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. 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. MDS maps of dialect continuua), 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 MFCC-based acoustic distance.

Introduction

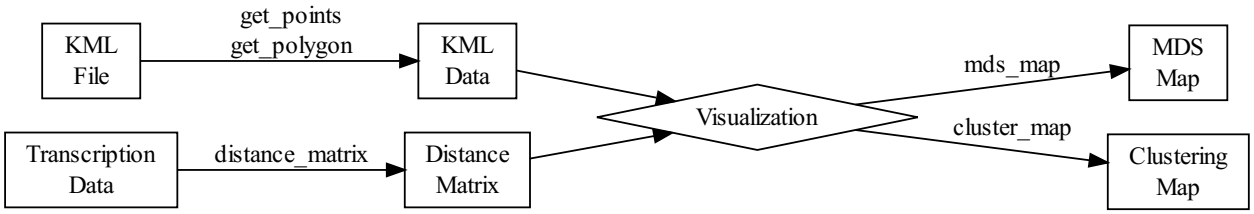


Figure 1: Minimal dialectR workflow

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 (Leinonen et al., 2016). 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. In this paper, we demonstrate the functionalities of this package by following the steps in Figure 1 to replicate a visual analysis of Dutch dialects with data from the Goeman-Taeldeman-Van Reenen-project.

Edit Distance in Dialectometry

Distance computations based on modified forms of edit distance constitute an important part of dialectometry, 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

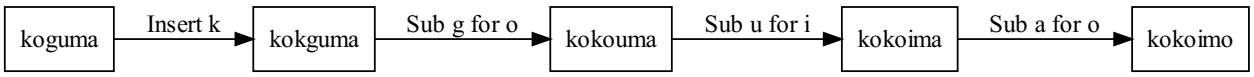


Figure 2: Edit distance between "koguma" and "kokoimo"

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, 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 results in substantially better results (), 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":

```
## [1] 1
```

```
## [1] 2
```

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
```

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:

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

```
## [1] 0.5
```

```
vc_leven("vir/via", "via/jura", delim = "/")
```

```
## [1] 1.25
```

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, since many analyses in dialectometry have been done on this, and may serve as a backdrop against which to compare the results of this package. The data from the project may be loaded as below. 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 a sample of this below:

```
load("C:\\Users\\USER\\Documents\\R_packages\\dialectR\\data\\Dutch.rda")
Dutch[1:3, 1:2]
```

```
##                             aarde                         adem
## Aalsmeer NH <U+0294><U+0252>rde <U+0294><U+0252>d<U+0259>m
## Aalst BeLb    <U+025B><U+0259>t                os<U+0259>m
## Aalst BeOv  e<U+025B>rd<U+0259>                os<U+0259>m
```

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:

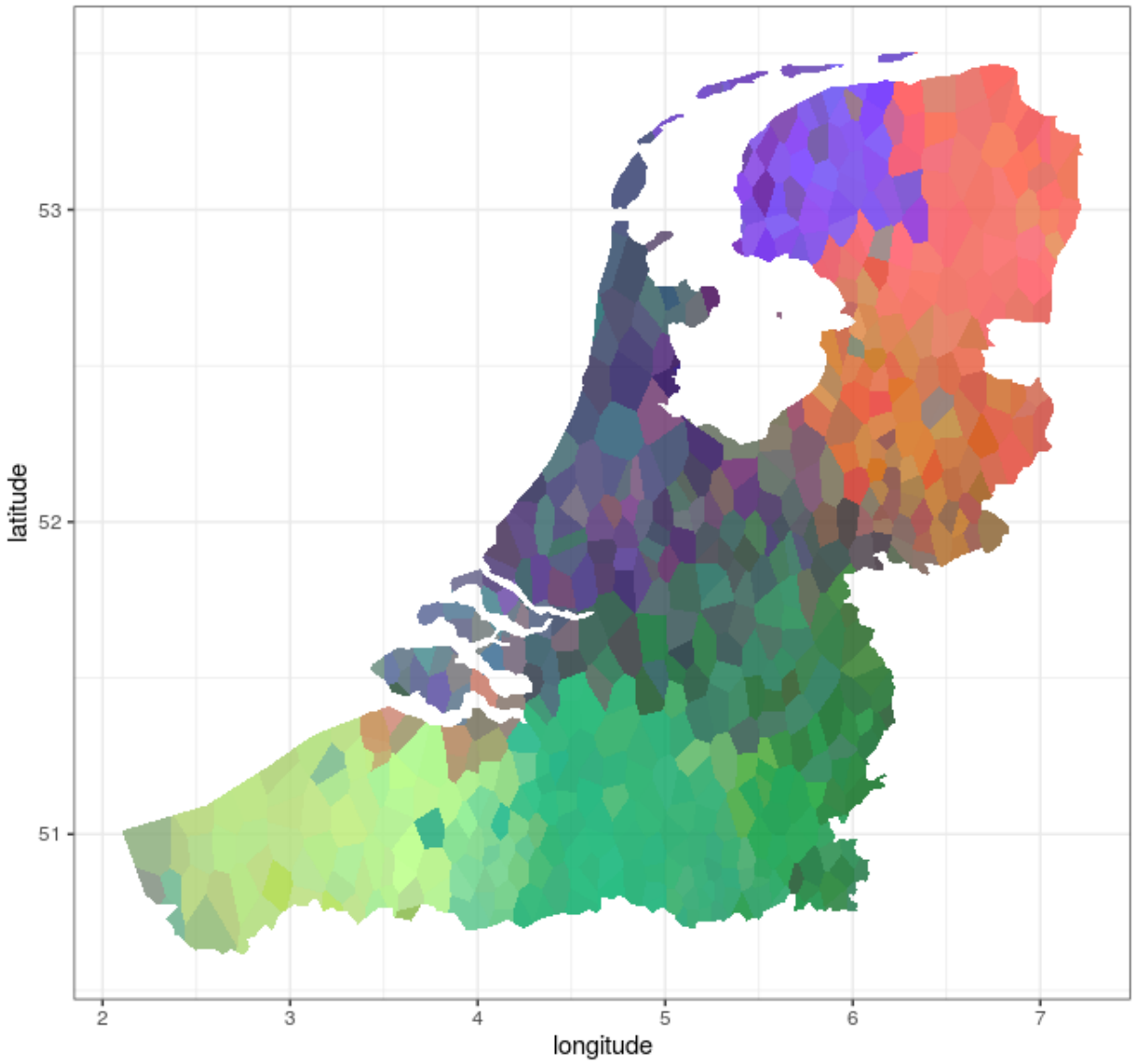
```
distDutch <- distance_matrix(Dutch[1:50, 1:50], "vc_leven", alignment_normalization = TRUE)
distDutch[1:3,1:3]
```

```
##             Aalsmeer NH Aalst BeLb Aalst BeOv
## Aalsmeer NH     0.00000   19.61190   22.42302
## Aalst BeLb     19.61190    0.00000   18.71905
## Aalst BeOv     22.42302   18.71905    0.00000
```

Multidimensional Scaling

With the results obtained above, we are now at a position to reproduce a map of the Dutch dialect continuum as shown in (XXX). In essence, this procedure performs classical multidimensional 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. The function and the result is as below:

```
mds_map(distDutch)
```



Clustering

We also provide functions for hierarchical clustering which, as it is based on R's native hclust method (XXX), 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 the weighted average and Ward's method, as shown below:

```
cluster_map(distDutch)
cluster_map(distDutch)
```

Extending dialectR

```
dfDutch <- distmat_to_df(distDutch)
dfDutch[1:5,]
```

```
##           row        col      sim
## 1 Aalsmeer NH Aalst BeLb 19.61190
```

```
## 2 Aalsmeer NH Aalst BeOv 22.42302
## 3  Aalst BeLb Aalst BeOv 18.71905
## 4 Aalsmeer NH  Aalten Gl 19.91429
## 5  Aalst BeLb  Aalten Gl 19.67143
```

```
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 4.0.4
```

```
dfDutch %>%
  filter(row == "Aalsmeer NH")
```

```
##            row                            col       sim
## 1  Aalsmeer NH                      Aalst BeLb 19.611905
## 2  Aalsmeer NH                      Aalst BeOv 22.423016
## 3  Aalsmeer NH                       Aalten Gl 19.914286
## 4  Aalsmeer NH                    Aardenburg Ze 30.127778
## 5  Aalsmeer NH                 Aarlanderveen ZH  8.417857
## 6  Aalsmeer NH                    Aarschot BeBr 17.300000
## 7  Aalsmeer NH                      Abbekerk NH 18.915476
## 8  Aalsmeer NH                    Achterberg Ut 17.246032
## 9  Aalsmeer NH                        Aduard Gn 27.370635
## 10 Aalsmeer NH                      Afferden Lb 22.046825
## 11 Aalsmeer NH                        Akkrum Fr 25.271825
## 12 Aalsmeer NH                         Almen Gl 25.371429
## 13 Aalsmeer NH                      Almkerk NB 15.680952
## 14 Aalsmeer NH                       Alphen NB 18.209524
## 15 Aalsmeer NH                   Alsemberg BeBr 19.511111
## 16 Aalsmeer NH                        Ameide ZH 17.382540
## 17 Aalsmeer NH                    Amersfoort Ut 21.667857
## 18 Aalsmeer NH                       Angerlo Gl 19.242857
## 19 Aalsmeer NH             Anjum Fr - Eanjum Fr 25.832540
## 20 Aalsmeer NH                        Anloo Dr 22.618651
## 21 Aalsmeer NH                  Antwerpen BeAnt 17.695635
## 22 Aalsmeer NH                    Apeldoorn Gl 27.463492
## 23 Aalsmeer NH       Appelscha Fr - Appelskea Fr 27.813095
## 24 Aalsmeer NH                    Appingedam Gn 26.856349
## 25 Aalsmeer NH                   Arendonk BeAnt 19.449206
## 26 Aalsmeer NH                    Arnemuiden Ze 22.778571
## 27 Aalsmeer NH                         Arum Fr 25.355952
## 28 Aalsmeer NH                       Asse BeBr 21.963492
## 29 Aalsmeer NH                    Assenede BeOv 29.571429
## 30 Aalsmeer NH                    Austerlitz Ut 23.419444
## 31 Aalsmeer NH                     Avelgem BeWv 26.333333
## 32 Aalsmeer NH                          Axel Ze 23.392857
## 33 Aalsmeer NH                  Baardegem BeOv 24.408730
## 34 Aalsmeer NH                Baarle-Nassau NB 21.646825
## 35 Aalsmeer NH                        Bakel NB 22.657143
## 36 Aalsmeer NH       Bakkeveen Fr - Bakkefean Fr 25.229762
## 37 Aalsmeer NH                     Balen BeAnt 19.915873
## 38 Aalsmeer NH                     Balkbrug Ov 22.797619
## 39 Aalsmeer NH           Barger-Oosterveld Dr 23.761111
## 40 Aalsmeer NH                    Barneveld Gl 16.757143
## 41 Aalsmeer NH                      Bathmen Ov 24.148810
## 42 Aalsmeer NH                         Beek NB 21.024603
```

```
## 43 Aalsmeer NH Beetgumermolen Fr - Bitgummole Fr 27.780556
## 44 Aalsmeer NH                      Bellegem BeWv 27.966667
## 45 Aalsmeer NH                    Bellingwolde Gn 27.747619
## 46 Aalsmeer NH                       Benschop Ut 20.202778
## 47 Aalsmeer NH                  Bergen op Zoom NB 16.128571
## 48 Aalsmeer NH                     Bergentheim Ov 24.937302
## 49 Aalsmeer NH                      Beringen BeLb 17.730952
```

```
  # ggplot for ref point and beammap
```

MFCC-based Acoustic Distance

While dialectR has followed Gabmap in implementing many of the functionalities, we have additionally made available an acoustic distance published by Martijn Bartelds et al. in a recent paper, which makes it possible to perform the 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 a number of times, both in the paper by Bartelds et al. and in dialectometry in general (), and thus again may serve as a good backdrop of comparison.

For the data, we make use of (cardinal?) vowels as pronounced by Peter Ladefoged (), which we refer the reader to download directly from the website if she wishes to reproduce the results. 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],
                    list.files("C:/Users/USER/Downloads/ipa_vowels", full.names = TRUE)[y])
  })
})

vowel_names <- c(" ", "a", "e", "i", " ", "o", " ", " ", "ø", "u", "y", " ")
row.names(vowel_dist) <- vowel_names
colnames(vowel_dist) <- vowel_names

vowel_dist
```

```
##           <U+025B>         a         e         i <U+026A>         o  <U+0251>
## <U+025B>  0.000000  9.518959  9.356685  9.714830 8.488894 10.308562  9.839774
## a         9.518959  0.000000 10.494420 10.295286 9.211383 10.861628  9.708700
## e         9.356685 10.494420  0.000000  9.413114 8.331382  9.796983 10.198128
## i         9.714830 10.295286  9.413114  0.000000 7.716021  9.558167  9.948395
## <U+026A>  8.488894  9.211383  8.331382  7.716021 0.000000  8.539531  8.926520
## o        10.308562 10.861628  9.796983  9.558167 8.539531  0.000000 10.480151
## <U+0251>  9.839774  9.708700 10.198128  9.948395 8.926520 10.480151  0.000000
## <U+0254> 10.101746 10.105714  9.931819 10.102632 8.683098 10.111965  9.684038
## ø         8.952950  9.714352  9.124663  8.552038 7.131615  9.688471  9.626592
## u        10.522844 11.301530 10.482043  9.893002 9.287932 10.121782 10.505822
## y         8.878690  9.863611  8.681085  7.485968 6.737329  9.152288  9.857818
## <U+028F>  9.386086 10.023431  9.133382  8.528394 7.112040  9.938591  9.873471
##           <U+0254>        ø         u        y  <U+028F>
## <U+025B> 10.101746 8.952950 10.522844 8.878690  9.386086
## a        10.105714 9.714352 11.301530 9.863611 10.023431
## e         9.931819 9.124663 10.482043 8.681085  9.133382
```

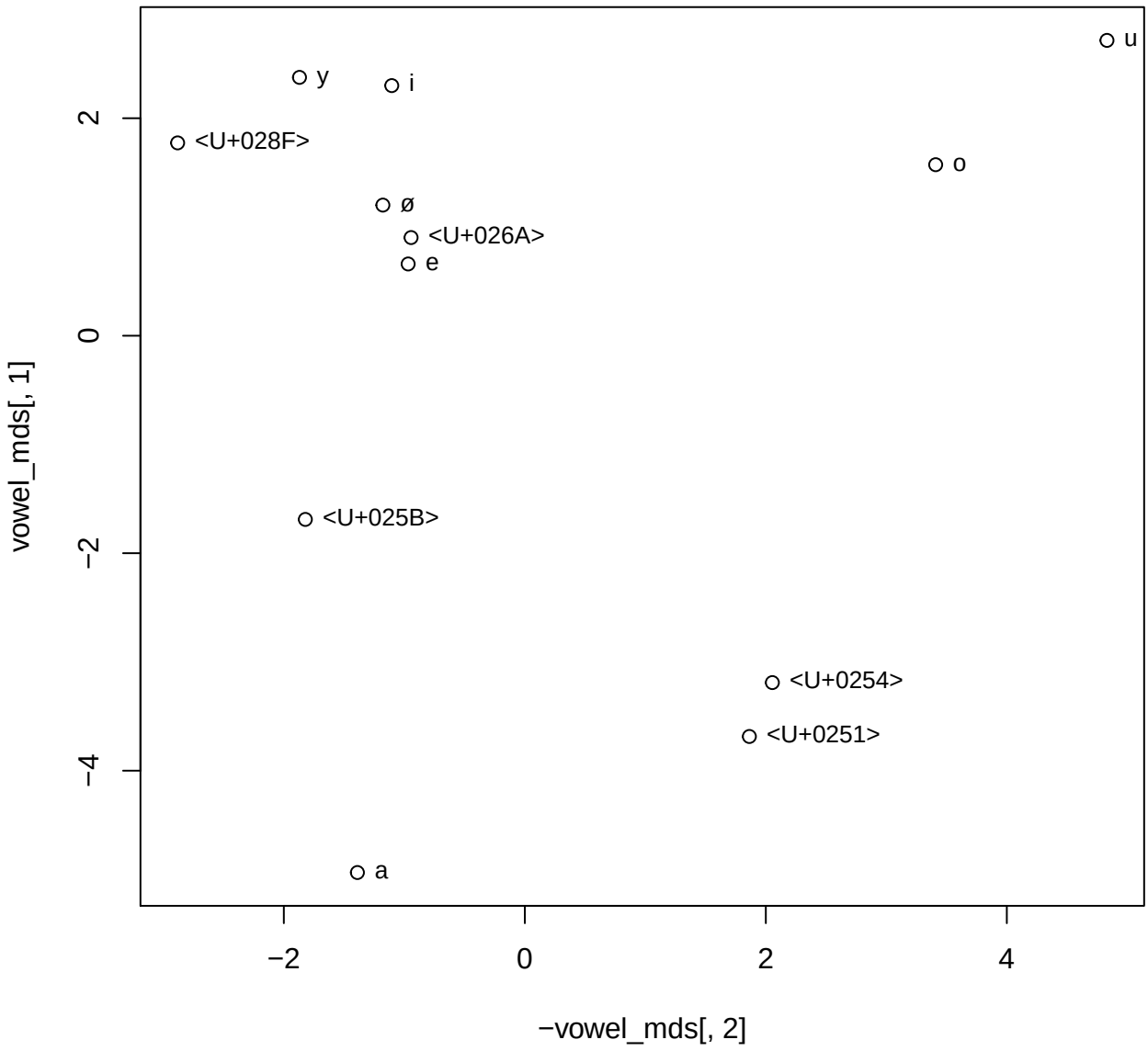
```
## i        10.102632 8.552038  9.893002 7.485968  8.528394
## <U+026A>  8.683098 7.131615  9.287932 6.737329  7.112040
## o        10.111965 9.688471 10.121782 9.152288  9.938591
## <U+0251>  9.684038 9.626592 10.505822 9.857818  9.873471
## <U+0254>  0.000000 9.700367 10.919620 9.823628 10.022058
## ø         9.700367 0.000000  9.336642 7.955654  7.818396
## u        10.919620 9.336642  0.000000 9.486313 10.179041
## y         9.823628 7.955654  9.486313 0.000000  7.303921
## <U+028F> 10.022058 7.818396 10.179041 7.303921  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:

```
vowel_mds <- cmdscale(vowel_dist, k = 3)

plot(-vowel_mds[,2],
     vowel_mds[,1])

text(-vowel_mds[,2],
     vowel_mds[,1],
     cex=0.8,
     labels = vowel_names, pos = 4)
```



Which would, on the whole, appear to be consistent with a formant-based acoustic vowel space, as confirmed by the results in the original paper.

Conclusion

dialectR sets out to fill a unique gap in dialectology research, 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 self-proclaimed shortcomings of Gabmap, where " . . . 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" [???]. 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 pointwise-mutual-information-based edit distance [???], 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.

References

Aurrekoetxea, G., Nerbonne, J., & Rubio, J. (2020). Unifying Analyses of Multiple Responses. *Dialectologia*. https://doi.org/10.1344/Dialectologia2020.25.4

Heeringa, W. (2004). *Measuring Dialect Pronunciation Differences using Levenshtein Distance*. 324.

Heeringa, W., Kleiweg, P., Gooskens, C., & Nerbonne, J. (2006). Evaluation of String Distance Algorithms for Dialectology. *Proceedings of the Workshop on Linguistic Distances*, 51–62.

Leinonen, T., Çöltekin, Ç., & Nerbonne, J. (2016). Using Gabmap. *Lingua*, *178*, 71–83. https://doi.org/10.1016/j.lingua.2015.02.004