dialectR: Doing Dialectometry in R

Ryan Soh-Eun Shim

Abstract

dialectR is an open-source R software package that provides access to methods in dialectometry. While other open-source software exist for such analyses (e.g. RuG/L04 and Gabmap), dialectR aims to facilitate the possibility of extending these methods with the statistical packages available in the R ecosystem. The core of the computations are implemented in C++, thus ensuring reasonably fast results. Visual presentations common in dialectometry are also implemented (e.g. maps of dialect continuua based on multidimensional scaling and hierarchical clustering), which allow for modifiable graphics. Here we present the package by replicating an MDS-based dialect continuum plot for Dutch, using data from the Goeman-Taeldeman-Van Reenen-project. We also showcase newer functionalities not yet available in other dialectometry software, by replicating a plot of the acoustic vowel space with an acoustic distance based on mel-scale frequency cepstral coefficients.

Introduction

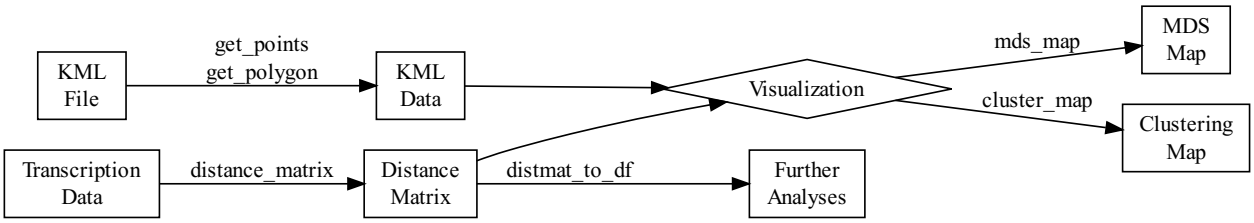

Figure 1: Minimal dialectR workflow

Although in recent years the application of quantitative and computational methods to linguistic data has become increasingly common, in dialectology such methods have generally relied on specialized software that vary in accessibility, and generally do not lend well to easy extension (Nerbonne et al., 2011: 87). dialectR seeks to fill this gap by providing an implementation of dialectometric methods in R, which previously have required software such as RuG/L04 and Gabmap for its analysis (Nerbonne et al., 2011; Leinonen et al., 2016). In this paper, we demonstrate the functionalities of this package by replicating some common dialectometric methods with Dutch data from the Goeman-Taeldeman-Van Reenen-project (Taeldeman and Goeman, 1996).

Edit Distance in Dialectometry

Distance computations based on modified forms of edit distance constitute an important part of dialectometry (Heeringa, 2004), and stands as a first step before further analyses can be performed. We here provide a brief introduction to edit distance, along with modifications to it that have become of use in dialectometry, before delving into how it may be computed with dialectR.

Edit distance, alternatively known also as Levenshtein distance, is a method of comparing two sequences with the end goal of deriving a "distance" between the two. This distance is based on the number of insertions, deletions, and substitutions required for one string to transform into the other. As an example, consider how the string "koguma," the word for sweet potato in Korean, may be transformed into "kokoimo," the origin of the Korean term from the Tsushiba dialect in Japan, with the three operations:

k	o	-	g	u	m	a
k	o	k	o	i	m	o
		1	1	1		1

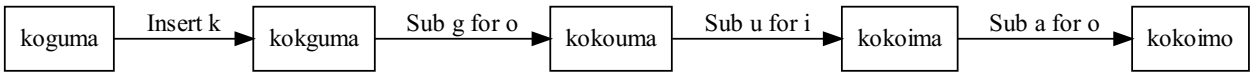

Figure 2: Edit distance between ”koguma” and ”kokoimo”

We see in Figure 2 that with one insertion and three substitutions, we are able to transform the string in Korean into the string in the Japanese Tsushiba dialect. While this method may seem crude for not taking into account the degree of similarity between phonemes (e.g. ‘d’ should ideally be closer to ‘t’ than to, say, ‘r’), empirical studies have shown that aggregate results correlate well with human perceptions of dialect differences (Gooskens and Heeringa, 2004), and thus stands as a good, albeit approximate, probe. A simple modification in dialectometry upon this method is to set a constraint that forbids the alignment of vowels and consonants by increasing the cost of their alignment, which is reported to yield better results (Heeringa, 2004; Wieling et al., 2009: 28-29), and is provided in this package. The distance functions can be called as below, where the results are different because vc_leven (standing for VC-sensitive levenshtein distance) penalizes the alignment of the word final “r” and “a”:

```
library(dialectR) # load dialectR
leven("vir", "via")
```

```
[1] 1
```

```
# plain edit distance
vc_leven("vir", "via")
```

```
[1] 2
```

```
# VC-sensitive edit distance
```

For aggregate analyses however, there will inevitably be cases where multiple responses are given for a given dialect location, or the difference in string lengths might distort the data so as to give inaccurate results. We describe options to account for these conditions below.

Normalization by Alignment Length

In comparing two sequences, longer sequences possess a much higher chance containing more differences than shorter sequences. If used directly, this would bias the results by causing varieties with longer sequences to appear more different. We thus follow Wilbert Heeringa in normalizing the distance by dividing the alignment length (Heeringa, 2004; Heeringa et al., 2006), which can be done as below:

```
leven("vir", "via", alignment_normalization = TRUE)
```

```
[1] 0.3333333
```

```
# edit distance with normalization
```

Bilbao Distance for Multiple Responses

For a treatment of multiple responses in dialect data, Aurrekoetxea et al. specifies the following desiderata (Aurrekoetxea et al., 2020):

1. The frequency of a response should not matter, thus the difference between {a, a, b, c, c, c} and {a, b, d, d} should be the same as between {a, b, c} and {a, b, d}.
2. The distance between identical sets should be 0.
3. The procedure should be able to accommodate different cost functions, such as edit distance and inverse frequency weighting
4. It should be based on the measure of individual responses.

Aurrekoetxea et al. proposes Bilbao distance as a solution that meets the above criteria. The formula for the procedure is as below:

$$D_B(A,B) = \frac{\sum_{i=1}^{|A|} \min_{b_j \in |B|} d(a_i b_j) + \sum_{j=1}^{|B|} \min_{a_j \in |A|} d(a_i b_j)}{|A| + |B|}$$

Where, in plain words, every element in a given set A is computed for a minimum distance with the elements of set B, and the same is done for all the elements in set B against those of set A. We illustrate this with an example: suppose we have {a,b} as set A and {b,c} as set B. Using edit distance as the distance metric, the minimum distance for a in set A as compared against set B is 1, and for b in set A it is 0 (since an identical element also exists in set B). Moving to set B, we similarly have a minimum distance of 0 for b and 1 for c when compared against set A. We add all the distances up, and divide the sum by the total number of elements, which yields:

$$D_B(\{a,b\},\{b,c\}) = \frac{1+0+0+1}{2+2} = \frac{1}{2}$$

This procedure is implemented in the package and may be used as below, where the delimiter of the multiple responses should be specified in delim:

```
leven("a/b", "b/c", delim = "/")
```

```
[1] 0.5
```

```
# multiple responses for plain edit dist.
vc_leven("vir/via", "via/jura", delim = "/")
```

```
[1] 1.25
```

```
# for VC-sensitive
```

Visual Presentation of Variation

With the modified edit distance function above, we demonstrate how an aggregate analysis of dialect differences may be carried out with the package. We use data from the Goeman-Taeldeman-Van Reenen-project as an example (Taeldeman and Goeman, 1996), since many analyses in dialectometry have been done on this (Wieling et al., 2007; Wieling and Nerbonne, 2011), and may serve as a backdrop against which to compare the results of this package. The data from the project presented here is retrieved from Gabmap, and may be loaded as below:

```
data(Dutch) # load GTRP data
```

Note however that the data must be in a format where the rows represent the data collections sites and the columns the data item types, which is the required format for Gabmap as well. We show an example of this below:

```
            Item Name 1 Item Name 2 Item Name 3
Site Name 1 "response"  "response"  "response"
Site Name 2 "response"  "response"  "response"
Site Name 3 "response"  "response"  "response"
```

A distance matrix as computed with the modified edit distance discussed above may be obtained as below, where an additional option to choose either a VC-sensitive edit distance or a plain one is possible:

```
distDutch <- distance_matrix(Dutch, "vc_leven", alignment_normalization = TRUE)
# distance matrix, may take a while for large data (around 15 minutes for GTRP)

distDutch[1:3,1:3]
```

```
            Aalsmeer NH Aalst BeLb Aalst BeOv
Aalsmeer NH      0.0000   196.8762   246.7484
Aalst BeLb     196.8762     0.0000   205.2460
Aalst BeOv     246.7484   205.2460     0.0000
```

Multidimensional Scaling

Having obtained the distance matrix, we are now in a position to reproduce a map of the Dutch dialect continuum with multidimensional scaling (Nerbonne et al., 1999). In essence, this procedure performs classical multi-dimensional scaling on the distance matrix to reduce the dimensions to three; the three dimensions are then mapped to RGB values respectively, mixed together, and then projected onto a map. We first show how to load KML data with the get_points and get_polygons functions:

```
dutch_points <- get_points(system.file("extdata", "DutchKML.kml", package="dialectR"))
# KML points data
dutch_polygons <- get_polygons(system.file("extdata", "DutchKML.kml", package="dialectR"))
# KML polygons data
```

We may then visualize the results (Figure 3):

```
mds_map(distDutch, dutch_points, dutch_polygons) +
  ggplot2::theme_bw()
```

Clustering

We also provide functions for hierarchical clustering which, as it is based on R's hclust method, allows for a range of parameters, most importantly including the choice of the agglomeration method. For instance, it is possible to obtain maps computed by Ward's method and average linkage, as shown below (Figure 4):

```
cluster_map(distDutch,
            method = "ward.D2", # Ward's method
            kml_points = dutch_points,
            kml_polygon = dutch_polygons,
            cluster_num = 5)

cluster_map(distDutch,
            method = "average", # average linkage
            kml_points = dutch_points,
            kml_polygon = dutch_polygons,
            cluster_num = 5)
```

We can verify the validity of the cluster number with multidimensional scaling (Figure 5), a procedure which Gabmap also provides (Leinonen et al., 2016: 79):

```
mdsDutch <- cmdscale(distDutch, k=3)
# multidimensional scaling
clusterDutch <- get_clusters(distDutch, 5, "ward.D2")
plot(mdsDutch[,1],mdsDutch[,2], col = clusterDutch$grouping)
```

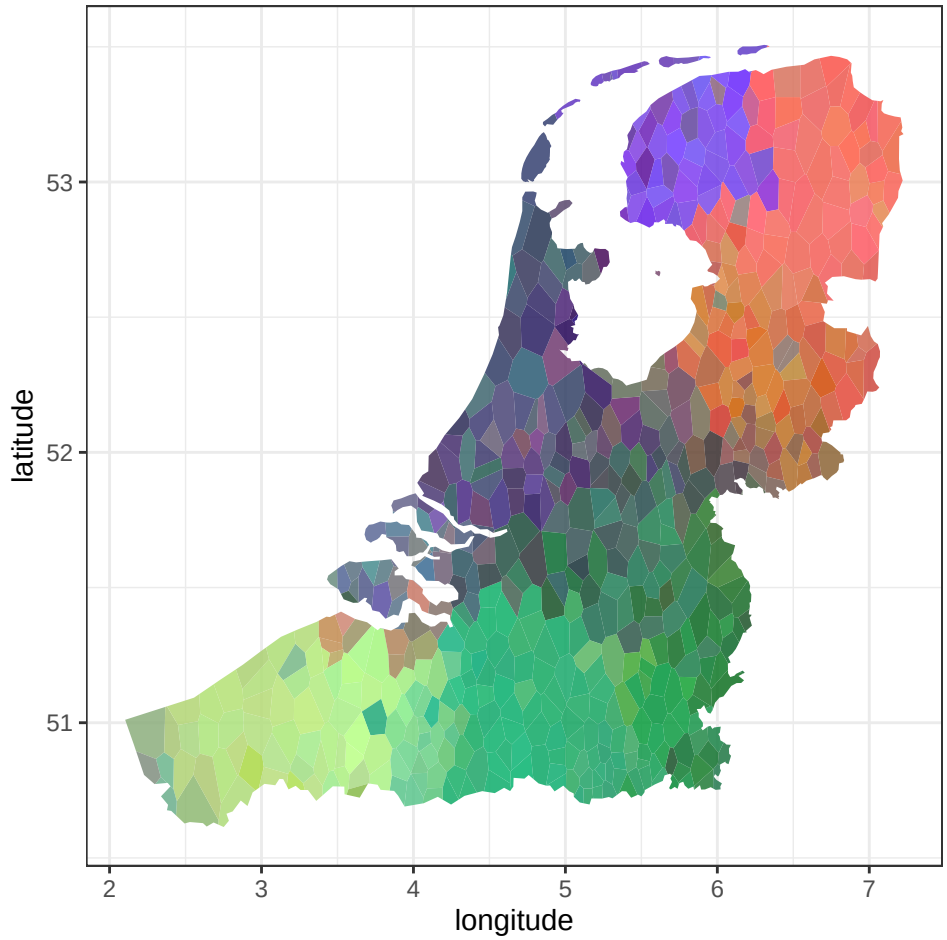

Figure 3: Map of Dutch variation based on multidimensional scaling

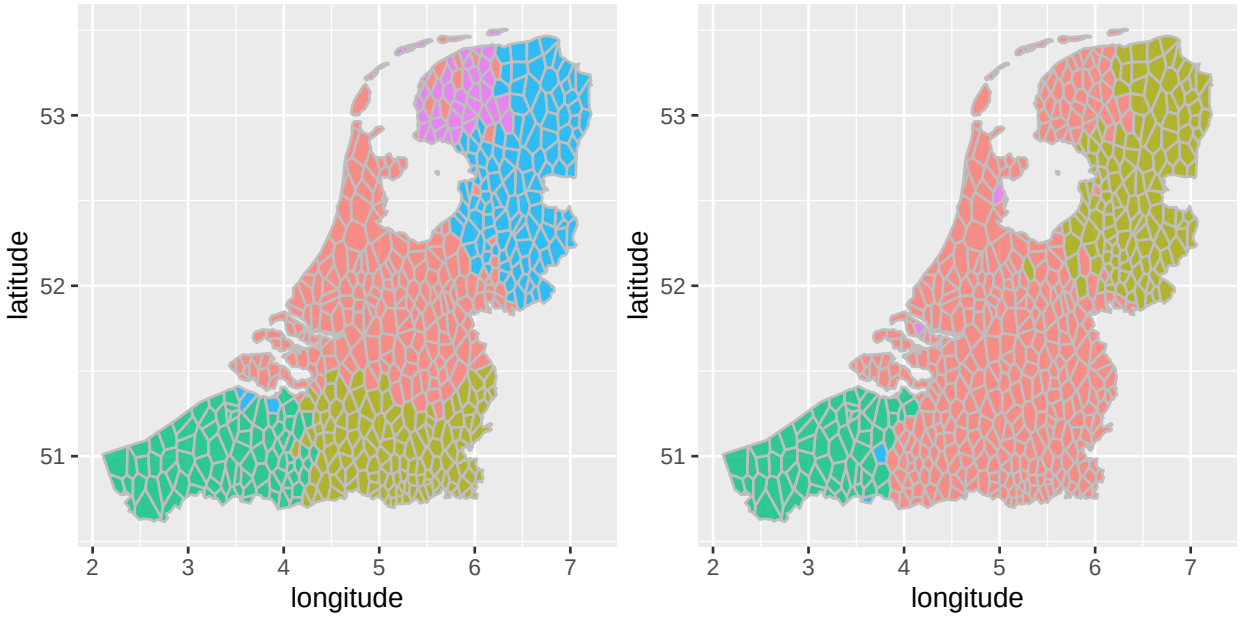

Figure 4: Left: Ward's method; Right: average linkage

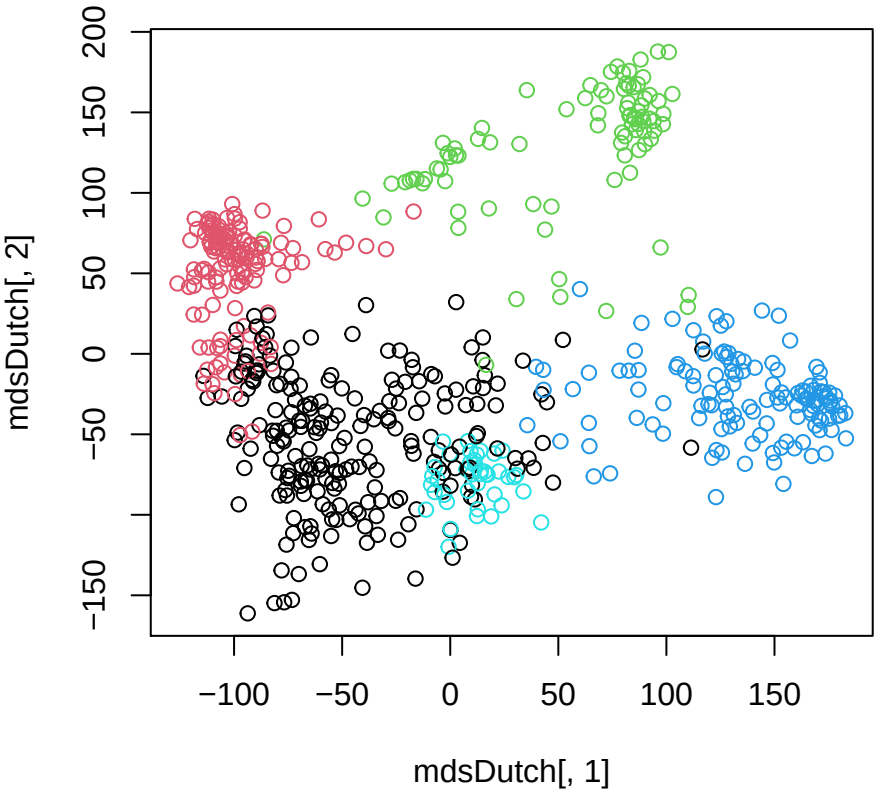

Figure 5: Multidimensional scaling to verify cluster validity

Extending dialectR

In this section, we give an example of how users could extend dialectR for analyses that are not implemented. We illustrate this by creating a reference point map of a dialect site in Friesland, which in the analyses above have consistently been marked out as being a dialect area of its own. We do this first by converting the distance matrix to a data frame with the function distmat_to_df:

```
dfDutch <- distmat_to_df(distDutch)
# transform from distance matrix to dataframe
dfDutch[1:5,]
```

```
          row        col     dist
1 Aalsmeer NH Aalst BeLb 196.8762
2 Aalsmeer NH Aalst BeOv 246.7484
3  Aalst BeLb Aalst BeOv 205.2460
4 Aalsmeer NH  Aalten Gl 290.6292
5  Aalst BeLb  Aalten Gl 285.6391
```

With the data as a data frame, we are now in a position to perform the following steps:

1. Filter out the distance data between Nij Beets (a dialect site in Friesland) and the rest of the sites
2. Merge the filtered data with the KML point data we retrieved above, in order to plot the distances back onto a map
3. Plot the data with ggplot2

This is illustrated as follows:
Step 1. Filter out data:

```
library(dplyr)
library(ggplot2)

ref_point <- dfDutch %>%
  filter(row == "Nij Beets Fr")
# to filter out the data that pertains to Nij Beets
```

Step 2. Merge with KML points data:

```
ref_point_with_coors <- merge(ref_point,
                              dutch_points,
                              by.x = "col",
                              by.y = "name")
# merge with KML points data for plotting
```

Step 3. Plot as reference point map (Figure 6):

```
  ggplot(data = ref_point_with_coors,
                  aes(x = longitude,
                      y = latitude)) + # base plot
    geom_point(aes(color = dist)) + # points
    geom_polygon(data = dutch_polygons,
                         aes(x = longitude,
                             y = latitude,
                             group = name),
                         color = "black",
                         fill = NA) + # polygon
    scale_color_binned(n.breaks=10,
                       low = "red",
                       high = "green") # color
```

With this minimal reference point map, we see clearly that the majority of the sites least different from Nij Beets are in Friesland, and that the similarity stops abruptly (as opposed to a gradual change).

MFCC-based Acoustic Distance

While dialectR has followed Gabmap in implementing many of the functionalities, we have additionally made available an acoustic distance published in a recent paper (Bartelds et al., 2020), which makes it possible to perform similar analyses as those based on edit distance, without having to go through the effort of transcription. While we refer the reader to the paper for the technical details, we briefly remark that the distance essentially performs dynamic time warping upon audio files, which are represented as Mel-frequency cepstral coefficients (henceforth MFCC). We illustrate its use by calculating the distance between vowels in typical vowel chart, in an attempt to reproduce in this manner the acoustic vowel space. This experiment has been succesfully attempted both in the paper by Bartelds et al. and in approaches based on more sensitive modifications to edit distance (Wieling et al., 2011; Wieling et al., 2012), and thus again may serve as a good backdrop of comparison.

We make use of vowels as pronounced by Peter Ladefoged for the data (Ladefoged, 2005), which we refer the reader to download directly from the website if she wishes to reproduce the results.[1] Assuming all the vowels were in a single folder however, it is possible to obtain a distance matrix with a nested loop, as shown below:

```
vowel_dist <- sapply(1:12, function(x){
  sapply(1:12, function(y){
  acoustic_distance(list.files("C:/Users/USER/Downloads/ipa_vowels", full.names = TRUE)[x],
```

[1] http://www.phonetics.ucla.edu/course/chapter1/vowels.html

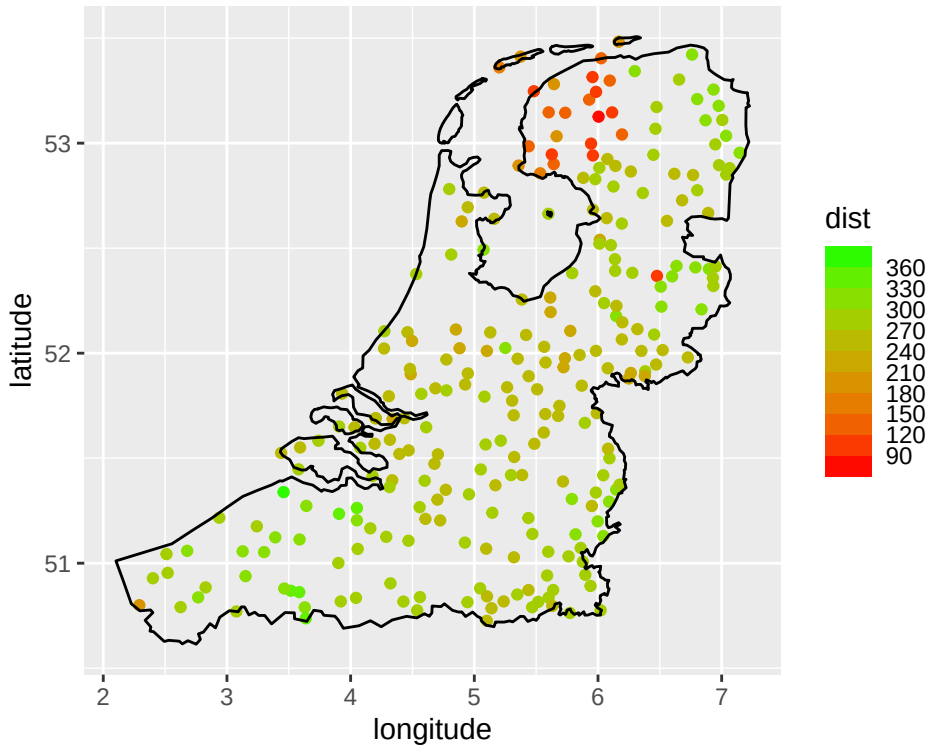

Figure 6: Reference point map of Nij Beets

```
                    list.files("C:/Users/USER/Downloads/ipa_vowels", full.names = TRUE)[y])
  })
})

vowel_dist[2:4,2:4]

          a         e         i
a  0.00000 10.494420 10.295286
e 10.49442  0.000000  9.413114
i 10.29529  9.413114  0.000000
```

With the distance matrix, we perform multidimensional scaling to reduce the dimensions, and plot the first two as the x and y axes (Figure 7). The plot appears to be consistent with a formant-based acoustic vowel space, as confirmed by the results in the original paper (Bartelds et al., 2020).

Conclusion

dialectR sets out to fill a specific gap in dialectometric software, where with the growing number of linguists with computational training, it is reasonable to assume a level of technical competence that was previously not possible. As such, this package can be seen as a preliminary answer to some of the shortcomings of Gabmap, among which " . . . the most important . . . would involve making it easier for others to contribute modules, i.e. adopting an open-source development mode. Once it becomes easier for others to contribute, then the scientific imagination is the limiting factor" (Nerbonne et al., 2011: 87). As dialectR remains in active development at this stage however, many possible improvements do also remain. An important one among these is the addition of the edit distance based on pointwise mutual information (Wieling et al., 2012; Wieling et al., 2011), which has been reported to generate significantly better alignments on the one hand, and being able to remedy differences in transcription on the other. The point remains however, that with the open-source nature of the project, such additions should become all the more viable.

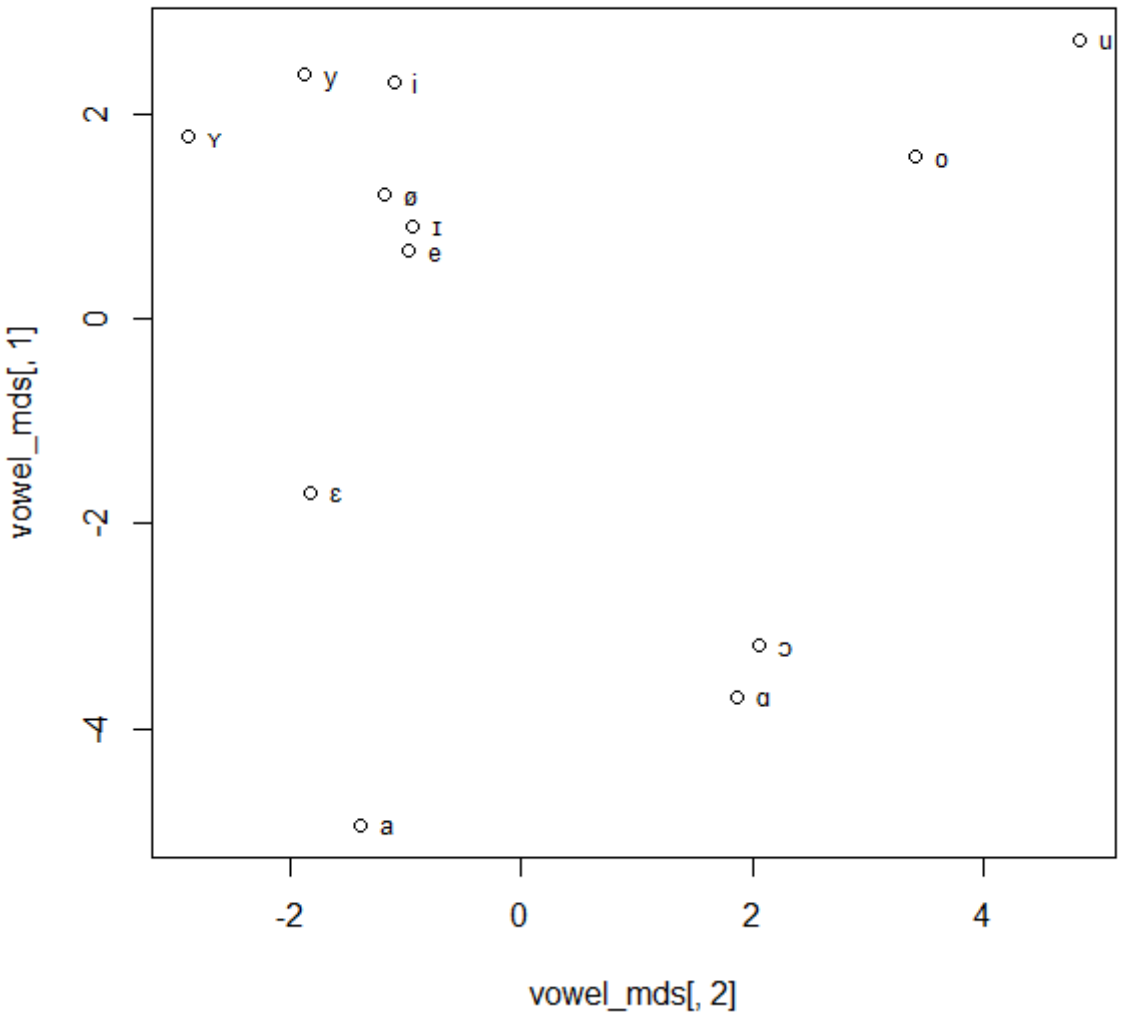

Figure 7: Acoustic vowel plot

References

Gotzon Aurrekoetxea, John Nerbonne, and Jesus Rubio. 2020. Unifying Analyses of Multiple Responses. *Dialectologia*, 25:59–86.

Martijn Bartelds, Caitlin Richter, Mark Liberman, and Martijn Wieling. 2020. A New Acoustic-Based Pronunciation Distance Measure. *Frontiers in Artificial Intelligence*, 3, May.

Charlotte Gooskens and Wilbert Heeringa. 2004. Perceptive evaluation of Levenshtein dialect distance measurements using Norwegian dialect data. *Language Variation and Change*, 16(3):189–207, October.

Wilbert Heeringa. 2004. *Measuring Dialect Pronunciation Differences using Levenshtein Distance*. PhD thesis, University of Groningen, January.

Wilbert Heeringa, Peter Kleiweg, Charlotte Gooskens, and John Nerbonne. 2006. Evaluation of String Distance Algorithms for Dialectology. In *Proceedings of the Workshop on Linguistic Distances*, pages 51–62, Sydney, Australia, July. Association for Computational Linguistics.

Peter Ladefoged. 2005. Vowels. June.

Therese Leinonen, Çağrı Çöltekin, and John Nerbonne. 2016. Using Gabmap. *Lingua*, 178:71–83, July.

John Nerbonne, Rinke Colen, Charlotte Gooskens, Therese Leinonen, and Peter Kleiweg. 2011. Gabmap A web application for dialectology. *Dialectologia*, SI II:65–89.

John Nerbonne, Wilbert Heeringa, and Peter Kleiweg. 1999. Comparison and classification of dialects. In *Proceedings of the ninth conference on European chapter of the Association for Computational Linguistics*, pages 281–282, USA, June. Association for Computational Linguistics.

Johan Taeldeman and A. Goeman. 1996. Fonologie en morfologie van de Nederlandse dialecten: een nieuwe materiaalverzameling en twee nieuwe atlasprojecten. *Taal en Tongval*, 48:38–59.

Martijn Wieling, W. Heeringa, and J. Nerbonne. 2007. An Aggregate Analysis of Pronunciation in the Goeman-Taeldeman-van Reenen-Project Data. *Taal en Tongval*, 59:84–116.

Martijn Wieling, Eliza Margaretha, and John Nerbonne. 2011. Inducing phonetic distances from dialect variation. *Computational Linguistics in the Netherlands Journal*, 1:109–118, December.

Martijn Wieling, Eliza Margaretha, and John Nerbonne. 2012. Inducing a measure of phonetic similarity from pronunciation variation. *Journal of Phonetics*, 40(2):307–314, March.

Martijn Wieling and John Nerbonne. 2011. Bipartite spectral graph partitioning for clustering dialect varieties and detecting their linguistic features. *Computer Speech & Language*, 25(3):700–715, July.

Martijn Wieling, Jelena Prokić, and John Nerbonne. 2009. Evaluating the pairwise string alignment of pronunciations. In *Proceedings of the EACL 2009 Workshop on Language Technology and Resources for Cultural Heritage, Social Sciences, Humanities, and Education*, pages 26–34, USA, March. Association for Computational Linguistics.