Persian to English Back-Transliteration

Anonymised

1 Introduction

Back transliteration is the process of converting transliterated words back into their original form. This report analyses the performance of basic global edit distance algorithms applied to the task of Back-Transliterating from a Farsi (modified Arabic script) transliteration, to its matching original Latin word. The dataset, as curated by [1][2], comprises a training set of 13,000 word pairs, the second of each pair an English word in lowercase, and the first the matching uppercase Latinate representation (DIN31635, with a few small changes) of the Farsi transliteration. In this paper this is referred to as Farsi representation/script/characters, or just simply 'Farsi'.

This task is particularly difficult given that the Latin characters and their Farsi matches have a many-to-many relationship. For example, `ebrahim` and `ibrahim` are both represented in Farsi script by `ABRAHYM`, and the Farsi character `V` represents Latin characters `o`, `u`, `v`, and `w`, as demonstrated by the Latin name `augusto` being represented by `AGVSTV` in Farsi. Additionally, null correspondences, where a Latin character is simply removed from the Farsi representation, are also common. This means that a simple character to character mapping is not sufficient to perform Farsi to English back-transliteration.

Note that reproducing a Transliteration dictionary by any means would not solve the more general problem, wherein there is a continuous assimilation of new transliterations that are not yet added to any dictionary [3]. Therefore, this report explores a variety of modifications to an approximate matching algorithm - and its suitability to this problem.

2 Methodology

2.1 Global Edit Distance

The global edit distance between each possible matching word pair was found, and the word with the lowest edit distance was returned as the most likely match. Where ties occurred, matches were sub-sorted by bi-gram similarity [4]. Note that each system predicts a Latin name for every given transliteration. The Needleman-Wunsch Algorithm [5] was used to complete this in an efficient manner. The scoring parameters tested included:

- Levenshtein distance[1] (LD) [7] [6]
- Damerau-Levenshtein distance[1] (DLD) [7][8]
- Normalised Levenshtein[2] Distance (NLD) [7]
- Custom Parameter Distance (CPD)

2.2 Established Parameters

While LD provides a "true" distance between words, DLD attempts to improve on this by taking into account character transpositions. This does not make so much sense in the context of matching transliterations, where transpositions are unlikely to occur between matching word pairs, but it was tested nonetheless to explore the empirical results. Normalised Levenshtein Distance (NLD) attempts to normalise LD by taking into account the comparative lengths of the two sequences that are being compared.

2.3 Heavily Customised Parameters

Finally, a method that used LD, and also took into account a large number of observations about the valid and invalid transformations in matching word-pairs, was attempted. This approach sought to minimise distance penalties for

[1] LD and DLD imported from the jellyfish Python Library [6]

[2] Imported from the distance Python Library [9]

valid transformations and maximise penalties for invalid character transformations. Examples of these rules include the following:

- Vowels are often swapped for one another, so their replacements shouldn't incur a significant penalty a penalty of 0.5 was used, so as to continue to discourage a significant number of vowel edits.
- A deletion or replacement of certain characters from Farsi to Latin never occurs (`m`, `n`, `d`, `l`, etc.) and as such their removal should incur a significant penalty, to discourage their removal a penalty of 3 was used.
- A `ph` in a Latin word is always translated to an `f` in Farsi, so if a `f` matches with a , then we return -1, to make up for the deletion or replacement of the `h`.

2.4 Phonetic Distance Method

Using Levenshtein Distance to compare custom phonetic representations was also attempted Soundex, as outlined in Russell and Odell's patent [10] was unsatisfactory however this method will focus on the orthographic methods in this work.

3 Evaluation

In this section I present an evaluation of the effectiveness of the methods outlined above.

3.1 Accuracy

As a method of evaluation, Accuracy was selected given that the usefulness of the program is determined by its prediction given a transliteration, not a list of top ranked predictions. Accuracy was calculated by taking a random sample of 500 Farsi words from training.txt, using the string `bestseed` as a seed, and comparing the predicted Latin word for each given Farsi transliteration to the actual matching Latin word, for each method.

3.2 Precision

As a second method of evaluation, to further inform our understanding of the relative performance of the various methods, Precision at 3 (P@3) was found. P@3 was calculated by taking the same sample from above, and determining whether or not the matching Latin word was contained in the top 3 ranked results for each method.

3.3 Results

The results are presented below:

Method	P@3	Accuracy
LD	0.328	0.244
DLD	0.326	0.242
NLD	0.442	0.372
CPD	0.76	0.646

Table 1: Approximated Efficiency

3.3.1 Levenshtein Distance & Damerau-Levenshtein Distance

As predicted earlier, we see that LD slightly outperforms DLD. This is likely due to DLD not penalising character transpositions, even when there is no circumstance in which a transposition would occur in a transliteration, meaning some words are less accurately and precisely predicted than in LD. An example to illustrate:

> Given `ABRKRVMBY`, LD is equal (LD=5) for both `abraham` and `barraby`, but DLD prefers `barraby` to `abraham` (DLD=4, 5, respectively). It's clear that the transposition of the ab in `abraham` has been considered transposed in `barraby`, however this is clearly not a desirable outcome; transpositions should not occur.

3.3.2 Normalised Levenshtein Distance

NLD, on the other hand, reports significant improvements in comparison to LD and DLD. This is likely due to the primary problem of GED, being that it prefers shorter strings when matching. An example to illustrate:

> Given Farsi transliteration `ADVNCR`, LD prefers `acuner` (LD=3) to adventure (LD=4), whereas NLD prefers `adventure` to `acuner` (NDLD=0.44, 0.5, respectively). This is because NLD prefers, in the case of equal LDs, longer alignments, given that it's calculated to be the ratio of edit distance as a proportion of shortest alignment length.

The non-normalised GED bias is particularly significant here given that many Farsi transliterations are significantly shorter than their Latin counterparts; the average difference in

length of matching pairs is 1.15 characters, the maximum is 6 characters.

The performance difference might also be due to NLD's increased ability to differentiate between edit distances of different significance, given that it uses a ratio along a continuous set of values, rather than discrete distances.

3.3.3 Custom Parameter Distance

CPD reports significant improvements in comparison to NLD. One might note the very significant increase in Precision, however this result should be viewed sceptically, and more rigorous testing should be run to confirm the validity of such results. However, if confirmed, this is a substantial improvement over any other method. This method was a manual, and tedious approach, as it involved a great deal of testing and scanning of the data, but it proved the most successful of any method attempted. Often, this was more effective than NLD, as shown:

> Given `ABRBAC` (matching `eberbach`), NLD prefers `aboba` to `eberbach` (0.33, 0.375, respectively), while the custom approach prefers `eberbach` by a significant margin (1.5, 6, respectively).

Clearly, the custom parameters are preferred in this circumstance, as they take into account the knowledge that deleting r or c, or replacing r with o are invalid transformations.

Other times, NLD is preferred:

> Given `ABLY`, the custom approach doesn't differentiate between `ably` and `abele`, and ranks `abele` first (0.5, 0.5, respectively), whereas NLD clearly prefers `ably` to `abele` (0.2, 0.4, respectively).

The custom parameters are far from a perfect weighting and are not effective in many circumstances, but they do show clear effectiveness gains over the other, traditional GED parameters.

4 Improvements

Potential improvements considered include:

1. An improved method of tie breaking words with equal edit distances; perhaps the comparative likelihood of different transformations could be systematically taken into account.
2. A systematic method of deriving the matching matrix scores would likely yield a significant improvement over the CPD. Characters of each candidate pair could be aligned, after which Supervised Learning techniques could be used to determine the optimal scoring parameter for each aligned grapheme to all possible aligned graphemes.
3. An optimal Phonetic representation of the word pairs could be attempted; a Supervised learning technique could be used to determine the optimal character groupings and length, after which edit distances between representations could be compared and used to rank candidates.
4. A naïve Bayesian calculation of matching grapheme likelihoods could be used to determine how likely a candidate pair is a match. This was attempted using bigrams and trigrams, and while it proved relatively effective, the accurate calculation of probability was less effective than an incorrect probability measure, and proper character alignment was not achieved, so it was not reported above. This attempt was used to generate answers for the project's associated Kaggle competition (comedic username: HaramSnackPack).
5. A phoneme-based probability method, like the one demonstrated in [11].

5 Conclusions

This report established that it is possible to improve the back-transliteration from Persian Transliterations achieved by basic global edit distance-based approximate matching algorithms by optimising their edit parameters, or substitution matrix. However, the methods attempted in this report proved highly imprecise, especially in comparison to results from the Machine Translation community. Improvements to this approach, especially as suggested in section 4(2) of this report, should be performed to

find the theoretical limits of the global edit distance and the Needleman-Wunsch [5] algorithm in Persian back-transliteration, and perhaps in transliteration more generally.

References

[1] S. Karimi, A. Turpin, and F. Scholer, "English to persian transliteration," in *International Symposium on String Processing and Information Retrieval*, pp. 255–266, Springer, 2006.

[2] S. Karimi, A. Turpin, and F. Scholer, "Corpus effects on the evaluation of automated transliteration systems," in *ANNUAL MEETING-ASSOCIATION FOR COMPUTATIONAL LINGUISTICS*, vol. 45, p. 640, 2007.

[3] S. Bilac and H. Tanaka, "Improving back-transliteration by combining information sources," in *International Conference on Natural Language Processing*, pp. 216–223, Springer, 2004.

[4] G. P. Michel Albert, "NGram: The ngram class extends the python 'set' class with efficient fuzzy search for members by means of an n-gram similarity measure. it also has static methods to compare a pair of strings. Python," 2007–. [Online; accessed 7–04–2017].

[5] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of molecular biology*, vol. 48, no. 3, pp. 443–453, 1970.

[6] M. S. James Turk, "Jellyfish: a python library for doing approximate and phonetic matching of strings. Python," 2010–. [Online; accessed 7–04–2017].

[7] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet physics doklady*, vol. 10, pp. 707–710, 1966.

[8] F. J. Damerau, "A technique for computer detection and correction of spelling errors," *Communications of the ACM*, vol. 7, no. 3, pp. 171–176, 1964.

[9] M. Meyer, "distance: This package provides helpers for computing similarities between arbitrary sequences. included metrics are levenshtein, hamming, jaccard, and sorensen distance, plus some bonuses. all distance computations are implemented in pure python, and most of them are also implemented in c. Python," 2013–. [Online; accessed 7–04–2017].

[10] M. Odell and R. Russell, "The soundex coding system," *US Patents*, vol. 1261167, 1918.

[11] K. Knight and J. Graehl, "Machine transliteration," *Computational Linguistics*, vol. 24, no. 4, pp. 599–612, 1998.