Package 'qlcVisualize'

August 29, 2016

Type Package

Title Visualization for Quantitative Language Comparison (QLC)

Version 0.1.0

Date 2015-10-20

Author Michael Cysouw

Maintainer Michael Cysouw <cysouw@mac.com>

Description Collection of visualizations as used in quantitative language comparison.

License ACM | file LICENSE

Encoding UTF-8

Imports fields, MASS, qlcMatrix, seriation, spatstat, raster, maps, mapdata, maptools, mapplots, methods, sp, alphahull

NeedsCompilation no

Repository CRAN

Date/Publication 2015-10-23 08:50:33

R topics documented:

qlcVisualize-package *Visualizations for Quantitative Language Comparison (QLC)*

Description

A collection of specific visualisations of data as used in QLC.

Details

Package:	qlcVisualize
Type:	Package
Version:	0.1.0
Date:	2015-10-20
License:	GPL-3

Currently implemented are visualisations dealing nominal data with multiple levels lmap ("level-map") and limage "level-image", and some assistance for making geographic voronoi maps vmap.

Author(s)

Michael Cysouw <cysouw@mac.com>

boundary *Checking boundary parameters vor plotting of* lmap

Description

The function lmap can be tweaked by various parameters determining the boundary of the interpolation. The function boundary helps finding suitable parameters.

Usage

```
boundary(points, density = 0.02, grid = 10, box.offset = 0.1
  , tightness = "auto", manual = NULL, plot = TRUE)
```

Arguments

points	Points, typically a two-column matrix with x and y coordinates.
density	Density of points below which there should be no interpolation.
grid	Density of the grid.
box.offset	Distance of the box around the points.

`tightness`	Parameter influencing how tightly the boundary should be wrapped around the points. Passed internally to kde2d. When `"auto"` this defaults to bandwidth.nrd. Lower values will result in tighter boundaries.
`manual`	Manually added boundary points in the form of a two-column matrix with coordinates.
`plot`	Logical: by default the impact of the chosen parameters is shown. If `FALSE` then coordinates are returned that are the outside of the boundary.

Details

Instead of trying to use a polygon as a boundary for the interpolation internally in `lmap` it turned out to be easier to use a collection of points that mark th

Value

By default, returns a plot with the original points in black, the points below density in red, and the box around the points in blue. Contour lines of the density are shown to choose different density parameters.

When `plot = FALSE`, the blue and red points from the graphic are returned as a two-column matrix of x and y coordinates.

Author(s)

Michael Cysouw <cysouw@mac.com>

See Also

Used internally in `lmap`. The parameters of this function can be passed through, typically `density` and `box.offset`.

Examples

```
data(hessen)

# show impact of the chosen parameters
boundary(hessen$villages, density = 0.1, grid = 20
  , manual = cbind(x = c(8.3, 9.2), y = c(49.9, 50.0)))

# return coordinates
boundary(hessen$villages, plot = FALSE)

# abstract example, showing tightness in action
p <- cbind(c(1:10, 1:10), c(1:10, 10:1))
par(mfrow = c(1,3))
boundary(p, density = 0.005, grid = 20, tightness = "auto")
boundary(p, density = 0.005, grid = 20, tightness = 5)
boundary(p, density = 0.005, grid = 20, tightness = 3)
par(mfrow = c(1,1))
```

dialects *Multiple correspondences of "f"-like sounds in German Dialects*

Description

In total 34 different words in which an f-like sound occurs. The differnt pronunciations of this sound in 183 different German villages are included in this dataset.

Usage

```
data(dialects)
```

Format

List of 2:

`villages` Dataframe with two variables LONGITUDE and LATITUDE for all 183 villages.

`data` Matrix with 34 columns showing the pronunciation in the 183 villages.

Source

Excerpt from http://github.com/cysouw/PAD/

Examples

```
data(dialects)

require(mapdata)
map("worldHires", "Germany", fill = TRUE, col = "grey90")

lmap(dialects$villages, dialects$data[,21]
  , levels = c(0.20, 0.22, 0.24), add = TRUE, position = "topleft")

title(main = "f-sound in \'Kochlöffel\'")
```

haspelmath *Data about indefinite constructions in 39 different languages*

Description

Summary of the data in Haspelmath (1997), classical example of the usage of semantic maps.

Usage

```
data(haspelmath)
```

Format

The dataset is a matrix with nine rows, describing the different indefinite functions, and 134 columns documenting for each individual construction which functions are possiblu expressed by it.

Source

Haspelmath, Martin. Indefinite Pronouns. Oxford Studies in Typology and Linguistic Theory. Oxford: Clarendon, 1997.

Examples

```
data(haspelmath)
## English data
haspelmath[,7:9]
```

---

heeringa *Heeringa-style colours*

Description

Proposed in Heeringa (2004) to colour a (dis)similarity by decomposing it into three dimensions (using `cmdscale` here) and then mapping these dimensions to RGB to make colours. Hihgly useful to visualize pairwise similarities between geographic regions.

Usage

```
heeringa(dist, power = 0.5, mapping = c(1, 2, 3))
```

Arguments

`dist`	`dist` object specifying distances between points.
`power`	Factor used to influence the results of the multidimensional scaling. Values closer to one will lead to clearer separated colours, while higher values will lead to more gradual colours.
`mapping`	Optional vector to change the mapping of the dimensions to the colours. Should be of length 3, specifying to which color each of the three dimensions is mapped. A 1 means 'red', a 2 meand 'green' and a 3 means 'blue'. Adding a minus reverses the mapping.

Details

This proposal goes back to Heeringa (2004). The idea is to visualize distances by mapping the first three dimensions of a multidimensional scaling to the the red-green-blue scales. The `mapping` vector can be used to change the mapping to the colours.

Value

A vector of colours of the same length as the size of the dist object.

Author(s)

Michael Cysouw <cysouw@mac.com>

References

Heeringa, Wilbert. "Measuring Dialect Pronunciation Differences Using Levenshtein Distance." Ph.D. Thesis, Rijksuniversiteit Groningen, 2004.

Examples

```
data(hessen)
tessalation <- voronoi(hessen$villages, hessen$boundary)
d <- dist(hessen$data, method = "canberra")

# different mappings of the colors
c1 <- heeringa(d)
vmap(tessalation, col = c1, border = NA)

c2 <- heeringa(d, power = 1, mapping = c(3, -2, 1))
vmap(tessalation, col = c2, border = NA)
```

---

hessen *Extract from the SyHD Project on the syntax of the dialect of Hessen (Germany)*

Description

An example dataset of dialect data.

Usage

```
data(hessen)
```

Format

List of 3

`boundary` An object of type `owin` describing a geographical boundary. This format is necessary for voronoi diagrams.

`villages` A dataframe with two variables "longitude" and "latitude" for the 157 villages on this there is data in this dataset.

`data` Dataframe with 56 different characteristics of these 157 villages, distributed over 15 different variables (as indicated in the column names).

Source

Data from https://www.uni-marburg.de/fb09/dsa/projekte/syhd

References

Jürg Fleischer, Simon Kasper & Alexandra N. Lenz (2012): Die Erhebung syntaktischer Phänomene durch die indirekte Methode: Ergebnisse und Erfahrungen aus dem Forschungsprojekt "Syntax hessischer Dialekte" (SyHD). In: Zeitschrift für Dialektologie und Linguistik 79/1, 2-42.

Examples

```
data(hessen)

tessalation <- voronoi(hessen$villages, hessen$boundary)
plot(tessalation)
```

`limage` *Drawing multi-level images for visualisation of nominal data with various levels.*

Description

A multi-level image ("l-image") is a counterpart of the base function `image` for nominal data with various levels. A matrix (or dataframe coerced as matrix) is visualised by showing the most frequent contents of the cells by colouring. There are various methods for ordering of rows and columns provided (like with `heatmap`).

Usage

```
limage(x, col = rainbow(4), order = NULL,
  show.remaining = FALSE, cex.axis = 1, cex.legend = 1, cex.remaining = 1,
  font = "", asp = 1, method = "hamming", control = NULL, plot = TRUE)
```

Arguments

`x`	A matrix or dataframe with the data to be displayed. Rows are shown on the x-axis, columns on the y-axis, showing the row- and column-names in the display. All data in the whole matrix is interpreted as one large factor with different levels.
`col`	Colors to be used for the display. The number of colours specified here are used for the top most frequent phenomena in the data. All other are shown as 'others'. Optionally use `show.remaining` to show these others in the visualisation
`order`	How should rows and columns be ordered? By default the order of the data matrix `x` is used. Many possible orderings are available, see Details.
`show.remaining`	Logical: should all levels without color be shown inside the boxes?
`cex.axis`	Size of the row and columns names of `x`, shown as axis labels.
`cex.legend`	Size of the legend text.
`cex.remaining`	Size of the text in the boxes. Only shown when `show.remaining = TRUE`.

font	Font to be used in the plotting, can be necessary for unusual unicode symbols. Passed internally to par(family).
asp	Aspect-ratio of the plotting of the boxes. By default the boxes are drawn square. Manually resizing the boxes by changing the plotting window can be acheived by setting asp = NA.
method	Method used to determine similarity, passed to sim.obs, which is used internally to determine the order of rows and columns, using the method chosen in order.
control	List of options passed to seriate.
plot	By default, a plot is returned. When FALSE, nothing is plotted, but the reordering is returned.

Details

There are many different orderings implemented: "pca" and "varimax" use the second dimension of prcomp and varimax respectively. "mds" will use the first dimension of cmdscale.

Further, all methods as provided in the function seriate can be called. Specifcally, "R2E" and "MDS_angle" seem worthwhile to try out. Any paramters for these methods can be passed using the option control.

Value

A plot is returned by default. When plot = FALSE, a list is returned with the reordering of the rows and the columns.

Note

Note that it is slightly confusing that the image is sort-of a transposed version of the data matrix (rows of the matrix are shown as horizontal lines in the graphic, and they are shown from bottom to top). This is standard practice though, also used in image and heatmap, so it is continued here.

Author(s)

Michael Cysouw <cysouw@mac.com>

See Also

image in base and pimage in the package seriation.

Examples

```
# a simple data matrix
x <- matrix(letters[1:5],3,5)
x[2,3] <- x[1,4] <- NA
rownames(x) <- c("one", "two", "three")
colnames(x) <- 1:5
x

# some basic level-images
limage(x)
```

```
limage(x, col = heat.colors(5), asp = NA)
limage(x, col = list(b="red",e="blue"), show.remaining = TRUE)

## Not run:
# more interesting example, different "f" sounds in german dialects
# note that fonts might be problematic on some platforms
# plotting window should be made really large as well
data(dialects)
limage(dialects$data, col = rainbow(8), order = "R2E"
    , cex.axis = 0.3, cex.legend = 0.7
    , show.remaining = T, cex.remaining = 0.2)

# get reordering of rows
# to identify the group of words with "p-f" correspondences
limage(dialects$data, order = "R2E", plot = FALSE)

## End(Not run)
```

`lmap` *Drawing multi-level maps (e.g. semantic maps or linguistic isoglosses)*

Description

A multi-level map ("l-map") is a plot of the distribution of nominal data with multiple levels in space. Such visualisations have two direct use-case in linguistics, viz. semantic maps and isoglosses. The drawing if the lines in space is performed by interpolation in this function (see details).

Semantic maps (Haspelmath 2003) are a visualisation of linguistic diversity. A semantic map shows a predefined configuration of functions/senses in two-dimensional space with an overlay of language-specific encoding of these functions/senses. An level-map tries to emulate this linguistic visualiation in an automatic fashion with various options for visual presentation.

Isoglosses show lines surrounding similar phenomena in space. Instead of drawing an exact boundary around measured points, an interpolation-technique is used here to show areas of interest. By only showing boundaries, multiple phenomena can be shown in one graphic.

Usage

```
lmap(points, data,
   main = NULL, draw = 5, levels = c(0.41, 0.46, 0.51),
   labels = NULL, cex = 0.7, col = "rainbow", add = FALSE, lambda = NA,
   legend = TRUE, position = "bottomleft", cex.legend = 0.7, font = "",
   note = TRUE, file.out = NULL, ...)
```

Arguments

`points`	Coordinates of the data points specified as a two-column matrix or dataframe.

`data`	Language data to be plotted as contour-overlay over the points. Either specified as a vector of language-specific forms, or as a numeric matrix with the forms as columns and the points as rows (the language-specific forms should be specified as `colnames`). The values in the matrix designate the occurrence of the forms, allowing for the encoding of frequency/typicality and of overlap of different forms being used in the same function. see Details.
`main`	Title for the plot
`draw`	Which forms to be drawn by contours. Specifying a numeric value will only draw the uppermost frequent forms in the data, by default only the topmost five forms are drawn (automatically ordered by frequency). Alternatively, a vector with names or column-indices of the forms to be drawn can be specified.
`levels`	height of contours to be drawn. Internally, all values are normalized between zero and one, so only values between those extremes are sensible. Line thickness is automatically balanced.
`labels`	Optionally, labels for the points to be drawn instead of symbols in the plot.
`cex`	Character expansion of the labels (see previous option). Also influences the size of symbols or pie-charts.
`col`	Colour specification, either in the form of the name of a built-in color palettes, like rainbow, or a manually specified vector of colors. When NULL, an attempt is made to use grey-scales.
`add`	Logical: should the plot be added to an exitings plot or not?
`lambda`	Parameter for the interpolation, passed internally to the function Krig. Low values result in more detailed boundaries around the measured points.
`legend`	Logical: should a legend be added?
`position`	Where should the legend be positioned? Passed internally to legend.
`cex.legend`	Character expansion passed to legend, and also used for the indication of the levels in the plot
`font`	Font to be used for the legend and the labels. Passed internally to `par(family)`.
`note`	Logical: should a note be added to the bottom of the graphic to document the levels of the countour lines?
`file.out`	Location for writing the image to a file instead of plotting it on screen
`...`	Additional parameters optionally passed to boundary for the specification of the area of interpolation.

Details

The basic idea is to use some kind of interpolation to show areas of high-occurrence of a specific phonenomenon. Internally Kriging is used, and then only contour lines are shown of the interpolation. Multiple lines are suggested to indicate the probibabilistic interpretation of the lines.

Value

A plot is produces with the different phenomena in space surrounded by lines. When multiple options are possible at each point then pie charts are added.

Author(s)

Michael Cysouw <cysouw@mac.com>

Examples

```
# isogloss example
# choose one feature from hessen dataset (number 4)
data(hessen)
f4 <- hessen$data[,9:13]

# look for area for interpolation, changing density and grid parameters
# suitable parameters can be passed through to function lmap below
boundary(hessen$villages, density = 0.1, grid = 10)

# useful size of pies has to be determined by changing cex
plot(hessen$boundary, main = NULL)
lmap(hessen$villages, f4, draw = 3, cex = 0.8
  , density = 0.1, grid = 10, add = TRUE, cex.legend = 0.5)

# another isogloss example:
# "f" sounds in German dialects in the words "Kochlöffel"
require(mapdata)
map("worldHires", "Germany", fill = TRUE, col = "grey90")

data(dialects)
lmap(dialects$villages, dialects$data[,21], levels = c(0.20, 0.22, 0.24)
    , add = TRUE, position = "topleft")
title(main = "f-sound in \'Kochlöffel\'")

# semantic map example
# location of points via multidimensional scaling of complete data
data(haspelmath)
d <- dist(haspelmath)
p <- MASS::isoMDS(d)$points

# testing boundary parameters
boundary(p)
boundary(p, density = 0.004, box = 0.15, tightness = 8)

# labels to be plotted instead of points
text <- gsub("\\.", "\n", rownames(haspelmath))

# show a few languages
# using a quick dummy function to set all parameters
tmp <- function(columns) {
  lmap(p, haspelmath[,columns]
    , levels = 0.1, labels = text
    , density = 0.004, box = 0.15, tightness = 8
    , lambda = 0.1, note = FALSE)
}

## Not run:
```

```
par(mfcol = c(2,3))
tmp(1:3)
tmp(4:6)
tmp(7:9)
tmp(10:12)
tmp(13:17)
tmp(18:22)
par(mfcol = c(1,1))
## End(Not run)
```

---

vmap *Plotting a Voronoi-map ("v-map")*

Description

A "voronoi-map" (voronoi-tessellation, also known as dirichlet tessellation) is used in quantitative dialectology. This function is a convenience wrapper to easily produce dialect maps with voronoi tessellations. Also described here are a helper functions to produce the tessellation.

Usage

```
vmap(tessellation, col = NULL, add = FALSE,
  outer.border = "black", border = "grey", lwd = 1, ...)

voronoi(points, window)
```

Arguments

`tessellation`	Tessellation of class `tess` from the library `spatstat`. Can easily be produces by using the convenience function `voronoi` provided here.
`col`	Vector of colors for the filling of the tessellation. Is recycled when there are more tiles than colours. The order of the tiles is the same as the order of the points as specified in the function `voronoi`.
`add`	Add graphics to an existing plot
`outer.border`	Colour of the outer border. Specifying `NA` removes the border.
`border`	Colour of the inner borders. Specifying `NA` removes all borders.
`lwd`	Line width of borders.
`...`	Further arguments passed to polygon.
`points`	Two-column matrix with all coordinates of the points to make a voronoi tessellation.
`window`	Outer boundary for the voronoi tessellation. Should be in the form of an `owin` object. There are two helper functions provided here to get such object. Note that the function `voronoi` will give warnings if there are points outside of this window.

Details

This code is almost completely based on functions from the spatstat package. For convenience, first some geographical boundaries can easily be accessed and converted for use in spatstat. Then a voronoi tessellation can be made (based on the function dirichlet, which in turn is based on deldir). Finally, this tessalation can be plotted filled with different colours.

Any legends have to be added manually by using legend, see examples below.

The function voronoi returns a warning when points are attested that lie outside of the specified border. For these points there is no polygon specified. Indices for the rejected points outside the border can be accessed by attr(x, "rejects").

Value

voronoi returns a tessellation of the class tess from the package spatstat. When points outside of the border are attested, the indices of these points are added to an attribute "rejects". vmap plots a map.

Author(s)

Michael Cysouw <cysouw@mac.com>

Examples

```
# make a voronoi tessellation for some villages in hessen
data(hessen)
plot(hessen$boundary)
points(hessen$villages, cex = 0.3)

tessellation <- voronoi(hessen$villages, hessen$boundary)
plot(tessellation)

# make a resizable plot with random colour specification
vmap(tessellation, col = rainbow(5), border = NA)
legend("bottomright", legend = c("a","b","c","d","e"), fill = rainbow(5))

# use actual colors from data, using first feature from supplied data
# multiple levels cannot easily be shown
# consider \link{lmap} for more detail
d1 <- hessen$data[,1:3]
d1 <- d1[,1]/rowSums(d1)
vmap(tessellation, col = rgb(1, 1-d1, 1-d1))
text(hessen$villages,labels=hessen$data[,1],cex=.5)
legend("bottomright", legend = c("es mir", "mir es / other"),
  fill = c("red", "white"))

# Use distances to determine colour, as proposed by Heeringa (2004)
# Note that different methods to establish distances can lead to rather
# different results! Also try method = "euclidean"
d <- dist(hessen$data, method = "canberra")
cols <- heeringa(d)
vmap(tessellation, col = cols, border = NA)
```

window *Producing windows of class "owin"*

Description

Different ways to easily produce windows of class "owin" from the package "spatstat" are presented here. These are used by voronoi.

Usage

```
hullToOwin(points, shift, alpha)
mapsToOwin(country, database = "worldHires")
gadmToOwin(country, sub = NULL, level = 0)
```

Arguments

points	Set of points that need a window around them. Two column matrix.
shift	The amount of space around the outer points at determining the window.
alpha	Parameter for the 'curviness': lower values show more detail. Passed internally to ahull.
country	Name of the country to obtain borders and turn them into an owin object needed for the function voronoi. For mapsToOwin check map how to specify the names. For gadmToOwin, check getData.
database	Database as used by map.
sub, level	Names for Subdivisions of countries as available in the GADM database. Read the help file getData for more information about GADM, the subdivisions and how to acces them.

Details

For hullToOwin, the function ahull is used to make a hull around the points. This is then converted to an "owin" window.

The functions mapsToOwin and GadmToWin use external topogaphic boundaries to produce windows.

Value

All functions return an object of class owin from the package spatstat.

Note

Includes code from code from Andrew Bevan, based on code from Dylan Beaudette, see https://stat.ethz.ch/pipermail/r-sig-geo/2012-March/014409.html.

The function gadmToOwin needs online access to download the data. The data is saved in the current working directory, and will not be downloaded again when it is already available there.

Author(s)

Michael Cysouw <cysouw@mac.com>

Examples

```
# Boundary of the German state "Hessen"
# This will need to access the GADM database online
## Not run:
boundary <- gadmToOwin("DEU","Hessen",1)
## End(Not run)

# A window does not have to be continuous
## Not run:
random <- mapsToOwin(c("Germany","Greece"))
plot(random, main = NULL)
## End(Not run)

# hull around some points
# note influence of alpha and shift
data(hessen)

hull <- hullToOwin(hessen$villages, shift = 0.2, alpha = 1)
plot(hull)
points(hessen$villages)

hull <- hullToOwin(hessen$villages, shift = 0.1, alpha = 0.2)
plot(hull)
points(hessen$villages)
```

Index