Evaluating the pairwise string alignment of pronunciations

Martijn Wieling
University of Groningen
The Netherlands
m.b.wieling@rug.nl

Jelena Prokić
University of Groningen
The Netherlands
j.prokic@rug.nl

John Nerbonne
University of Groningen
The Netherlands
j.nerbonne@rug.nl

Abstract

Pairwise string alignment (PSA) is an important general technique for obtaining a measure of similarity between two strings, used e.g., in dialectology, historical linguistics, transliteration, and in evaluating name distinctiveness. The current study focuses on evaluating different PSA methods at the alignment level instead of via the distances it induces. About 3.5 million pairwise alignments of Bulgarian phonetic dialect data are used to compare four algorithms with a manually corrected gold standard. The algorithms evaluated include three variants of the Levenshtein algorithm as well as the Pair Hidden Markov Model. Our results show that while all algorithms perform very well and align around 95% of all alignments correctly, there are specific qualitative differences in the (mis)alignments of the different algorithms.

1 Introduction

Our cultural heritage is not only accessible through museums, libraries, archives and their digital portals, it is alive and well in the varied cultural habits practiced today by the various peoples of the world. To research and understand this cultural heritage we require instruments which are sensitive to its signals, and, in particular sensitive to signals of common provenance. The present paper focuses on speech habits which even today bear signals of common provenance in the various dialects of the world's languages, and which have also been recorded and preserved in major archives of folk culture internationally. We present work in a research line which seeks to develop digital instruments capable of detecting common provenance among pronunciation habits, focusing in this paper on the issue of evaluating the quality of these instruments.

Pairwise string alignment (PSA) methods, like the popular Levenshtein algorithm (Levenshtein, 1965) which uses insertions (alignments of a segment against a gap), deletions (alignments of a gap against a segment) and substitutions (alignments of two segments) often form the basis of determining the distance between two strings. Since there are many alignment algorithms and specific settings for each algorithm influencing the distance between two strings (Nerbonne and Kleiweg, 2007), evaluation is very important in determining the effectiveness of the distance methods.

Determining the distance (or similarity) between two phonetic strings is an important aspect of dialectometry, and alignment quality is important in applications in which string alignment is a goal in itself, for example, determining if two words are likely to be cognate (Kondrak, 2003), detecting confusable drug names (Kondrak and Dorr, 2003), or determining whether a string is the transliteration of the same name from another writing system (Pouliquen, 2008).

In this paper we evaluate string distance measures on the basis of data from dialectology. We therefore explain a bit more of the intended use of the pronunciation distance measure.

Dialect atlases normally contain a large number of pronunciations of the same word in various places throughout a language area. All pairs of pronunciations of corresponding words are compared in order to obtain a measure of the aggregate linguistic distance between dialectal varieties (Heeringa, 2004). It is clear that the quality of the measurement is of crucial importance.

Almost all evaluation methods in dialectometry focus on the aggregate results and ignore the individual word-pair distances and individual alignments on which the distances are based. The focus on the aggregate distance of 100 or so word

Proceedings of the EACL 2009 Workshop on Language Technology and Resources for Cultural Heritage, Social Sciences, Humanities, and Education –LaTeCH – SHELT&R 2009, pages 26–34,
Athens, Greece, 30 March 2009.

pairs effectively hides many differences between methods. For example, Heeringa et al. (2006) find no significant differences in the degrees to which several pairwise string distance measures correlate with perceptual distances when examined at an aggregate level. Wieling et al. (2007) and Wieling and Nerbonne (2007) also report almost no difference between different PSA algorithms at the aggregate level. It is important to be able to evaluate the different techniques more sensitively, which is why this paper examines alignment quality at the segment level.

Kondrak (2003) applies a PSA algorithm to align words in different languages in order to detect cognates automatically. Exceptionally, he does provide an evaluation of the string alignments generated by different algorithms. But he restricts his examination to a set of only 82 gold standard pairwise alignments and he only distinguishes correct and incorrect alignments and does not look at misaligned phones.

In the current study we introduce and evaluate several alignment algorithms more extensively at the alignment level. The algorithms we evaluate include the Levenshtein algorithm (with syllabicity constraint), which is one of the most popular alignment methods and has successfully been used in determining pronunciation differences in phonetic strings (Kessler, 1995; Heeringa, 2004). In addition we look at two adaptations of the Levenshtein algorithm. The first adaptation includes the swap-operation (Wagner and Lowrance, 1975), while the second adaptation includes phonetic segment distances, which are generated by applying an iterative pointwise mutual information (PMI) procedure (Church and Hanks, 1990). Finally we include alignments generated with the Pair Hidden Markov Model (PHMM) as introduced to language studies by Mackay and Kondrak (2005). They reported that the Pair Hidden Markov Model outperformed ALINE, the best performing algorithm at the alignment level in the aforementioned study of Kondrak (2003). The PHMM has also successfully been used in dialectology by Wieling et al. (2007).

2 Dataset

The dataset used in this study consists of 152 words collected from 197 sites equally distributed over Bulgaria. The transcribed word pronunciations include diacritics and suprasegmentals (e.g., intonation). The total number of different phonetic types (or segments) is 98.[1]

The gold standard pairwise alignment was automatically generated from a manually corrected gold standard set of N multiple alignments (see Prokić et al., 2009) in the following way:

- Every individual string (including gaps) in the multiple alignment is aligned with every other string of the same word. With 152 words and 197 sites and in some cases more than one pronunciations per site for a certain word, the total number of pairwise alignments is about 3.5 million.
- If a resulting pairwise alignment contains a gap in both strings at the same position (a gap-gap alignment), these gaps are removed from the pairwise alignment. We justify this, reasoning that no alignment algorithm may be expected to detect parallel deletions in a single pair of words. There is no evidence for this in the single pair.

To make this clear, consider the multiple alignment of three Bulgarian dialectal variants of the word 'I' (as in 'I am'):

j	ˈɑ	s		
	ˈɑ	z	i	
j	ˈɑ			

Using the procedure above, the three generated pairwise alignments are:

j	ˈɑ	s		j	ˈɑ	s			ˈɑ	z	i
	ˈɑ	z	i	j	ˈɑ			j	ˈɑ		

3 Algorithms

Four algorithms are evaluated with respect to the quality of their alignments, including three variants of the Levenshtein algorithm and the Pair Hidden Markov Model.

3.1 The VC-sensitive Levenshtein algorithm

The Levenshtein algorithm is a very efficient dynamic programming algorithm, which was first introduced by Kessler (1995) as a tool for computationally comparing dialects. The Levenshtein distance between two strings is determined by counting the minimum number of edit operations (i.e. insertions, deletions and substitutions) needed to transform one string into the other.

[1]The dataset is available online at the website http://www.bultreebank.org/BulDialects/

For example, the Levenshtein distance between [jˈɑs] and [ˈɑzi], two Bulgarian dialectal variants of the word 'I' (as in 'I am'), is 3:

jˈɑs	delete j	1
ˈɑs	subst. s/z	1
ˈɑz	insert i	1
ˈɑzi		
		3

The corresponding alignment is:

j	ˈɑ	s	
	ˈɑ	z	i
1		1	1

The Levenshtein distance has been used frequently and successfully in measuring linguistic distances in several languages, including Irish (Kessler, 1995), Dutch (Heeringa, 2004) and Norwegian (Heeringa, 2004). Additionally, the Levenshtein distance has been shown to yield aggregate results that are consistent (Cronbach's $\alpha = 0.99$) and valid when compared to dialect speakers judgements of similarity ($r \approx 0.7$; Heeringa et al., 2006).

Following Heeringa (2004), we have adapted the Levenshtein algorithm slightly, so that it does not allow alignments of vowels with consonants. We refer to this adapted algorithm as the VC-sensitive Levenshtein algorithm.

3.2 The Levenshtein algorithm with the swap operation

Because metathesis (i.e. transposition of sounds) occurs relatively frequently in the Bulgarian dialect data (in 21 of 152 words), we extend the VC-sensitive Levenshtein algorithm as described in section 3.1 to include the swap-operation (Wagner and Lowrance, 1975), which allows two adjacent characters to be interchanged. The swap-operation is also known as a transposition, which was introduced with respect to detecting spelling errors by Damerau (1964). As a consequence the Damerau distance refers to the minimum number of insertions, deletions, substitutions and transpositions required to transform one string into the other. In contrast to Wagner and Lowrance (1975) and in line with Damerau (1964) we restrict the swap operation to be only allowed for string X and Y when $x_i = y_{i+1}$ and $y_i = x_{i+1}$ (with x_i being the token at position i in string X):

x_i	x_{i+1}
y_i	y_{i+1}
$><$	1

Note that a swap-operation in the alignment is indicated by the symbol '><'. The first number following this symbol indicates the cost of the swap-operation.

Consider the alignment of [vrˈɤ] and [vˈɤr],[2] two Bulgarian dialectal variants of the word 'peak' (mountain). The alignment involves a swap and results in a total Levenshtein distance of 1:

v	r	ˈɤ
v	ˈɤ	r
	><	1

However, the alignment of the transcription [vrˈɤ] with another dialectal transcription [vˈar] does not allow a swap and yields a total Levenshtein distance of 2:

v		r	ˈɤ
v	ˈa	r	
	1		1

Including just the option of swapping identical segments in the implementation of the Levenshtein algorithm is relatively easy. We set the cost of the swap operation to one[3] plus twice the cost of substituting x_i with y_{i+1} plus twice the cost of substituting y_i with x_{i+1}. In this way the swap operation will be preferred when $x_i = y_{i+1}$ and $y_i = x_{i+1}$, but not when $x_i \neq y_{i+1}$ and/or $y_i \neq x_{i+1}$. In the first case the cost of the swap operation is 1, which is less than the cost of the alternative of two substitutions. In the second case the cost is either 3 (if $x_i \neq y_{i+1}$ or $y_i \neq x_{i+1}$) or 5 (if $x_i \neq y_{i+1}$ and $y_i \neq x_{i+1}$), which is higher than the cost of using insertions, deletions and/or substitutions.

Just as in the previous section, we do not allow vowels to align with consonants (except in the case of a swap).

3.3 The Levenshtein algorithm with generated segment distances

The VC-sensitive Levenshtein algorithm as described in section 3.1 only distinguishes between vowels and consonants. However, more sensitive segment distances are also possible. Heeringa (2004) experimented with specifying phonetic segment distances based on phonetic features and

[2] We use transcriptions in which stress is marked on stressed vowels instead of before stressed syllables. We follow in this the Bulgarian convention instead of the IPA convention.

[3] Actually the cost is set to 0.999 to prefer an alignment involving a swap over an alternative alignment involving only regular edit operations.

also based on acoustic differences derived from spectrograms, but he did not obtain improved results at the aggregate level.

Instead of using segment distances as these are (incompletely) suggested by phonetic or phonological theory, we tried to determine the sound distances automatically based on the available data. We used pointwise mutual information (PMI; Church and Hanks, 1990) to obtain these distances. It generates segment distances by assessing the degree of statistical dependence between the segments x and y:

$$\text{PMI}(x, y) = \log_2 \left(\frac{p(x, y)}{p(x)\, p(y)} \right) \qquad (1)$$

Where:

- $p(x, y)$: the number of times x and y occur at the same position in two aligned strings X and Y, divided by the total number of aligned segments (i.e. the relative occurrence of the aligned segments x and y in the whole dataset). Note that either x or y can be a gap in the case of insertion or deletion.
- $p(x)$ and $p(y)$: the number of times x (or y) occurs, divided by the total number of segment occurrences (i.e. the relative occurrence of x or y in the whole dataset). Dividing by this term normalizes the empirical frequency with respect to the frequency expected if x and y are statistically independent.

The greater the PMI value, the more segments tend to cooccur in correspondences. Negative PMI values indicate that segments do not tend to cooccur in correspondences, while positive PMI values indicate that segments tend to cooccur in correspondences. The segment distances can therefore be generated by subtracting the PMI value from 0 and adding the maximum PMI value (i.e. lowest distance is 0). In that way corresponding segments obtain the lowest distance.

Based on the PMI value and its conversion to segment distances, we developed an iterative procedure to automatically obtain the segment distances:

1. The string alignments are generated using the VC-sensitive Levenshtein algorithm (see section 3.1).[4]
2. The PMI value for every segment pair is calculated according to (1) and subsequently transformed to a segment distance by subtracting it from zero and adding the maximum PMI value.
3. The Levenshtein algorithm using these segment distances is applied to generate a new set of alignments.
4. Step 2 and 3 are repeated until the alignments of two consecutive iterations do not differ (i.e. convergence is reached).

The potential merit of using PMI-generated segment distances can be made clear by the following example. Consider the strings [vˈʏn] and [vˈʏɳkə], Bulgarian dialectal variants of the word 'outside'. The VC-sensitive Levenshtein algorithm yields the following (correct) alignment:

v	ˈʏ	n		
v	ˈʏ	ɳ	k	ə
		1	1	1

But also the alternative (incorrect) alignment:

v	ˈʏ		n	
v	ˈʏ	ɳ	k	ə
		1	1	1

The VC-sensitive Levenshtein algorithm generates the erroneous alignment because it has no way to identify that the consonant [n] is nearer to the consonant [ɳ] than to the consonant [k]. In contrast, the Levenshtein algorithm which uses the PMI-generated segment distances only generates the correct first alignment, because the [n] occurs relatively more often aligned with [ɳ] than with [k] so that the distance between [n] and [ɳ] will be lower than the distance between [n] and [k]. The idea behind this procedure is similar to Ristad's suggestion to learn segment distances for edit distance using an expectation maximization algorithm (Ristad and Yianilos, 1998). Our approach differs from their approach in that we only learn segment distances based on the alignments generated by the VC-sensitive Levenshtein algorithm, while Ristad and Yianilos (1998) learn segment distances by considering all possible alignments of two strings.

3.4 The Pair Hidden Markov Model

The Pair Hidden Markov Model (PHMM) also generates alignments based on automatically generated segment distances and has been used suc-

[4] We also used the Levenshtein algorithm without the vowel-consonant restriction to generate the PMI values, but this had a negative effect on the performance.

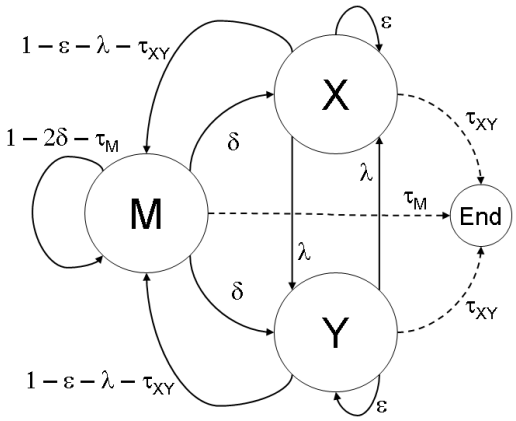

Figure 1: Pair Hidden Markov Model. Image courtesy of Mackay and Kondrak (2005).

cessfully in language studies (Mackay and Kondrak, 2005; Wieling et al., 2007).

A Hidden Markov Model (HMM) is a probabilistic finite-state transducer that generates an observation sequence by starting in an initial state, going from state to state based on transition probabilities and emitting an output symbol in each state based on the emission probabilities in that state for that output symbol (Rabiner, 1989). The PHMM was originally proposed by Durbin et al. (1998) for aligning biological sequences and was first used in linguistics by Mackay and Kondrak (2005) to identify cognates. The PHMM differs from the regular HMM in that it outputs two observation streams (i.e. a series of alignments of pairs of individual segments) instead of only a series of single symbols. The PHMM displayed in Figure 1 has three emitting states: the substitution ('match') state (M) which emits two aligned symbols, the insertion state (Y) which emits a symbol and a gap, and the deletion state (X) which emits a gap and a symbol.

The following example shows the state sequence for the pronunciations [jˈɑs] and [ˈɑzi] (English 'I'):

j	ˈɑ	s	
	ˈɑ	z	i
X	M	M	Y

Before generating the alignments, all probabilities of the PHMM have to be estimated. These probabilities consist of the 5 transition probabilities shown in Figure 1: ϵ, λ, δ, τ_{XY} and τ_M. In addition there are 98 emission probabilities for the insertion state and the deletion state (one for every segment) and 9604 emission probabilities for the substitution state. The probability of starting in one of the three states is set equal to the probability of going from the substitution state to that particular state. The Baum-Welch expectation maximization algorithm (Baum et al., 1970) can be used to iteratively reestimate these probabilities until a local optimum is found.

To prevent order effects in training, every word pair is considered twice (e.g., $w_a - w_b$ and $w_b - w_a$). The resulting insertion and deletion probabilities are therefore the same (for each segment), and the probability of substituting x for y is equal to the probability of substituting y for x, effectively yielding 4802 distinct substitution probabilities.

Wieling et al. (2007) showed that using Dutch dialect data for training, sensible segment distances were obtained; acoustic vowel distances on the basis of spectrograms correlated significantly ($r = -0.72$) with the vowel substitution probabilities of the PHMM. Additionally, probabilities of substituting a symbol with itself were much higher than the probabilities of substituting an arbitrary vowel with another non-identical vowel (*mutatis mutandis* for consonants), which were in turn much higher than the probabilities of substituting a vowel for a consonant.

After training, the well known Viterbi algorithm can be used to obtain the best alignments (Rabiner, 1989).

4 Evaluation

As described in section 2, we use the generated pairwise alignments from a gold standard of multiple alignments for evaluation. In addition, we look at the performance of a baseline of pairwise alignments, which is constructed by aligning the strings according to the Hamming distance (i.e. only allowing substitutions and no insertions or deletions; Hamming, 1950).

The evaluation procedure consists of comparing the alignments of the previously discussed algorithms including the baseline with the alignments of the gold standard. For the comparison we use the standard Levenshtein algorithm without any restrictions. The evaluation proceeds as follows:

1. The pairwise alignments of the four algorithms, the baseline and the gold standard are generated and standardized (see section 4.1). When multiple equal-scoring alignments are

generated by an algorithm, only one (i.e. the final) alignment is selected.

2. In each alignment, we convert each pair of aligned segments to a single token, so that every alignment of two strings is converted to a single string of segment pairs.

3. For every algorithm these transformed strings are aligned with the transformed strings of the gold standard using the standard Levenshtein algorithm.

4. The Levenshtein distances for all these strings are summed up resulting in the total distance between every alignment algorithm and the gold standard. Only if individual segments match completely the segment distance is 0, otherwise it is 1.

To illustrate this procedure, consider the following gold standard alignment of [vlˈɤk] and [vˈɤlk], two Bulgarian dialectal variants of the word 'wolf':

v	l	ˈɤ	k	
v	ˈɤ	l	k	

Every aligned segment pair is converted to a single token by adding the symbol '/' between the segments and using the symbol '-' to indicate a gap. This yields the following transformed string:

v/v l/ˈɤ ˈɤ/l k/k

Suppose another algorithm generates the following alignment (not detecting the swap):

v	l	ˈɤ		k
v		ˈɤ	l	k

The transformed string for this alignment is:

v/v l/- ˈɤ/ˈɤ -/l k/k

To evaluate this alignment, we align this string to the transformed string of the gold standard and obtain a Levenshtein distance of 3:

v/v	l/ˈɤ	ˈɤ/l		k/k
v/v	l/-	ˈɤ/ˈɤ	-/l	k/k
	1	1	1	

By repeating this procedure for all alignments and summing up all distances, we obtain total distances between the gold standard and every alignment algorithm. Algorithms which generate high-quality alignments will have a low distance from the gold standard, while the distance will be higher for algorithms which generate low-quality alignments.

4.1 Standardization

The gold standard contains a number of alignments which have alternative equivalent alignments, most notably an alignment containing an insertion followed by a deletion (which is equal to the deletion followed by the insertion), or an alignment containing a syllabic consonant such as [ˈɹ̩], which in fact matches both a vowel and a neighboring r-like consonant and can therefore be aligned with either the vowel or the consonant. In order to prevent punishing the algorithms which do not match the exact gold standard in these cases, the alignments of the gold standard and all alignment algorithms are transformed to one standard form in all relevant cases.

For example, consider the correct alignment of [vˈiɑ] and [vˈij], two Bulgarian dialectal variations of the English plural pronoun 'you':

v	ˈi		ɑ
v	ˈi	j	

Of course, this alignment is as reasonable as:

v	ˈi	ɑ	
v	ˈi		j

To avoid punishing the first, we transform all insertions followed by deletions to deletions followed by insertions, effectively scoring the two alignments the same.

For the syllabic consonants we transform all alignments to a form in which the syllabic consonant is followed by a gap and not vice versa. For instance, aligning [vˈɹ̩x] with [vˈɑrx] (English: 'peak') yields:

v		ˈɹ̩	x
v	ˈɑ	r	x

Which is transformed to the equivalent alignment:

v	ˈɹ̩		x
v	ˈɑ	r	x

5 Results

We will report both quantitative results using the evaluation method discussed in the previous section, as well as the qualitative results, where we focus on characteristic errors of the different alignment algorithms.

5.1 Quantitative results

Because there are two algorithms which use generated segment distances (or probabilities) in their alignments, we first check if these values are sensible and comparable to each other.

5.1.1 Comparison of segment distances

With respect to the PMI results (convergence was reached after 7 iterations, taking less than 5 CPU minutes), we indeed found sensible results: the average distance between identical symbols was significantly lower than the distance between pairs of different vowels and consonants ($t < -13, p < .001$). Because we did not allow vowel-consonants alignments in the Levenshtein algorithm, no PMI values were generated for those segment pairs.

Just as Wieling et al. (2007), we found sensible PHMM substitution probabilities (convergence was reached after 1675 iterations, taking about 7 CPU hours): the probability of matching a symbol with itself was significantly higher than the probability of substituting one vowel for another (similarly for consonants), which in turn was higher than the probability of substituting a vowel with a consonant (all t's $> 9, p < .001$).

To allow a fair comparison between the PHMM probabilities and the PMI distances, we transformed the PHMM probabilities to log-odds scores (i.e. dividing the probability by the relative frequency of the segments and subsequently taking the log). Because the residues after the linear regression between the PHMM similarities and PMI distances were not normally distributed, we used Spearman's rank correlation coefficient to assess the relationship between the two variables. We found a highly significant Spearman's $\rho = -.965\,(p < .001)$, which means that the relationship between the PHMM similarities and the PMI distances is very strong. When looking at the insertions and deletions we also found a significant relationship: Spearman's $\rho = -.736\,(p < .001)$.

5.1.2 Evaluation against the gold standard

Using the procedure described in section 4, we calculated the distances between the gold standard and the alignment algorithms. Besides reporting the total number of misaligned tokens, we also divided this number by the total number of aligned segments in the gold standard (about 16 million) to get an idea of the error rate. Note that the error rate is 0 in the perfect case, but might rise to nearly 2 in the worst case, which is an alignment consisting of only insertions and deletions and therefore up to twice as long as the alignments in the gold standard. Finally, we also report the total number of alignments (word pairs) which are not exactly equal to the alignments of the gold standard.

The results are shown in Table 1. We can clearly see that all algorithms beat the baseline and align about 95% of all string pairs correctly. While the Levenshtein PMI algorithm aligns most strings perfectly, it misaligns slightly more individual segments than the PHMM and the Levenshtein algorithm with the swap operation (i.e. it makes more segment alignment errors per word pair). The VC-sensitive Levenshtein algorithm in general performs slightly worse than the other three algorithms.

5.2 Qualitative results

Let us first note that it is almost impossible for any algorithm to achieve a perfect overlap with the gold standard, because the gold standard was generated from multiple alignments and therefore incorporates other constraints. For example, while a certain pairwise alignment could appear correct in aligning two consonants, the multiple alignment could show contextual support (from pronunciations in other varieties) for separating the consonants. Consequently, all algorithms discussed below make errors of this kind.

In general, the specific errors of the VC-sensitive Levenshtein algorithm can be separated into three cases. First, as we illustrated in section 3.3, the VC-sensitive Levenshtein algorithm has no way to distinguish between aligning a consonant with one of two neighboring consonants and sometimes chooses the wrong one (this also holds for vowels). Second, it does not allow alignments of vowels with consonants and therefore cannot detect correct vowel-consonant alignments such as correspondences of [u] with [v] initially. Third, for the same reason the VC-sensitive Levenshtein algorithm is also not able to detect metathesis of vowels with consonants.

The misalignments of the Levenshtein algorithm with the swap-operation can also be split in three cases. It suffers from the same two problems as the VC-sensitive Levenshtein algorithm in choosing to align a consonant incorrectly with one of two neighboring consonants and not being able to align a vowel with a consonant. Third, even though it aligns some of the metathesis cases correctly, it also makes some errors by incorrectly applying the swap-operation. For example, consider the alignment of [sˈirʲɪmi] and [sˈirʲnɪ], two Bulgarian dialectal variations of the word 'cheese', in which the swap-operation is applied:

Algorithm	Misaligned segments (error rate)	Incorrect alignments (%)
Baseline (Hamming algorithm)	2510094 (0.1579)	726844 (20.92%)
VC-sens. Levenshtein algorithm	490703 (0.0309)	191674 (5.52%)
Levenshtein PMI algorithm	399216 (0.0251)	**156440 (4.50%)**
Levenshtein swap algorithm	392345 (0.0247)	161834 (4.66%)
Pair Hidden Markov Model	**362423 (0.0228)**	160896 (4.63%)

Table 1: Comparison to gold standard alignments. All differences are significant ($p < 0.01$).

s	ˈi	rʲ	ɪ	n	i
s	ˈi	rʲ	n	ɪ	
0	0	0	><	1	1

However, the two ɪ's are not related and should not be swapped, which is reflected in the gold standard alignment:

s	ˈi	rʲ	ɪ	n	i
s	ˈi	rʲ		n	ɪ
0	0	0	1	0	1

The incorrect alignments of the Levenshtein algorithm with the PMI-generated segment distances are mainly caused by its inability to align vowels with consonants and therefore, just as the VC-sensitive Levenshtein algorithm, it fails to detect metathesis. On the other hand, using segment distances often solves the problem of selecting which of two plausible neighbors a consonant should be aligned with.

Because the PHMM employs segment substitution probabilities, it also often solves the problem of aligning a consonant to one of two neighbors. In addition, the PHMM often correctly aligns metathesis involving equal as well as similar symbols, even realizing an improvement over the Levenshtein swap algorithm. Unfortunately, many wrong alignments of the PHMM are also caused by allowing vowel-consonant alignments. Since the PHMM does not take context into account, it also aligns vowels and consonants which often play a role in metathesis when no metathesis is involved.

6 Discussion

This study provides an alternative evaluation of string distance algorithms by focusing on their effectiveness in aligning segments. We proposed, implemented, and tested the new procedure on a substantial body of data. This provides a new perspective on the quality of distance and alignment algorithms as they have been used in dialectology, where aggregate comparisons had been at times frustratingly inconclusive.

In addition, we introduced the PMI weighting within the Levenshtein algorithm as a simple means of obtaining segment distances, and showed that it improves on the popular Levenshtein algorithm with respect to alignment accuracy.

While the results indicated that the PHMM misaligned the fewest segments, training the PHMM is a lengthy process lasting several hours. Considering that the Levenshtein algorithm with the swap operation and the Levenshtein algorithm with the PMI-generated segment distances are much quicker to (train and) apply, and that they have only slightly lower performance with respect to the segment alignments, we actually prefer using those methods. Another argument in favor of using one of these Levenshtein algorithms is that it is *a priori* clearer what type of alignment errors to expect from them, while the PHMM algorithm is less predictable and harder to comprehend.

While our results are an indication of the good quality of the evaluated algorithms, we only evaluated the algorithms on a single dataset for which a gold standard was available. Ideally we would like to verify these results on other datasets, for which gold standards consisting of multiple or pairwise alignments are available.

Acknowledgements

We are grateful to Peter Kleiweg for extending the Levenshtein algorithm in the L04 package with the swap-operation. We also thank Greg Kondrak for providing the original source code of the Pair Hidden Markov Models. Finally, we thank Therese Leinonen and Sebastian Kürschner of the University of Groningen and Esteve Valls i Alecha of the University of Barcelona for their useful feedback on our ideas.

References

Leonard E. Baum, Ted Petrie, George Soules, and Norman Weiss. 1970. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov Chains. *The Annals of Mathematical Statistics*, 41(1):164–171.

Kenneth W. Church and Patrick Hanks. 1990. Word association norms, mutual information, and lexicography. *Computational Linguistics*, 16(1):22–29.

Fred J. Damerau. 1964. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7:171–176.

Richard Durbin, Sean R. Eddy, Anders Krogh, and Graeme Mitchison. 1998. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, United Kingdom, July.

Richard Hamming. 1950. Error detecting and error correcting codes. *Bell System Technical Journal*, 29:147–160.

Wilbert Heeringa, Peter Kleiweg, Charlotte Gooskens, and John Nerbonne. 2006. Evaluation of string distance algorithms for dialectology. In John Nerbonne and Erhard Hinrichs, editors, *Linguistic Distances*, pages 51–62, Shroudsburg, PA. ACL.

Wilbert Heeringa. 2004. *Measuring Dialect Pronunciation Differences using Levenshtein Distance*. Ph.D. thesis, Rijksuniversiteit Groningen.

Brett Kessler. 1995. Computational dialectology in Irish Gaelic. In *Proceedings of the seventh conference on European chapter of the Association for Computational Linguistics*, pages 60–66, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Grzegorz Kondrak and Bonnie Dorr. 2003. Identification of Confusable Drug Names: A New Approach and Evaluation Methodology. *Artificial Intelligence in Medicine*, 36:273–291.

Grzegorz Kondrak. 2003. Phonetic Alignment and Similarity. *Computers and the Humanities*, 37:273–291.

Vladimir Levenshtein. 1965. Binary codes capable of correcting deletions, insertions and reversals. *Doklady Akademii Nauk SSSR*, 163:845–848.

Wesley Mackay and Grzegorz Kondrak. 2005. Computing word similarity and identifying cognates with Pair Hidden Markov Models. In *Proceedings of the 9th Conference on Computational Natural Language Learning (CoNLL)*, pages 40–47, Morristown, NJ, USA. Association for Computational Linguistics.

John Nerbonne and Peter Kleiweg. 2007. Toward a dialectological yardstick. *Journal of Quantitative Linguistics*, 14:148–167.

Bruno Pouliquen. 2008. Similarity of names across scripts: Edit distance using learned costs of N-Grams. In Bent Nordström and Aarne Ranta, editors, *Proceedings of the 6th international Conference on Natural Language Processing (GoTal'2008)*, volume 5221, pages 405–416.

Jelena Prokić, Martijn Wieling, and John Nerbonne. 2009. Multiple sequence alignments in linguistics. In Piroska Lendvai and Lars Borin, editors, *Proceedings of the EACL 2009 Workshop on Language Technology and Resources for Cultural Heritage, Social Sciences, Humanities, and Education*.

Lawrence R. Rabiner. 1989. A tutorial on Hidden Markov Models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286.

Eric Sven Ristad and Peter N. Yianilos. 1998. Learning string-edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20:522–532.

Robert Wagner and Roy Lowrance. 1975. An extension of the string-to-string correction problem. *Journal of the ACM*, 22(2):177–183.

Martijn Wieling and John Nerbonne. 2007. Dialect pronunciation comparison and spoken word recognition. In Petya Osenova, editor, *Proceedings of the RANLP Workshop on Computational Phonology*, pages 71–78.

Martijn Wieling, Therese Leinonen, and John Nerbonne. 2007. Inducing sound segment differences using Pair Hidden Markov Models. In Mark Ellison John Nerbonne and Greg Kondrak, editors, *Computing and Historical Phonology: 9th Meeting of the ACL Special Interest Group for Computational Morphology and Phonology*, pages 48–56.