PMI Levenshtein distance

Wilbert Heeringa

April 16th, 2024

Algorithm

In LED-A, PMI Levenshtein is implemented in almost the same way as described in Wieling (2012). Just like Wieling (2012), the algorithm is VC-sensitive and we use the so-called diagonal-exclusive version (see p. 19, 25-27). However, the swap operation is not implemented in LED-A.

The algorithm learns the weights (segment distances) from the alignments. This is done through a series of iterations. In each iteration Levenshtein distances of *k* word pairs for each of (*n* * (*n*-1))/2 dialect pairs of *n* dialects are calculated.

In the first iteration all insertions, deletions and substitutions have a weight of 1. However, since the algorithm is VC-sensitive, 'forbidden' sound segment combinations have a distance of 99999 in the initial segment distance matrix. These high distances prevent vowels to be aligned with consonants and vice versa. The hight distances are kept troughout all iterations.

In LED-A the user can choose to allow [i], [u], [j] and [w] to match with anything (and vice versa), and [ə] and [ɐ] to match with a sonorant (and vice versa).

When the distances are calculated, the alignments are examined. Segment pairs that occur at least one time in an alignment get as PMI value:

```
log2((pxy/(px*py)) + 1)
```

where:

- pxy is estimated by calculating the number of times sound segments x and y occur in correspondence in two aligned pronunciations X and Y, divided by the total number of aligned segments (i.e. the relative occurrence of the aligned sound segments x and y in the whole dataset);

- px and py are estimated as the number of times sound segment x (or y) occurs, divided by the total number of segment occurrences (i.e. the relative occurrence of sound segments x or y in the whole dataset); dividing by this term normalizes the correspondence frequency with respect to the frequency expected if x and y are statistically independent (see Wieling 2012, p. 18).

Allowed segment combinations that are not found in any alignment are given as ‘PMI’ value:

```
log2(0 + 1) = 0
```

The PMI distances are scaled between 0 and 1 by calculating:

```
(maxPMI - PMI) / (maxPMI - minPMI)
```

so that most frequent segment pairs get a distance of 0, and the least frequent segment pairs get a distance of 1.

Note that both the distances of segment pairs with equal segments and the distances of segment pairs that have the highest frequency (i.e. maxPMI) are 0.

Note that also distances between 'nothing' and segments (representating insertions) and between segments and 'nothing' (representing deletions) are likewise calculated. In fact, 'nothing' is treated as a sound segment.

In the next iteration Levenshtein distances of *k* word pairs for each of (*n* * (*n*-1))/2 dialect pairs of *n* dialects are calculated again. But now the segment distances (i.e.scaled PMI distances) that were calculated in the previous iteration are used as operation weights. The resulting alignments are checked and new segment distances are calculated.

The iterations are repeated until the correlation between the segment distances that were calculated in the current iteration and the segment distances that were calculated in the previous iteration (and used in the current iteration) is equal to or lower than the previous obtained correlation. In other words, iterations are repeated as long as the correlation is increasing.

When correlating the segment distances calculated in the current iteration with the segment distances in the previous iteration, the high distances between segments of 'forbidden' pairs segment pairs are not considered. Here we differ from the approach of Wieling (2012). Note also that allowed segment pairs that did not occur in the alignment get the (scaled) maximum PMI distance of 1. In the approach of Wieling (2012) these pairs get a high distance so that they will not occur any more in subsequent iterations. In our approach, a segment pair that was not found in a current iteration is theoretically not completely blocked, but can still appear in subsequent iterations.

Normalization

In LED-A the user can choose the raw Levenshtein distances to be divided by the alignment length. In order to do this, first the maximum weight for indel (=**in**sertion and **del**etion) slots and for substitution slots is determined. We call them respectively maxIndel and maxSubst. Then the length of the alignment is equal to:

```
number of indels × maxIndel +  number of substitutions (incl. matches) × maxSubst
```

In other words: the length of the alignment is equal to the sum of all maximum penalties per slot. When using PMI, maxIndel and maxSubst are recalculated in each iteration.

References

Heeringa, Wilbert, Van Heuven, Vincent & Van de Velde, Hans (2022), *LED-A: Levenshtein Edit Distance App* [Computer program]. Retrieved 2 January 2023 from https://www.led-a.org.

Ristad, E.S. & Yianilos, P.N. (1998). Learning string-edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20:522–532.

Wieling, M., Prokić, J. & Nerbonne, J. (2009). Evaluating the pairwise alignment of pronunciations. In Borin, L. and Lendvai, P., editors, *Proceedings of the EACL 2009 Workshop on Language Technology and Resources for Cultural Heritage, Social Sciences, Humanities, and Education, (LaTeCH - SHELT&R 2009). Workshop at the 12th Meeting of the European Chapter of the Association for Computational Linguistics. Athens, 30 March 2009*, pages 26—34.

Wieling, M.B. (2012). *A quantitative approach to social and geographical dialect variation*. PhD thesis University of Groningen.

Wieling, M., Margaretha, E. & Nerbonne, J. (2012). Inducing a measure of phonetic similarity from pronunciation variation. *Journal of Phonetics*, 40(2), 307-314.