alineR: an R Package for Optimizing Feature-Weighted Alignments and Linguistic Distances

by Sean S. Downey, Guowei Sun and Peter Norquest

Abstract Linguistic distance measurements are commonly used in anthropology and biology when quantitative and statistical comparisons between words are needed. This is common, for example, when analyzing linguistic and genetic data. Such comparisons can provide insight into historical population patterns and they provide general insight into evolutionary processes. However, the most commonly used linguistic distances are derived from edit distances, which do not weight phonetic features that may, for example, represent smaller-scale patterns in linguistic evolution. Thus, computational methods for calculating feature-weighted linguistic distances are needed for linguistic, biological, and evolutionary applications; additionally, the linguistic distances presented here are generic and may have broader applications in fields such as text mining and search, as well as applications in psycholinguistics and morphology. To facilitate this research, we are making our software available as an open-source R software package that performs feature-weighted linguistic distance calculations. The package includes a supervised learning methodology that uses a genetic algorithm and manually determined alignments to estimate 13 linguistic parameters including feature weights and a skip penalty. Here we present the package and use it to demonstrate a supervised learning methodology to estimate the optimal linguistic parameters for a sample of Austronesian languages. Our results show that the methodology can estimate these parameters for both simulated language data and for real language data, that optimizing feature weights improves alignment accuracy by approximately 29%, and that optimization and significantly affects the resulting distance measurements. Availability: **alineR** is available on CRAN.

Introduction

Human speech patterns change through time in response to both cultural and demographic processes of speech communities such as migration and social contact. Analyzing differences among languages can provide insight into historical patterns and general processes of biological and cultural evolution (Pagel, 2012). Linguistic distances based on the comparison of two words are often used when quantitative analyses are required. For example, numerous studies make language/gene comparisons on continental and regional scales (Sokal, 1988; Barbujani and Sokal, 1990; Cavalli-Sforza et al., 1992; Smouse and Long, 1992; Chen et al., 1995; Cavalli-Sforza, 1991; Cox, 2003; Hunley and Long, 2005; Diamond and Bellwood, 2003; Nettle and Harriss, 2003), and also at smaller geographical scales (Lansing et al., 2007; Downey et al., 2008; Tumonggor et al., 2014; Cox and Mirazón Lahr, 2006). In addition, edit distances are used in text mining, for example in the extraction of news content (Qiujun, 2010), and in biological applications such as extracting mutation data from the literature (Horn et al., 2004).

The use of evolutionary linguistics in anthropology suggests that further development of quantitative methods are necessary in order to identify new patterns in language families, to identify controversial or undiscovered language families, and to address outstanding problems in human prehistory (Croft, 2008). Research in computational phonology has developed several quantitative metrics for measuring linguistic distances between pairs of words. Algorithms for quantifying the distance between cognate pairs (words with a shared meaning) include measuring phonetic sequence distance based on types of segments (Covington, 1998), or the feature scores of phonemes (Somers, 1998). However, the most common approach is the Levenshtein distance – also called the 'edit distance' – which is defined as the minimum total number of additions, deletions, and substitutions of symbols necessary to transform one word to the other (Levenshtein, 1966). Various mathematical refinements to the Levenshtein distance have been proposed (Wichmann et al., 2010; Petroni and Serva, 2010), including a recent approach that uses empirically determined log-odds (Fine and Florian Jaeger, 2013). The Levenshtein distance is parsimonious and robust and it has been found to correlate with perceptions of dialectical distances (Gooskens and Heeringa, 2004); however, feature-based alignment approaches have been found to be a complementary approach to calculating linguistic distances (Kondrak, 2000).

The ALINE algorithm

ALINE is an automatic phonetic sequence alignment algorithm that determines the similarity of two words (Kondrak, 2000). It uses dynamic programming to calculate the optimal similarity score of candidate alignments. In addition to binary character comparisons, insertions, and deletions, the algorithm uses phonetic information to determine the resulting optimal score. A set of feature weighting parameters and a skip penalty are used to determine individual similarity scores for each phonetic feature in the words being measured; thus, the optimal phonetic sequence alignment depends on the values of the feature weight parameters, and the resulting similarity scores are sensitive to the selection of these values.

Similarity scores can range from $[0, \infty]$ and are strongly influenced by word length. To facilitate integration with biological and evolutionary research we previously defined the ALINE Distance as,

$$\text{ALINE}_{Dist} = 1 - \frac{2s}{s_1 + s_2} \tag{1}$$

where s is the similarity score and $s_{1,2}$ are similarity scores for each word's self-comparison (Downey et al., 2008). This equation results in a finite value $[0, 1]$ that can be easily compared, for example, to common population differentiation statistics such as the fixation index (Fst). For this reason, our package by default returns ALINE_{Dist}, but can optionally return the similarity score. Because similarity scores and ALINE Distances are expected to be sensitive to feature weights, the package parameterizes the values used by the ALINE algorithm so they can be easily modified within the R environment.

We provide **alineR** as an open-source package for the R statistical computing language that facilitates calculation of linguistic distances using the ALINE algorithm. The original ALINE algorithm is provided as an executable (`http://webdocs.cs.ualberta.ca/~kondrak/`) so the default parameters cannot be modified. An open-source python version called PyAline (Huff, 2010) (`http://sourceforge.net/p/pyaline/wiki/Home/`) allows these values to be modified; however, parameter estimation was not a focus. And while the R base command `adist()` and several packages can calculate Levenshtein distances (see `stringdist()` in **stringdist** (Van der Loo, 2014), `levenshteinDist()` in **RecordLinkage** (Borg and Sariyar, 2016), and `stringDist()` in **Biostrings**) (Pagès et al., 2016), to the best of our knowledge, this is the first time ALINE Distances can be calculated directly from an R function. An important feature of **alineR** is to provide a way to estimate the feature-weight parameters and skip penalty. Below we analyze how changing these values affects the resulting alignments and distance measurements. We present a supervised learning methodology that uses manual alignment determinations and a genetic algorithm (GA) to estimate the optimal feature weights for any paired word data. First, we use a simulation analysis and determine that the GA can correctly estimate known feature weights for simulated data. Second, we show that a supervised learning methodology can successfully estimate optimal (unknown) linguistic parameters for a data set consisting of Austronesian word lists from Sumba in Eastern Indonesia. Third, we show that optimizing feature weights improves alignment accuracy using manual determinations as a baseline. Finally, we show how estimating feature-weights and skip penalties affects the resulting distance calculations.

Parameterizing the ALINE algorithm

The ALINE algorithm is a phonetic sequence alignment algorithm based on multivalued features. The program runs quickly because it uses dynamic programming and it is written in C++. Twelve (12) features are considered in calculating the phonetic similarity score: syllabic, voice, lateral, high, manner, long, place, nasal, aspirated, back, retroflex, and round. In addition, there is a skip penalty. Weighting values for each of these parameters are used to choose the optimal string alignments as well as the resulting similarity score. However, in the publicly available version of ALINE the default values were compiled into the original program so they could not be modified. Our **alineR** package includes a modified version of the original ALINE code that interfaces directly with R.

Overview of article

In the next section we provide a how-to guide for calculating ALINE Distances and similarity scores with **alineR**. We present simple instructions for basic alignment operations and for users who want to calculate linguistic distances using this alternative to the Levenshtein distance, the instructions in this section may be sufficient. We then describe the genetic algorithm and illustrate with simple examples how to use it with supervised-learning to optimize ALINE's feature-weight parameters. Next, we show the results from a simulation experiment that validates that the GA can recover a set of 'unknown' feature weighting parameters. We then present a proof-of-concept case study in which we use the GA to determine the optimal feature-weighting values for a sample of languages from

Eastern Indonesia. This includes a description of the training dataset and the work flow necessary to reproduce the analysis. We perform a statistical analysis of the effect the supervised learning and GA optimization process has on the resulting linguistic distance measurements. Finally, we perform a bootstrap analysis to determine whether the results are stable. We close by briefly discussing some possible applications of **alineR** in psycholinguistics and morphology.

Basic alignment and distance calculations with alineR

Calculating ALINE Distances minimally requires two scalar words using UTF-8 file encodings. Word lists should be encoded in the International Phonetic Alphabet to maximize the use of phonetic feature information (Association, 1999). If IPA is not available, the ASCII subset can also be used.

First, load the **alineR** package in the standard way.

```
library('alineR')
```

Consider the following example for calculating the phonetic distance between two related words meaning 'to float' from Borneo: [kərampu] and [ləməntʊŋ] . The most basic use of **alineR** entails passing the two words as the first and second parameters to the `aline()` function. The standardized ALINE_{Dist} is returned.

```
aline('kərampu','ləməntʊŋ')
[1] 0.4864198
```

More typically, word lists will be passed as two vectors that will be analyzed such that ALINE_{Dist} will be calculated for matching elements of each vector. For example, here we pass two vectors representing two cognate pairs ("stone" and "to float") for two related languages. Note that the phonetic difference between u and ʊ does not yield a quantitative difference in the resulting ALINE score.

```
aline(c('batu','kərampuʊ'), c('batʊ','ləməntʊŋ') )
[1] 0.0000000 0.4864198
```

The `aline()` function has several parameters that provide additional functionality.

```
aline( w1, w2, sim = FALSE, m1 = NULL, m2 = NULL, mark = FALSE, alignment = FALSE, ...)
```

The first and second elements, `w1` and `w2`, are required and they are used for passing the two word vectors to be compared as shown above. All additional parameters are optional and provide additional functionality, which will be illustrated in more detail below. These include the following: `sim` = TRUE returns the similarity score rather than the ALINE Distance; `m1` and `m2` allow user-defined feature mappings; setting `mark` = TRUE will mark invalid characters with an "@" character to assist in data checking; setting `alignment` = TRUE will return the IPA word pairs vertically arranged so that the aligned characters can be delimited with vertical bars "`|`". Additionally, feature weighting parameters can also be passed to the internal `raw.alignment()` function using (...). In the next sections we explain some more advanced uses of alineR which require using these optional parameters.

A typical use in historical linguistics is to calculate a matrix of language-distance comparisons among multiple lanaguges. Given the numerous ways that language data can be stored, the need for data consistency, and the difficulty of providing comprehensive error-handling, we do not provide a built-in function for multiple-language comparisons in **alineR**. However, data processing in R is relatively straight-forward. Here we illustrate one possible approach that can be used as a starting point for more complicated analyses. This example processes three word lists that each include three glosses. The results are combined into a distance matrix composed of average ALINE Distance scores.

```
# multiple language comparisons
word.lists <- rbind(c('baqa', NA, 'anax'), c('haqa', 'dodo', 'anar'),
                c('abut', 'suli', 'oan'))
glosses <- colnames(word.lists) <- c('akar','alir_me','anak')
languages <- rownames(word.lists) <- c('language.1', 'language.2', 'language.3')
word.lists
```

```
               akar     alir_me   anak
    language.1   "baqa"   NA        "anax"
    language.2   "haqa"   "dodo"    "anar"
    language.3   "abut"   "suli"    "oan"
```

```
# dim empty matrices: a (ALINE scores), and n (a counter)
n <- matrix(0, nrow = length(languages), ncol = length(languages),
        dimnames = list(languages, languages))
a <- n

# nested loops for calculating the mean ALINE Distances for multiple languages and glosses
for(i in 1:length(glosses)){ # loop glosses
        for(j in 1:length(languages)){ # outer language loop
                for(k in 1:length(languages)){ # inner language loop
                        if(j >= k){
                                x <- word.lists[j, i] # first word to compare
                                y <- word.lists[k, i] # second word to compare
                                if( !is.na(x) & !is.na(y) ) { # skip if missing data
                                        a[j, k] <- a[j, k] + aline(x, y) # ALINE Distance
                                        n[j, k] <- n[j, k] + 1 # increment counter
                                }
                        }
                }
        }
}

as.dist(a / n) # distance matrix composed of mean ALINE Distances
```

```
             language.1   language.2
 language.2   0.3500000
 language.3   0.3869697    0.5479798
```

alineR uses a custom ASCII encoding scheme to identify features. Valid encodings include lowercase letters are ["a"-"z"], and the uppercase modifiers shown in Table 1 are used to indicate features.

Feature	Code
dental	D
palato-alveolar	V
retroflex	X
palatal	P
spirant	S
nasal	N
aspirated	A
long	H
front	F
central	C

Table 1: ALINE Features

The full list of IPA::feature mappings, including ASCII values, is stored in a data frame included in the package. It can be seen using the `show.map()` function. For example, row 2 indicates that Latin Capital B with the Unicode value of 66 will be encoded as a spirant 'b', 'bS' with the two ASCII values, 98 and 83.

```
show.map()
        IPA Aline U.Val     A.Val
1                    32
2         B    bS    66     98 83
3         O    oF    79    111 70
4         a     a    97        97
5         b     b    98        98
...
102           N  8319        78
```

The `encode.ALINE()` function can be used to see the ASCII character encoding of an IPA character string or vector.

```
encode.ALINE('diŋŋira', 'dinnira')
```

```
     diŋŋira     dinnira
 "digNgNira"   "dinnira"
```

It is not required for basic usage to to manually encode words because this function is called internally. However, if an unknown IPA character is used in an alignment, under the default behavior a warning is issued, the character is dropped, and the remaining characters are aligned as usual. When this occurs, the `encode.ALINE()` command may be used with `mark = TRUE` to substitute all unknown characters with the '@' symbol. In the following example, "ɤ" is an unknown feature type.

```
encode.ALINE(c('ũmlatθ', 'ɤmlatθ'), mark = TRUE)
Invalid character: ɤin ɤmlatθ
```

```
     ũmlatθ       ɤmlatθ
 "uNmlattS"    "@mlattS"
```

In such cases, it is possible to eliminate the system warnings by providing these characters with feature mappings. In addition, it is possible to over-ride existing mappings given by `show.map()`. Both can be done using the `m1` and `m2` parameters of `aline()`. For example, the following substitutes all instances of "ɤ" with "o".

```
aline(w1 = c('ũmlatθ','dinnira'), w2 = c('ɤmlatθ', 'diŋŋira'), m1 = 'ɤ', m2 = 'o')
[1] 0.07647059 0.10810811
```

By default, `aline()` returns a vector of ALINE distances indexed by the position in the input vectors. However, additional information about the alignments can also be returned, including the optimal alignment and the similarity score, as shown here.

```
aline('watu', 'dat', alignment = TRUE, sim = TRUE)
                          pair1
w1                         watu
w2                          dat
scores                       50
a1     |  -   w   a   t   |  u
a2     |  d   -   a   t   |  -
```

In this example, setting the optional parameter `alignment = TRUE` will change the output format to a data frame in which each column represents a word-pair comparison. In this example, only one pair of words is compared. Rows 1 and 2 in this data frame contain word 1 (w1), word 2 (w2). The third element will contain either the ALINE distance (if `sim = FALSE`) or the similarity score (if `sim = TRUE`). Rows 4-5 contain the optimal alignment of word 1 and word 2. If three pairs of words are compared, the data frame would consist of three columns and five rows. We adopted this output format as a convenience so that the alignments in rows 4 and 5 could be easily examined directly in the R command output window.

In determining the optimal alignment, the ALINE algorithm associates feature values, or weights, with particular phonemes based on phonologically similar features. Note that each element of the resulting optimally aligned vector maps corresponding elements within the vertical bars. So when aligning [watu] and [dat] the phonetic similarity of [at] | [at] yielded the highest similarity score, 50. In making these calculations, the algorithm calculates weighted feature values for each pair of features, including skips, to determine the optimal similarity score. A vector of the individual similarity scores for the phoneme pairs can be extracted using the `ALINE.segments()` command:

```
aline('watu', 'dat', sim = TRUE) # returns similarity score for comparison
[1] 50
align <- raw.alignment(c('watu', 'dat'))
cat(align[[3]], align[[4]], sep = '\n')
|  -   w   a   t   |  u
|  d   -   a   t   |  -
s <- ALINE.segments(align)
s
[1]  0  0 15 35
sum(s)
[1] 50
```

One of the key features of **alineR** is the ability to easily change the default feature weight values used to determine the optimal alignment and resulting ALINE distances. Here for example, reducing the weight given to `place` from the default of 40 to 10 reduces the values of non-zero distances.

```
aline(c('batu','kərampuʊ'), c('batʊ','ləməntʊŋ'))
[1] 0.0000000 0.4864198

aline(c('batu','kərampuʊ'), c('batʊ','ləməntʊŋ'), Place = 10)
[1] 0.0000000 0.4567901
```

In examples introduced below, it will be more convenient to store the feature weights in a vector. In these instances, the vector can be passed as an argument to `aline()` by constructing a named list and using `do.call()`:

```
opts <- c(61, 92, 51, 26, 54, 38, 20, 40, 31, 38, 66, 72, 60) # feature weights
names(opts) <- names(formals(raw.alignment)[-1]) # add feature names

args <- c(list(w1 = c('batu', 'kərampuʊ'), w2 = c('batu', 'ləməntʊŋ')), opts)
do.call('aline', args)
[1] 0.0000000 0.5604938
```

Optimal parameter estimation using a genetic algorithm for alineR

The 13 parameters used by ALINE creates a high-dimensional search space, and a grid search would therefore be computationally inefficient. Instead, we employ a genetic algorithm (Back et al., 1997) to determine the optimal feature weights for a given word list. The general approach is to mimic biological evolution and natural selection by using a performance function that iteratively evaluates alignments generated with candidate parameters. We define the performance function as the total number of 'correct' alignments, where correctness is determined manually by a linguist trained in phonetic analysis.

An overview of the algorithm is provided in Figure 1. At startup, the total population size (N) and number of iterations are defined for each run of the genetic algorithm. (1) N vectors are initialized by sampling uniformly between [0, 100] for each of the linguistic parameters. (2) The performance of each vector in the current (initial or iterated) population is evaluated using the performance function. (3) Each parameter vector in the population is ranked according to performance and the top .25N are retained. (4) A new 0.25N vectors are initialized into the current generation. (5) Crossover 1 – the retained vectors (from step 3) are recombined using a crossover procedure to create .25N in the current generation. (6) Crossover 2 – the retained vectors (from step 3) are recombined with the new random vectors (from step 4) using a crossover procedure to create .25N new vectors. (7) The four subpopulations from steps 3-6 are combined. (8) Check if the maximum number of iterations has been reached. If not, the algorithm returns to step 2, but otherwise continues to step 9. (9) Aggregate all parameter vectors and estimate medians and variances from the top ranked .25N. Both crossover procedures involve selecting pairs of vectors, selecting a sequence of adjacent elements within each vector, and exchanging them between vectors. Crossover 1 samples .25N individuals from n_s (with replacement) and pairs them for the crossover procedure. Crossover 2 samples .125N individuals from n_s. Each GA run generates a vector of optimized values for each linguistic feature, and when convergence is achieved the median of each distribution is used to parameterize the ALINE algorithm for the training data. Convergence is determined using visual diagnostic plots. To account for the possibility that a single GA run incompletely explores the parameter space, we run multiple iterations and consolidate the results of all runs. In practice, we minimize the computation time by parallelizing this process, as we will demonstrate below. We note, however, that the GA we provide is only one approach to solving this high-dimensional search problem. There are any number of supervised and unsupervised optimization routines that could potentially be used to estimate the linguistic features available in **alineR**.

Supervised learning procedure using the genetic algorithm

We developed a supervised learning procedure for estimating the optimal feature weights using the GA. It is based on expert alignment determinations that are selected from a list of possible alignments for a given set of word lists. Commands in **alineR** are provided to implement this procedure. First, to create a training dataset we require a set of cognates and we use simulation to sample linguistic parameters from a uniform distribution to determine a list of possible alignments for each cognate pair. The simulation excludes the skip penalty (which creates numerous alternative alignments),

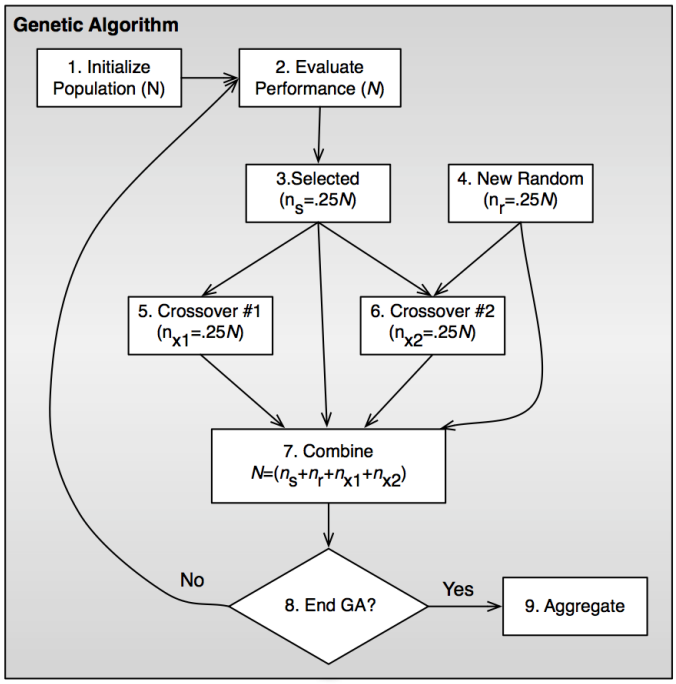

Figure 1: A flowchart showing how the linguistic parameters for ALINE are estimated using a genetic algorithm

and it eliminates cognate pairs with only a single alignment. We then quantify the number of possible alignments for each cognate pair. First, to prepare the cognate lists for the manual alignment determinations we will create some example data.

```
training_wordlists <- rbind(c('hamate', 'kanabu', 'delu'), c('pameti', 'penaʔo', 'telu'))
training_wordlists
       [,1]       [,2]      [,3]
 [1,]  "hamate"   "kanabu"  "delu"
 [2,]  "pameti"   "penaʔo"  "telu"
```

The next step is to perform the simulation process described above to generate a list of unique alignments. In the following series of commands, `generate_training()` will return an R object that will be used internally during the optimization process. However, specifying the optional `table = TRUE` parameters will export a file named 'candidate_alignments.csv' to the working directory that can be used to create a spreadsheet for manually identifying the best alignments.

```
training_set <- generate.training(raw.data = training_wordlists, search.size = 10,
                table = TRUE)
```

Typically, each cognate pair can generate 3-6 unique alignments. But long words contain more phonemes and therefore comparisons between long words will result in a greater number of possible alignments, and in rare cases some cognate pairs can generate 17 or more. When this procedure is used, the list of possible alignments is then provided as a spreadsheet to a trained linguist for manual evaluation (Table 2). Minimally, this requires identifying the 'best' of the provided alignments based on the rules of phonology and knowledge of the languages. The linguist's decisions are then provided to the GA as a vector indicating the numeric index of the 'best' alignment. These are subsequently used in the optimization process. The following configuration initializes 200 populations and instructs the GA to run 50 iterations and return a list of feature weights. These parameters are designed to run quickly for demonstration purposes, and therefore the following optimization examples do not converge. More realistic parameters may run slowly.

```
linguist_determinations <- c(2, 1, 2)
optimal_set <- optimize.features(set = training_set, ranking = linguist_determinations,
        num = 200, step = 50, replication = 3)
optimal_set
[1] 69 41 47 12 40 65 71 43 48 20 54 76 51
```

Alignment No.	Cognate pair 1	Cognate pair 2	Cognate pair 3
	hamate pameti	kanabu penaʔo	delu telu
1	| - h a m - a t - | e | p - a m e - t i | -	| - - k a n a - b u | - | p e - - n a ʔ - - | o	| - d e l u | | t - e l u |
2	| - h a m - a t e | | p - a m e - t i |	| k - a n a - b u | - | p e - n a ʔ - - | o	| d e l u | | t e l u |
3	| h a m - a t - | e | p a m e - t i | -	| k - a n a - b u | | p e - n a ʔ - o |	
4	| h a m - a t e | | p a m e - t i |	| k - a n a b - | u | p e - n a ʔ o | -	
Linguist's Choices	2	1	2

Table 2: Manual alignment determination worksheet

By default, `optimize.features()` returns a vector containing the optimized feature weights, but when `list = TRUE` the returned object can be used with the `features.plot()` function to generate a multi-panel plot showing the results of the optimization process (see Figure 3).

```
optimal_set <- optimize.features(set = training_set, ranking = linguist_determinations,
        num = 200, step = 50, replication = 3, list = TRUE)
features.plot(optimal_set) # not shown, but see Figure 3.
```

As noted above, the GA can be computationally intensive and may require replicates. Therefore it may be necessary to perform multiple runs. The following code illustrates this using a single processing core.

```
reps <- 4
MultiOptResult <- matrix(NA, nrow = reps, ncol = 13)
for (i in 1:reps){
        MultiOptResult[i,] <- optimize.features(set = training_set,
                ranking = linguist_determinations, num = 200, step = 50, replication = 3)
}
round(apply(MultiOptResult, 2, FUN = median)) # optimized feature weights
[1] 53 20 32 22 63 70 52 68 47 71 42 47 70
```

Here we show how to parallelize feature optimization using the **doMC** package (Revolution Analytics and Steve Weston, 2015).

```
# ...replicate using parallelization (OSX/linux)
library(doMC)
registerDoMC(cores = 4)
reps <- 4
MultiOptResult <- foreach(i = 1:reps, .combine = rbind) %dopar% {
        optimize.features(set = training_set, ranking = linguist_determinations, num = 200,
        step = 50,  replication = 3)
}
opts <- round(apply(MultiOptResult, 2, FUN = median)) # optimized feature weights
names(opts) <- names(formals(raw.alignment)[-1])
opts
[1] 61 92 51 26 54 38 20 40 31 38 66 72 60
```

Finally, we illustrate how to pass the results from the optimization process in *opts* to `aline()`, and the effects on the resulting ALINE distances.

```
list1 <- c('batu','kərampuʊ')
list2 <- c('batu','ləməntʊŋ')

aline(list1, list2)
```

```
[1] 0.0000000 0.4864198 # with default feature weights

args <- c(list(w1 = list1, w2 = list2), opts) # construct nested list for do.call()
do.call('aline', args)
[1] 0.0000000 0.5506173 # optimized Aline distances
```

Results

The **alineR** package redistributes a modified version of the original C++ ALINE code, and we maintained the original feature weight values originally used by Kondrak as defaults. Therefore, when the `aline()` function is used without providing optional feature weight parameters, the default performance should be equivalent with Kondrak's original algorithm. In this section we present the results of analyses we conducted to validate and test our GA optimization process. We do this by making an explicit comparison of the default performance of the ALINE algorithm provided in **alineR** (which is equivalent to the original C++ version of ALINE) version) to its performance using optimized parameters.

Genetic algorithm validation

First, we use a simulation experiment to validate that our algorithm converges under ideal conditions. We randomly simulate linguistic parameters and generate alignments for our data set that we designated as 'correct' and then used the GA to estimate these values. The results are shown in Figure 2A. Under these conditions, the GA quickly converges and returns linguistic parameter values that correctly align all the cognate pairs in the training set. Figure 2B also shows results from a proof-of-concept trial of the supervised learning methodology, which we described in the next section.

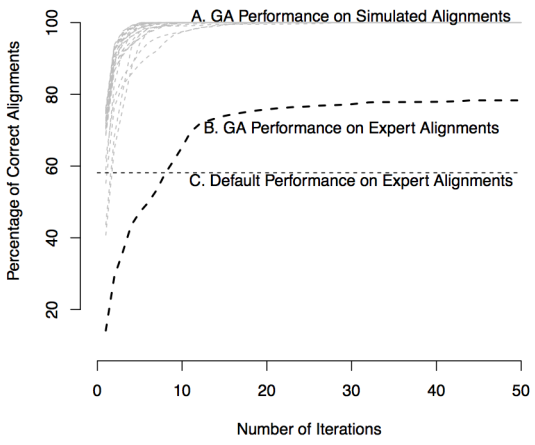

Figure 2: Genetic algorithm convergence plots showing (A) GA convergence using 100 simulated feature parameter sets. In all cases, the GA discovers parameters that could replicate the optimal alignments determined by the simulated parameter values; (B) GA performance estimating unknown parameter values using alignments determined manually by a trained linguist; (C) the alignment performance of the default parameter values when compared to manual determinations. In all cases the GA was run with a population of 100 and for 100 iterations to ensure convergence; only time steps 1-50 are plotted here.

Supervised learning with Indonesian languages

The overall goal of the optimization methodology is to determine whether our procedure can effectively estimate linguistic alignment parameters using real word lists with meaningful phonetic patterns. The reason this is important is because the frequency of linguistic features is expected to vary among languages or groups of languages so the optimal weighting parameters may also vary

(Kondrak, 2003). As a proof-of-concept, we analyze word lists that were collected on the island of Sumba in Eastern Indonesia (Lansing et al., 2007). The data set includes 11,515 unique words organized in a matrix consisting of 56 word lists and 439 cognates. Even though this sample includes a relatively small number of languages, there are approximately 12,000 unique pairwise cognate comparisons. To generate a training data set, we used a stratified framework to sample 203 cognate pairs from the database that are broadly representative of local phonemic complexity and submitted this data set to the linguist for manual alignment determinations. Strata were defined using the proportion of cognate pairs with each number of alignments (1-5) throughout the complete database. A historical linguist (Norquest) determined the 'best' alignment for a sample of cognate pairs by manually examining each word pair and choosing from the list of candidate alignments. These determinations were made qualitatively, based on detailed knowledge from our previous study of the historical relationships among these languages, including sound changes, drift, and inheritance. We then analyzed the performance of the default ALINE parameters for identifying the best possible alignment and compared it to the performance of optimized parameters determined by the genetic algorithm and the supervised learning methodology outlined above. The results of the supervised learning procedure are shown in Figure 3.

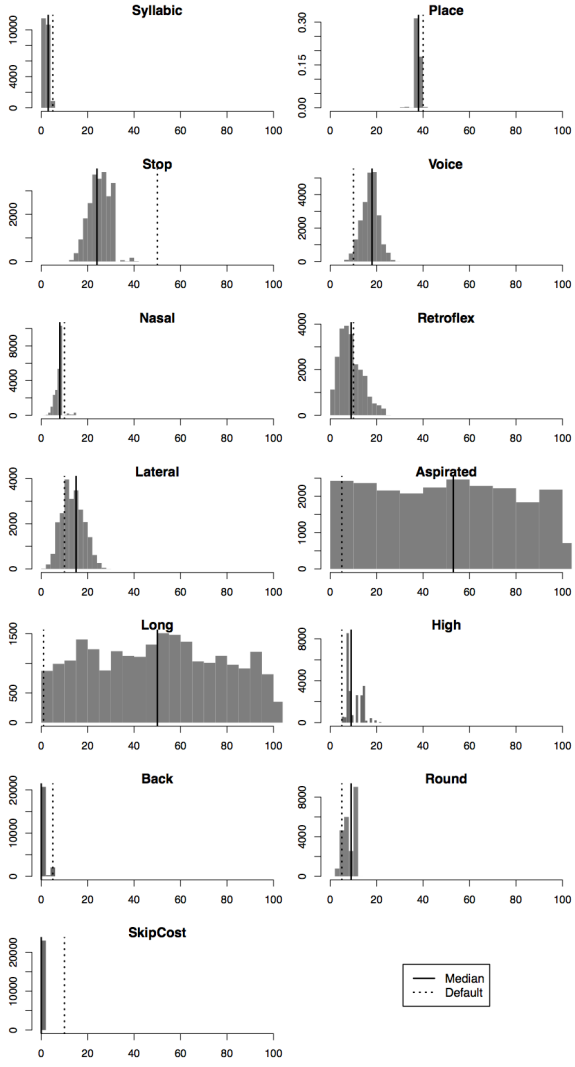

Figure 3: Distribution of optimized feature weights

Alignment accuracy using optimized and default weights

We analyze the ability of the GA to estimate unknown parameters using the alignments manually selected by the linguist. Figure 2 shows that GA performance converges at a lower level when analyzing these alignments. However, suboptimal performance is expected because the manual

coding process necessitates trade-offs between optimal features for aligning any given cognate pair and the optimal values for the entire training set. Nevertheless, we find that the optimized parameters performed better than the default parameters. Since the ALINE algorithm encodes linguistic rules, it is expected to achieve a certain number of correct alignments regardless of parameter weights. We find that this is the case: the median number of correct alignments using simulated random parameters is 19, while the default parameters achieve 118 correct alignments, and the optimized parameters achieve 158. Thus, our results show that in comparison to randomly selected feature weights, the optimal feature weights are approximately 29% more accurate than the defaults.

Statistical analysis of optimized and default weights

Next, we test whether the difference between the parameter estimates from the GA and the default values are statistically significant. The population of optimized candidate parameter values from the GA is used to determine empirical distributions for each linguistic parameter and to calculate the probability for each. The default values, median estimated values, and probabilities are reported in Table 3. We find 9/13 features differed from the default values. We include $\alpha \leq .10$ because of the size of the training sample – a larger data set is expected to return even more significant results. Regardless of sampling, it is expected that not all features would be significant in a given language family because certain features are more important than others for classification tasks (Kondrak, 2003). In two cases (aspirated, long), the estimated parameters differ significantly from the defaults based on the empirical distributions, but in these cases the algorithm is not sensitive to these parameters, most likely because those features are not prominently represented in the training data set.

Parameter	Default	Optimized	Pr(X<>x)
Syllabic	5	3	*** 0.0022
Place	40	38	** 0.0108
Stop	50	26	*** $< 1.0^{-16}$
Voice	10	18	* 0.0506
Nasal	10	8	* 0.0626
Retroflex	10	9	0.6972
Lateral	10	14	0.4710
Aspirated	5	52	* 0.0946
Long	1	52	** 0.0194
High	5	9	*** 0.0004
Back	5	0	0.1580
Round	5	9	0.1762
Skip	10	0	*** $< 1.0^{-16}$

Sig. levels: $\alpha \leq .01$ (***); $\alpha \leq .05$ (**); $\alpha \leq .1$ (*)

Table 3: Summary Statistics for a Population of Optimized Parameters

Cross-validation

To determine whether the resulting optimized parameters can correctly align reserved data, we use leave-one-out cross-validation. We iteratively optimize parameters using 202/203 of the cognate pairs from the training data such that each cognate pair is reserved once. After each iteration, the optimized parameters are used to align the reserved pair and a success is recorded when the automated alignment matches the linguist-determined alignment. In this binary classification routine the default parameters successfully predicted 118/203 (58%) of the manual determinations and the optimized parameters predicted 151/203 (75%).

Statistical analysis of distances generated with optimized and default weights

Finally, to determine whether optimizing parameters affects the linguistic distance measurements, we analyze the distribution of the pairwise differences between the distances calculated by the optimized and default parameters. A Wilcoxon signed rank test for paired samples finds that the median optimized distance is 0.0096 greater than the default median and that this difference is statistically significant ($M_o = 0.1896, M_d = 0.1800, V = 9247, \mathrm{p} < 2.612 \times 10^{-07}$). We also note that there are

instances when the difference between the default and optimized distances is less than 0. This suggests that optimizing linguistic parameters may have nonlinear effects on the resulting distances.

Future research: potential synchronic uses for alineR

Although our work has been used primarily for diachronic applications (the generation of phylogenetic trees), it has potential synchronic applications as well.[1] One potential use involves the computation of what are known as phonological neighbors – words which are similar to a target word but differ from that target by a single phoneme. The total number of words fitting this definition can be described as a phonological neighborhood. A word's neighborhood size can affect the way we understand and use it. When we hear a word, everything that sounds like that word becomes slightly more accessible in memory. It therefore takes listeners longer to determine what a word is when that word has several neighbors (Luce and Pisoni, 1998). Take, for example, the phonological neighborhood of cat, where each word in the neighborhood differs from cat by a single phoneme (either consonant or vowel):

cat: cab, cad, can, cam, cap, cart, cash, catch, caught, coat, cot, court, cut, bat, chat, fat, hat, kit, kite, mat, pat, rat, sat, that

With a large neighborhood such as the one above, it is more difficult to eliminate the words that were *not* said; *cot* or *cap* for example, rather than *cat*. A word like *green*, which has fewer neighbors than *cat*, is less confusable, and thus requires less work to identify:

green: grain, gran, greed, grief, grease, Greece, Greek, greet, grin, grown, groan

While phonological neighbors can be viewed in terms of differing segments, a finer-grained analysis may also break these segments down into phonological features. For example, *cad* differs from *cat* by one feature: voicing in the third segment. On the other hand, *cab* differs from *cat* in both voicing and place of articulation of the final segment. Likewise, *grain* differs from *green* by one feature in the vowel (height), but *groan* differs from *green* by three features (height, backness, and rounding).

As a feature-driven algorithm, ALINE could allow the members of a phonological neighborhood to be mapped in a quantitative way based on phonological distance. The resulting neighborhood maps can then be used in psycholinguistic experiments, testing the hypothesis that phonological distance between neighbors at the feature level correlates with both access times and production values.

Another way in which ALINE could be used synchronically involves certain kinds of speech errors and neologisms. Strictly phonological errors include metathesis, sound-exchange errors, and spoonerisms. These are all instances in which the linear order of a pair of phonemes is reversed, as in the examples below:

Metathesis: pus pocket > pos pucket

Sound-exchange error: night life > knife light

Spoonerism: light a fire > fight a liar

Other forms of phonological errors include additions and deletions:

Addition: optimal > moptimal

Deletion: unanimity > unamity

Still other forms include anticipation and perseveration, where a phoneme (or sequence of phonemes) in one word influences the articulation of another word in the same phrase:

Anticipation: reading list > leading list

Perseveration: black boxes > black bloxes

With ALINE's feature-driven alignment capabilities, and given an adequate corpus of errors, generalizations could be quickly drawn about both the position of errors within the word as well as the frequency with which various phonemes participate in unitary instances as well as in pairs.

A final example, which occurs in both speech errors and intentional neologisms, is what are known as blends:

channel x tunnel > chunnel

breakfast x lunch > brunch

The ALINE algorithm could foster some unique insights in these cases. In unmarked blends such as *chunnel* and *brunch*, the initial part of the first donor word (the onset of the initial syllable in these cases) replaces the same in the second donor word. On the other hand, in the unintentional blend perple, an entire syllable is copied from the first donor word:

person x people > perple

[1] We would like to thank an anonymous reviewer for suggesting the inclusion of this section.

At first it is not clear why this example differs from the ones given above. However, one fact which stands out is that the initial consonant [p] is common to both of the input forms as well as the output form (a fact which would be reflected in the output score of an ALINE comparison). Since both donor words have the same initial consonant, if the same rule was applied here which operated in the *chunnel* and *brunch* examples, the hypothetical blend would be indistinguishable from the second donor word:

person x people > people

A more ambiguous case is that of *motel*, where the blend contains two phonemes (a vowel and a consonant) which are shared between the donor words:

motor x hotel > motel

An ALINE analysis would indicate this ambiguity via the output score which correlates with the shared material overlapping between the two donor words.

ALINE could therefore be used to quickly analyze and categorize different kinds of blends (particularly in the case of a large corpus), with an eye toward answering questions such as what determines the size of the constituents of a blend (how many segments or syllables are copied from the donor words to the blend), what determines the type of constituents in a blend (i.e. are they syllable onsets, full syllables, and so on), and what role segmental overlap plays in blend formation. While an analysis of these phenomena lies outside the scope of this paper, given the appropriate type and size of corpi, we consider the examples above to be fruitful prospects for future research in phonology and psycholinguistics.

Conclusion

In conclusion, **alineR** is a new R package for calculating and optimizing feature-weighted linguistic distances. It is now available on CRAN. Supplemental materials accompanying this paper include an R script ('downey.R') for the commands and analyses included above. The linguistic data from Sumba used for training the GA is in the `train` object in 'downey.Rdata'. We suggest that optimized feature-weighted linguistic distances are an important complement to other linguistic distances such as the edit or Levenshtein distance. In addition to calculating the ALINE distance and returning relevant alignment information and similarity scores, the package provides a genetic algorithm that can be used to estimate linguistic parameters using supervised learning. It may be particularly useful for bioinformatic applications in anthropology and historical linguistics or in comparisons with well-resolved quantitative distance measurements (e.g., Fst). As such, it has the potential to help advance our understanding of the evolutionary relationships between languages and genetics. Not only can this help uncover historical demographic patterns, but coevolutionary analyses using the ALINE Distance may provide insight into general processes of biological and cultural evolution.

Acknowledgment

The authors wish to express their appreciation to Murray Cox for his input on early drafts of this paper and for his guidance submitting alineR to CRAN. Steve Lansing provided access to the Sumbanese language data set, which he collected during his previous projects. We also thank Tanmoy Bhattacharya, Aakrosh Ratan, and the other participants of the Santa Fe Institute Workshop, "Co-evolution of Social Structure and Communication in Different Genotypes," May 26 - 29, 2015 for critical and constructive input.

Funding

This work was supported by the National Science Foundation [Grant number SBS-1030031 to P.N. and S.D.], and the University of Maryland.

Bibliography

I. P. Association. *Handbook of the International Phonetic Association: A guide to the use of the International Phonetic Alphabet*. Cambridge University Press, 1999. [p3]

T. Back, D. B. Fogel, and Z. Michalewicz. *Handbook of evolutionary computation*. IOP Publishing Ltd., 1997. [p6]

G. Barbujani and R. R. Sokal. Zones of sharp genetic change in europe are also linguistic boundaries. *Proceedings of the National Academy of Sciences*, 87(5):1816–1819, 1990. [p1]

A. Borg and M. Sariyar. *RecordLinkage: Record Linkage in R*, 2016. URL https://CRAN.R-project.org/package=RecordLinkage. R package version 0.4-10. [p2]

L. L. Cavalli-Sforza. Genes, peoples and languages. *Scientific American*, 265(5):104–110, 1991. [p1]

L. L. Cavalli-Sforza, E. Minch, and J. Mountain. Coevolution of genes and languages revisited. *Proceedings of the National Academy of Sciences*, 89(12):5620–5624, 1992. [p1]

J. Chen, R. R. Sokal, and M. Ruhlen. Worldwide analysis of genetic and linguistic relationships of human populations. *Human Biology*, pages 595–612, 1995. [p1]

M. A. Covington. Alignment of multiple languages for historical comparison. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics-Volume 1*, pages 275–279. Association for Computational Linguistics, 1998. [p1]

M. P. Cox. *Genetic patterning at Austronesian contact zones*. PhD thesis, University of Otago, 2003. [p1]

M. P. Cox and M. Mirazón Lahr. Y-chromosome diversity is inversely associated with language affiliation in paired austronesian-and papuan-speaking communities from solomon islands. *American Journal of Human Biology*, 18(1):35–50, 2006. [p1]

W. Croft. Evolutionary linguistics. *Annual Review of Anthropology*, 37(1):219, 2008. [p1]

J. Diamond and P. Bellwood. Farmers and their languages: The first expansions. *Science*, 300(5619): 597–603, 2003. ISSN 0036-8075. doi: 10.1126/science.1078208. URL http://science.sciencemag.org/content/300/5619/597. [p1]

S. S. Downey, B. Hallmark, M. P. Cox, P. Norquest, and J. S. Lansing. Computational feature-sensitive reconstruction of language relationships: Developing the aline distance for comparative historical linguistic reconstruction. *Journal of Quantitative Linguistics*, 15(4):340–369, 2008. [p1, 2]

A. B. Fine and T. Florian Jaeger. Evidence for implicit learning in syntactic comprehension. *Cognitive Science*, 37(3):578–591, 2013. [p1]

C. Gooskens and W. Heeringa. Perceptive evaluation of levenshtein dialect distance measurements using norwegian dialect data. *Language variation and change*, 16:189–208, 2004. [p1]

F. Horn, A. L. Lau, and F. E. Cohen. Automated extraction of mutation data from the literature: application of mutext to g protein-coupled receptors and nuclear hormone receptors. *Bioinformatics*, 20(4):557–568, 2004. [p1]

P. A. Huff. Pyaline: Automatically growing language family trees using the aline distance. Master's thesis, Brigham Young University-Provo, 2010. [p2]

K. Hunley and J. C. Long. Gene flow across linguistic boundaries in native north american populations. *Proceedings of the National Academy of Sciences of the United States of America*, 102(5):1312–1317, 2005. [p1]

G. Kondrak. A new algorithm for the alignment of phonetic sequences. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, pages 288–295. Association for Computational Linguistics, 2000. [p1, 2]

G. Kondrak. Phonetic alignment and similarity. *Computers and the Humanities*, 37(3):273–291, 2003. [p10, 11]

J. S. Lansing, M. P. Cox, S. S. Downey, B. M. Gabler, B. Hallmark, T. M. Karafet, P. Norquest, J. W. Schoenfelder, H. Sudoyo, J. C. Watkins, et al. Coevolution of languages and genes on the island of sumba, eastern indonesia. *Proceedings of the National Academy of Sciences*, 104(41):16022–16026, 2007. [p1, 10]

V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966. [p1]

P. A. Luce and D. B. Pisoni. Recognizing spoken words: The neighborhood activation model. *Ear and hearing*, 19(1):1, 1998. [p12]

D. Nettle and L. Harriss. Genetic and linguistic affinities between human populations in eurasia and west africa. *Human Biology*, pages 331–344, 2003. [p1]

M. Pagel. *Wired for culture: origins of the human social mind*. WW Norton & Company, 2012. [p1]

H. Pagès, P. Aboyoun, R. Gentleman, and S. DebRoy. *Biostrings: String objects representing biological sequences, and matching algorithms*, 2016. R package version 2.40.2. [p2]

F. Petroni and M. Serva. Measures of lexical distance between languages. *Physica A: Statistical Mechanics and its Applications*, 389(11):2280–2283, 2010. [p1]

L. Qiujun. Extraction of news content for text mining based on edit distance. *Journal of Computational Information Systems*, 6(11):3761–3777, 2010. [p1]

Revolution Analytics and Steve Weston. *doMC: Foreach Parallel Adaptor for 'parallel'*, 2015. URL https://CRAN.R-project.org/package=doMC. R package version 1.3.4. [p8]

P. E. Smouse and J. C. Long. Matrix correlation analysis in anthropology and genetics. *American Journal of Physical Anthropology*, 35(S15):187–213, 1992. [p1]

R. R. Sokal. Genetic, geographic, and linguistic distances in europe. *Proceedings of the National Academy of Sciences*, 85(5):1722–1726, 1988. [p1]

H. L. Somers. Similarity metrics for aligning children's articulation data. In *Proceedings of the 17th international conference on Computational linguistics-Volume 2*, pages 1227–1232. Association for Computational Linguistics, 1998. [p1]

M. K. Tumonggor, T. M. Karafet, S. Downey, J. S. Lansing, P. Norquest, H. Sudoyo, M. F. Hammer, and M. P. Cox. Isolation, contact and social behavior shaped genetic diversity in west timor. *Journal of human genetics*, 59(9):494–503, 2014. [p1]

M. P. Van der Loo. The stringdist package for approximate string matching. *The R*, 2014. [p2]

S. Wichmann, E. W. Holman, D. Bakker, and C. H. Brown. Evaluating linguistic distance measures. *Physica A: Statistical Mechanics and its Applications*, 389(17):3632–3639, 2010. [p1]

Sean S. Downey
Department of Anthropology, University of Maryland
4302 Chapel Lane, College Park, MD 20742
United States
sdowney2@umd.edu

Guowei Sun
Department of Mathematics, University of Maryland
176 Campus Drive, College Park, MD 20742
United States
gwsun@umd.edu

Peter Norquest
Department of Anthropology, University of Arizona
1009 E. South Campus Drive, Tucson, AZ 85721
United States
norquesp@email.arizona.edu