Artrans transliteration tool transforms certain Arabic letters into English digraphs with the appropriate two characters from the following set: (kh, sh, th, dh). The Buckwalter transliteration method requires a one-to-one and recoverable mapping from the Arabic script to the transliterated script. We transformed these characters into the Basis representation with regular expressions. These regular expressions are shown in figure 9 as perl script.

Translit-eration	English equivalence class	Arabic letter
'	',a ,A,e,E,i,I,o,O,u,U	ء
|	',a ,A,e ,E,i ,I,o ,O,u ,U	آ
>	',a ,A,e ,E,i ,I,o ,O,u ,U	أ
&	',a ,A,e ,E,i ,I,o ,O,u ,U	ؤ
<	',a ,A,e ,E,i ,I,o ,O,u ,U	إ
}	',a ,A,e ,E,i ,I,o ,O,u ,U	ئ
A	',a ,A,e ,E,i ,I,o ,O,u ,U	ا
b	b ,B,p ,P,v,V	ب
p	a ,e	ة
+	a ,e	ة
t	t,T	ت
v	t ,T	ث
j	j,J,g,G	ج
H	h, H	ح
x	k, K	خ
d	d, D	د
*	d, D	ذ
r	r, R	ر
z	z, Z	ز
s	s, S,c, C	س
$	s, S	ش
S	s, S	ص
D	d, D	ض
T	t, T	ط
Z	z, Z,d, D	ظ
E	',`,c,a,A,e,E,i,I,o,O,u,U	ع
`	',`,c,a,A,e,E,i,I,o,O,u,U	ع
g	g, G	غ
f	f, F,v, V	ف
q	q, Q, g, G,k, K	ق
k	k, K,c, C,S, s	ك
l	l, L	ل
m	m, M	م
n	n, N	ن
h	h, H	هـ
w	w, W,u, u,o, O, 0	و
y	y, Y, i, I, e, E, ,j, J	ي
Y	a, A,e, E,i, I, o,O,u, U	ى
a	a, e	َ
i	i, e	ِ
u	u, o	ُ

Figure 8: Arabic to English character equivalence sets

```
$s1 =~ s/\$/sh/g;     #  normalize Buckwalter
$s1 =~ s/v/th/g;      # normalize Buckwalter
$s1 =~ s/\*/dh/g;     # normalize Buckwalter
$s1 =~ s/x/kh/g;      # normalize Buckwalter
$s1 =~ s/(F|K|N|o|~)//g; #  remove case vowels,
                      # the shadda and the sukuun
$s1 =~ s/\'aa/\|/g;   # normalize basis w/
                      # Buckwalter madda
$s1 =~ s/(U|W|I|A)/A/g; # normalize  hamza
$s1 =~ s/_//;         # eliminate underscores
$s1 =~ s/\s//g;       # eliminate white space
```

Figure 9. Normalizing the Arabic

5.3 Normalizing the English string

Normalization enhancements were aimed at making the English string more closely match the transliterated form of the Arabic string. These correspond to points 2 through 7 of the list in Figure 6. The perl code that implemented these transformations is shown in figure 10.

```
$s2 =~ s/(a|e|i|A|E|I)(e|i|y)/y/g;
        # hi dipthongs go to y in Arabic
$s2 =~ s/(e|a|o)(u|w|o)/w/g;
        # lo dipthongs go to w in Arabic
$s2 =~ s/(P|p)h/f/g;          # ph -> f in Arabic
$s2 =~ s/(S|s)ch/sh/g;        # sch is sh
$s2 =~ s/(C|c)h/tsh/g;        # ch is tsh or k ,
        # we catch the "k" on the pass
$s2 =~ s/-//g;       # eliminate all hyphens
$s2 =~ s/x/ks/g;   # x->ks in Arabic
$s2 =~ s/e( |$)/$1/g; # the silent final e
$s2 =~ s/(\S)\1/$1/g; # eliminate duplicates
$s2 =~ s/(\S)gh/$1/g; # eliminate silent gh
$s2 =~ s/\s//g;       # eliminate white space
$s2 =~ s/(\.|,|;)//g; # eliminate punctuation
```

Figure 10. Normalizing the English

5.4 Normalizing the vowel representations

Normalization of the vowel representations is based on two observations that correspond to points 2 and 8 of Figure 6. Figure 11 shows some English names represented in Arabic transliterated using the Buckwalter transliteration method.

Name in English	Name in Arabic	Arabic transliteration
Bill Clinton	بيل كلينتون	byl klyntwn
Colin Powell	كولين باول	kwlyn bAwl

Richard Cheney	ريتشارد تشيني	rytshArd tshyny

Figure 11. English names as represented in Arabic

All full, accented vowels are represented in the Arabic as a long vowel or diphthong. This vowel or diphthong will appear in the transliterated unvocalized text as either a "w," "y" or "A." Unaccented short vowels such as the "e" found in the second syllable of "Powell" are not represented in Arabic. Contrast figure 11 with the data in figure 12.

Name in Arabic	Arabic transliteration	Name in English
مصطفى الشيخ ديب	mSTfY Alshykh dyb	Mustafa al Sheikh Deeb
محمد عاطف	mHmd EATf	Muhammad Atef
حسني مبارك	Hsny mbArk	Hosni Mubarak

Figure 12. Arabic names as represented in English

The Arabic only has the lengtheners "y", "w", or "A" where there are lexically determined long vowels or diphthongs in Arabic. The English representation of these names must contain a vowel for every syllable. The edit-distance score for matching "Muhammad" with "mHmd" will fail since only 4 out of 7 characters match. Lowering the match threshold will raise the recall score while lowering the precision score. Stripping all vowels from both strings will raise the precision on the matches for Arabic names in English, but will lower the precision for English names in Arabic.

For each i from 0 to min(|*Estring*|, |*Astring*|),
each j from 0 to min(|*Estring*|, |*Astring*|)
if $Astring_i$ equals $Estring_j$
$Outstring_i = Estring_i$ increment i and j
if vowel($Astring_i$) and vowel($Estring_j$)
$Outstring_i = Estring_i$ increment i and j
if not vowel($Astring_i$) and vowel($Estring_j$)
increment j but not i
if j < |*Estring*|
$Outstring_i = Estring_j$ increment i and j
otherwise
$Outstring_i = Estring_{i,}$ increment i and j
Finally if there is anything left of *Estring*,
strip all vowels from what is left
append *Estring* to end of *Outstring*

Figure 13. Algorithm for retaining matching vowels

The algorithm presented in figure 13 retains only those vowels that are represented in both strings. The algorithm is a variant of a sorted file merge.

5.5 Normalizing "ch" representations with a separate pass

This enhancement requires a separate pass. The name "Buchanan" is represented in Arabic as "bywkAnAn" and "Richard" is "rytshArd." Thus, whichever choice the software makes for the correct value of the English substring "ch," it will choose incorrectly some significant number of times. In one pass, every "ch" in the English string gets mapped to "tsh." In a separate pass, every "ch" in the English string is transformed into a "k."

5.6 Light Stemming

The light stemming performed here was to remove the first letter of the transliterated Arabic name if it matched the prefixes "b," "l" or "w" and run the algorithm another time if the match score was below the match threshold but above another lower threshold. The first two items are prepositions that attach to any noun. The third is a conjunction that attaches to any word. Full stemming for Arabic is a separate and non-trivial problem.

6 Results

The algorithm with all enhancements was implemented in perl and in Java. Figure 14 presents the results of the enhanced algorithm on the original baseline as compared with the baseline algorithm. The enhancements improved the F-score by 22%.

Algorithm	Recall	Precision	F-score
Baseline	19/29	19/23	0.7308
Enhancements	29/29	29/32	0.9508

Figure 14. Enhanced edit distance on original data set

6.1 Results with a larger data set

After trying the algorithm out on a couple more "toy" data sets with similar results, we used a more realistic data set, which I will call the *TDT data set*. This data set was composed of 577 Arabic names and 968 English names that had been manually extracted from approximately 250 Arabic and English news articles on common topics in a NIST TDT corpus. There are 272 common names. The number of strings on the English side that correctly

match an Arabic language string is 591. The actual number of matches in the set is 641, since many Arabic strings match to the same set of English names. For instance, "Edmond Pope" has nine variants in English and six variants in Arabic. This gives 36 correct matches for the six Arabic spellings of Edmond Pope.

We varied the match threshold for various combinations of the described enhancements. The plots of the F-score, precision and recall from these experiments using the *TDT data set* are shown in figures 15, 16, and 17.

7 Discussion

Figure 15 shows that simply adding the "character equivalency classes" (CEQ) to the baseline algorithm boosts the F-score from around 48% to around 72%. Adding all other enhancements to the baseline algorithm, without adding CEQ only improves the f-score marginally. Combining these same enhancements with the CEQ raises the f-score by roughly 7% to almost 80%.

When including CEQ, the algorithm has a peak performance with a threshold near 85%. When CEQ is not included, the algorithm has a peak performance when the match threshold is around 70%. The baseline algorithm will declare that the strings match at a cutoff of 70%. Because we are normalizing by dividing by the lengths of both strings, this allows strings to match when half of their letters do not match. The CEQ forces a structure onto which characters are an allowable mismatch before the threshold is applied. This apparently leads to a reduction in the number allowable mismatches when the match threshold is tested.

The time and space complexity of the baseline Levenshtein algorithm is a function of the length of the two input strings, being |s| * |t|. This makes the time complexity (N^2) where N is the size of the average input string. The enhancements described here add to the time complexity. The increase is an average two or three extra compares per character and thus can be factored out of any equation. The new time complexity is K(|s|*|t|) where K >= 3.

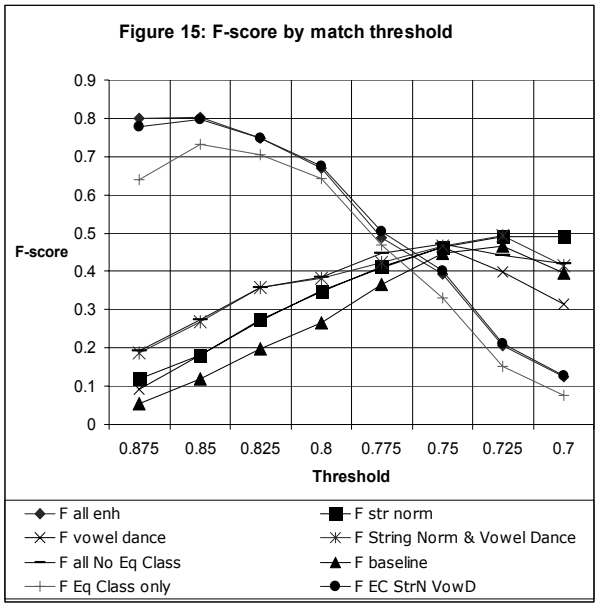
What we do here is the opposite of the approach taken by the Soundex and Editex algorithms. They try to reduce the complexity by collapsing groups of characters into a single super-class of characters. The algorithm here does some of that with the steps that normalize the strings. However, the largest boost in performance is with CEQ, which expands the number of allowable cross-language matches for many characters.

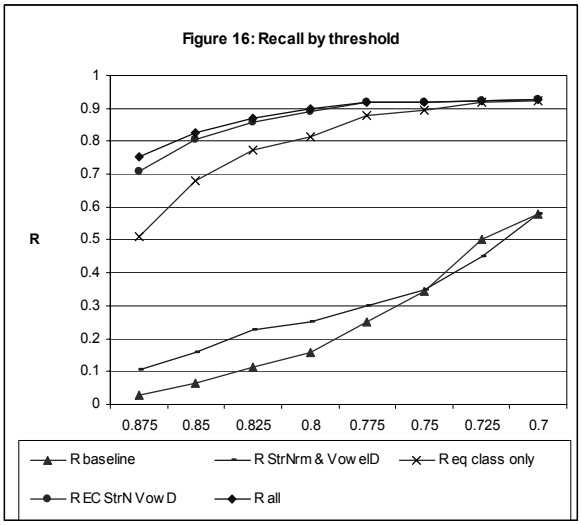
One could expect that increasing the allowable number of matches would over-generate, raising the recall while lowering the precision.

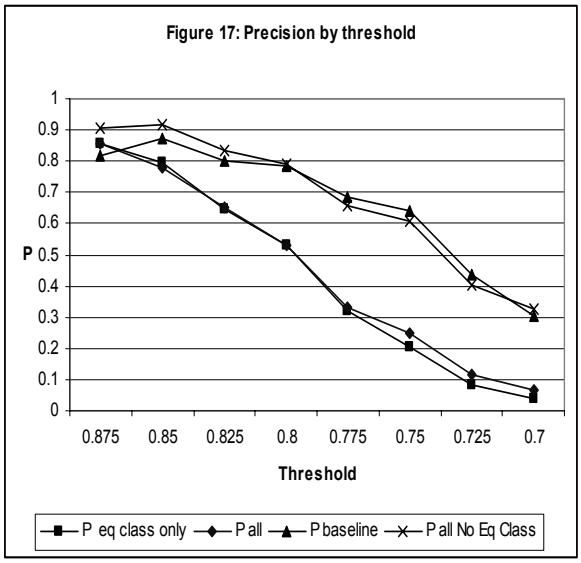
Referring to Figure 8, we see that's ome Arabic graphemes map to overlapping sets of characters in the English language strings.

Arabic جـ can be realized, as either [j] or [g], and one of the reflexes in English for Arabic ق can be [g] as well. How do we differentiate the one from the other? Quite simply, the Arabic input is not random data. Those dialects that produce ق as a [g] will as a rule not produce جـ as [g] and vice versa. The Arabic pronunciation of the string determines the correct alternation of the two characters for us *as it is written in English*. On a string-by-string basis, it is very unlikely that the two representations will conflict. The numbers show that by adding CEQ, the baseline algorithm's recall at threshold of 72.5%, goes from 57% to around 67% at a threshold of 85% for Arabic to English cross-linguistic name matching. Combining all of the enhancements raises the recall at a threshold of 85%, to 82%. As previously noted, augmenting the baseline algorithm with all enhancements except CEQ, does improve the performance dramatically. CEQ combines well with the other enhancements.

It is true that there is room for a lot improvement with an f-score of 80%. However, anyone doing cross-linguistic name matches would probably benefit by implementing some form of the character equivalence classes detailed here.







References

Basis Technology. 2004. Arabic Transliteration Module. Artrans documentation. (The documentation is available for download at http://www.basistech.com/arabic-editor.)

Bilenko, Mikael, Mooney, Ray, Cohen, William W., Ravikumar, Pradeep and Fienberg, Steve. 2003. Adaptive Name-Matching. in *Information Integration in IEEE Intelligent Systems,* 18(5): 16-23.

Buckwalter, Tim. 2002. Arabic Transliteration. http://www.qamus.org/transliteration.htm.

Cohen, William W., Ravikumar, Pradeep and Fienberg, Steve. 2003. A Comparison of String Distance Metrics for Name-Matching Tasks. *IIWeb 2003: 73-78*.

Jackson, Peter and Moulinier, Isabelle. 2002 . *Natural Language Processing for Online Applications: Text Retrieval, Extraction, and Categorization (Natural Language Processing, 5)*. John Benjamins Publishing.

Jurafsky, Daniel, and James H. Martin. 2000. *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition, and Computational Linguistics*. Prentice-Hall.

Knuth, Donald E. 1973. *The Art of Computer Programming, Volume 3: Sorting and Searching*. . Addison-Wesley Publishing Company,

Ukonnen, E. 1992. Approximate string-matching with q-grams and maximal matches. Theoretical Computer Science, 92: 191-211.

Wright, W. 1967. *A Grammar of the Arabic Language*. Cambridge. Cambridge University Press.

Zobel, Justin and Dart, Philip. 1996. Phonetic string matching: Lessons from information retrieval.in *Proceedings of the Eighteenth ACM SIGIR International Conference on Research and Development in Information Retrieval*, Zurich, Switzerland, August 1996, pp. 166-173.