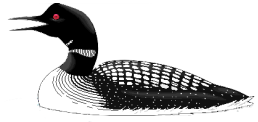
Introduction to loon

High dimensional data

. . . Toto, I've a feeling we're not in Kansas anymore . . .

Wayne Oldford
University of Waterloo

based on joint work with
Adrian Waddell
Roche



Visible minorities in Canadian cities 2006

This is Statistics Canada 2006 Census data on the counts of 14 Statistics Canada categories of "visible minorities" in 33 Canadian cities. The data are available in `loon` as `minority`.

```
library(loon)
data(minority)
names(minority)[1:14]
```

```
##  [1] "Arab"
##  [2] "Black"
##  [3] "Chinese"
##  [4] "Filipino"
##  [5] "Japanese"
##  [6] "Korean"
##  [7] "Latin.American"
##  [8] "Multiple.visible.minority"
##  [9] "South.Asian"
## [10] "Southeast.Asian"
## [11] "Total.population"
## [12] "Visible.minority.not.included.elsewhere"
## [13] "Visible.minority.population"
## [14] "West.Asian"
```

`minority` has two sets of latitude and longitude measurements as well.

(the set from Google has errors).

Visible minorities in Canadian cities 2006

For example, here are the counts for three Canadian cities (3 rows of `minority`) from across Canada:

	Victoria	Ottawa-Gatineau	St. John's
Arab	500	28195	190
Black	2360	45060	620
Chinese	12330	32445	990
Filipino	2760	7330	155
Japanese	2280	1800	65
Korean	1235	2280	45
Latin.American	1845	10630	320
Multiple.visible.minority	930	4540	25
South.Asian	7210	27130	890
Southeast.Asian	1585	11670	55
Total.population	325060	1117120	179270
Visible.minority.not.included.elsewhere	260	1720	40
Visible.minority.population	33870	179295	3460
West.Asian	575	6490	65

Visible minorities in Canadian cities 2006

Because the populations vary so much between cities we look at the percentage of the total population represented by each minority (and shorten the names).

```
minorityPercent <- data.frame(100*minority[, -c(11, 15:18)]
                              / minority[, "Total.population"])
#  Take the opportunity to shorten the Statistics Canada
#  names of the minorities as well:
names(minorityPercent) <- c("% Arabic",  "% Black", "% Chinese",
                            "% Filipino", "% Japanese",
                            "% Korean",  "% Latino", "% Multiple",
                            "% S. Asian",  "% SE Asian",  "% Other",
                            "% Minority", "% W Asian")
```

Could look at a histogram of the percent of visible minorities in each city:

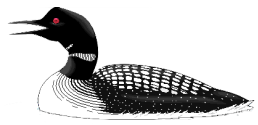
```
# The proportion of the city population that is a "visible minority"
l_hist(minorityPercent$"% Minority",
       title = "% Minority in 33 Canadian cities",
       xlabel = "percent of population",
       linkingGroup = "minority",
       yshows = "frequency",
       showBinHandle = FALSE,
       showGuides = TRUE,
       showScales = TRUE
)
```

```
## [1] ".l0.hist"
## attr(,"class")
## [1] "l_hist" "loon"
```

Top 3 cities with greatest proportion of visible minorities

```
# Order minorities by decreasing maximum percent
groups <- names(minorityPercent)[-c(12, 14, 15)]
bySize <- names(sort(apply(minorityPercent[,groups ], 2, max),
                     decreasing = TRUE) )
# And now look at the three largest
for (x in bySize[1:3]) {
  l_hist(minorityPercent[,x],
         title = x,
         xlabel = "percent of population",
         linkingGroup = "minority",
         showScales = TRUE,
         showGuides = TRUE,
         showStackedColors = TRUE,
         color = "grey"
         )
}
```

Explore the relatioships between these top three and the total population?



Link the data to a map of the cities

```
# get a map:
library(maps)
canada_map <- map("world", "Canada", plot=FALSE, fill=TRUE)
# plot the cities
cities_map <- with(minority,
                   l_plot(long, lat,
                          xlabel = "longitude", ylabel="latitude", showLabels = TRUE,
                          linkingGroup = "minority",
                          itemLabel = rownames(minority), showItemLabels = TRUE)
)

# Layer in the map of Canada
landcol <- "cornsilk"
canada_layer <- l_layer(widget = cities_map,
                        x = canada_map,
                        label = "Canada",
                        color = landcol,
                        index = "end")
# Rescale the plot to the size of the map
l_scaleto_layer(cities_map, canada_layer)

# Could also add the text of the city names to the map as glyphs
textGlyphs <- l_glyph_add_text(cities_map, text = row.names(minority),
                               label = "city names")
```

Explore the relations between histograms and map.

Interact with the inspector on the map to show text.

A pairs plot - `l_pairs()`

Could look for pairwise and conditional relationships via an interactive scatterplot matrix.

(this may take a few seconds)

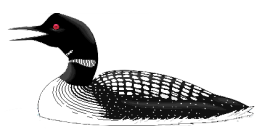
```
minPercentPairs <- l_pairs(data = minorityPercent,
                           size = 2, glyph = "ocircle",
                           linkingGroup = "minority",
                           itemLabel = rownames(minority),
                           showItemLabels = TRUE)
# a compound loon object consisting of many plots
names(minPercentPairs) [1:5]
```

```
## [1] "x2y1" "x3y1" "x4y1" "x5y1" "x6y1"
```

```
minPercentPairs$x2y1
```

```
## [1] ".l6.pairs.plot"
## attr(,"class")
## [1] "l_plot" "loon"
```

```
minPercentPairs$x2y1["glyph"] <- "otriangle"
```



Explore the relations between variates and map.

What happens when you pan and zoom on one of the plots?

Comparing cities by minority counts alone

Consider again the counts in each city, but only of each minority:

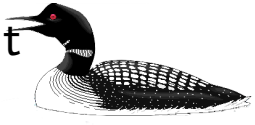
```
# Back to the raw counts
minority_only <- minority[, c(
    "Southeast.Asian", "Chinese", "Japanese", "South.Asian",
    "Visible.minority.not.included.elsewhere",
    "Black", "Multiple.visible.minority", "Filipino",
    "Arab", "Korean", "Latin.American")]
# Again, shorten the variable names
names(minority_only) <- c(
    "SE.Asian", "Chinese", "Japanese", "S.Asian", "Other",
    "Black", "Multiple", "Filipino", "Arab", "Korean", "Latino"
)
```

This data preserves the relative sizes of each minority between cities (comparison over rows).

- we might expect this to be largely determined by the total population of the cities

It also allows us to compare the relative sizes of minorities within cities

- not sure what to expect here, perhaps that some cities have different relative minority populations than others?

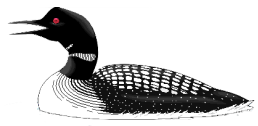


Serial axes - `l_serialaxes()`

Instead of cartesian coordinates, we might use parallel axes. Instead of a point, each city appears as a "curve" in the parallel axis system. This is a special case of **serial axes** where instead of axes being orthogonal, they appear in some serial order next to one another.

```
s <- l_serialaxes(data = minority_only,
                  axesLayout = "parallel",   # Note choice of parallel
                  linkingGroup = "minority",
                  sequence = names(minority_only),
                  showGuides = FALSE,
                  linewidth = 2,
                  scaling = "data",
                  showArea = FALSE,
                  itemLabel = rownames(minority),
                  showItemLabels = TRUE)
```

Hover over the lines to determine which cities have the largest minority populations.



Serial axes - `l_serialaxes()`

Instead of cartesian coordinates, we might use parallel axes. Instead of a point, each city appears as a "curve" in the parallel axis system. This is a special case of **serial axes** where instead of axes being orthogonal, they appear in some serial order next to one another.

```
s <- l_serialaxes(data = minority_only,
                  axesLayout = "parallel",   # Note choice of parallel
                  linkingGroup = "minority",
                  sequence = names(minority_only),
                  showGuides = FALSE,
                  linewidth = 2,
                  scaling = "data",
                  showArea = FALSE,
                  itemLabel = rownames(minority),
                  showItemLabels = TRUE)
```

Hover over the lines to determine which cities have the largest minority populations.

What does `scaling = "data"` mean?

Serial axes - `l_serialaxes()`

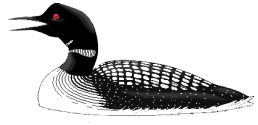
Instead of cartesian coordinates, we might use parallel axes. Instead of a point, each city appears as a "curve" in the parallel axis system. This is a special case of **serial axes** where instead of axes being orthogonal, they appear in some serial order next to one another.

```
s <- l_serialaxes(data = minority_only,
                  axesLayout = "parallel",   # Note choice of parallel
                  linkingGroup = "minority",
                  sequence = names(minority_only),
                  showGuides = FALSE,
                  linewidth = 2,
                  scaling = "data",
                  showArea = FALSE,
                  itemLabel = rownames(minority),
                  showItemLabels = TRUE)
```

Hover over the lines to determine which cities have the largest minority populations.

What does `scaling = "data"` mean?

How are lines "brushed" in the serial axes plot?



Serial axes - `l_serialaxes()`

Instead of cartesian coordinates, we might use parallel axes. Instead of a point, each city appears as a "curve" in the parallel axis system. This is a special case of **serial axes** where instead of axes being orthogonal, they appear in some serial order next to one another.

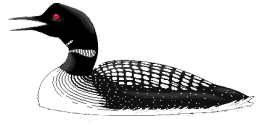
```
s <- l_serialaxes(data = minority_only,
                  axesLayout = "parallel",   # Note choice of parallel
                  linkingGroup = "minority",
                  sequence = names(minority_only),
                  showGuides = FALSE,
                  linewidth = 2,
                  scaling = "data",
                  showArea = FALSE,
                  itemLabel = rownames(minority),
                  showItemLabels = TRUE)
```

Hover over the lines to determine which cities have the largest minority populations.

What does `scaling = "data"` mean?

How are lines "brushed" in the serial axes plot?

How has the inspector changed?



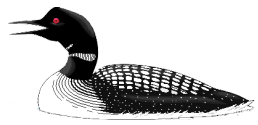
Serial axes - effect of scaling

Each serial axis only has values between 0 and 1. This means that, to appear on an axis in the plot, an observation's corresponding variable value must be in that range.

This is OK if the actual values are meaningfully between 0 and 1 (`scaling = "none"`), otherwise they need to be scaled.

The inspector presents the three other possibilities for scaling:

- `scaling = "data"` ... the min and the max over all the data map to 0 and 1 for all axes (i.e. range over all rows and columns of data)
- `scaling = "variable"` ... the min and the max for each variable map to 0 and 1 for the corresponding axis (i.e. range over each column of data)
- `scaling = "observation"` ... the min and the max over all variables for each observation map that observation's values to 0 and 1 on all axes (i.e. range over each row of the data)



Serial axes - effect of scaling

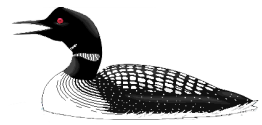
Each serial axis only has values between 0 and 1. This means that, to appear on an axis in the plot, an observation's corresponding variable value must be in that range.

This is OK if the actual values are meaningfully between 0 and 1 (`scaling = "none"`), otherwise they need to be scaled.

The inspector presents the three other possibilities for scaling:

- `scaling = "data"` ... the min and the max over all the data map to 0 and 1 for all axes (i.e. range over all rows and columns of data)
- `scaling = "variable"` ... the min and the max for each variable map to 0 and 1 for the corresponding axis (i.e. range over each column of data)
- `scaling = "observation"` ... the min and the max over all variables for each observation map that observation's values to 0 and 1 on all axes (i.e. range over each row of the data)

From the serial axes inspector, explore how each of these scalings (data, variable, observation) affect the serial axes plot **and its interpretation**.



Serial axes - effect of scaling

Each serial axis only has values between 0 and 1. This means that, to appear on an axis in the plot, an observation's corresponding variable value must be in that range.

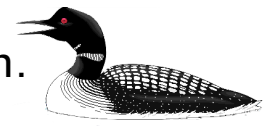
This is OK if the actual values are meaningfully between 0 and 1 (`scaling = "none"`), otherwise they need to be scaled.

The inspector presents the three other possibilities for scaling:

- `scaling = "data"` ... the min and the max over all the data map to 0 and 1 for all axes (i.e. range over all rows and columns of data)
- `scaling = "variable"` ... the min and the max for each variable map to 0 and 1 for the corresponding axis (i.e. range over each column of data)
- `scaling = "observation"` ... the min and the max over all variables for each observation map that observation's values to 0 and 1 on all axes (i.e. range over each row of the data)

From the serial axes inspector, explore how each of these scalings (data, variable, observation) affect the serial axes plot **and its interpretation**.

Do the same but, instead of parallel axes, change to a radial axes system.



Serial axes - as glyphs

To see whether the distribution of visible minorities among cities might have some spatial explanation, we can add them as glyphs to our map.

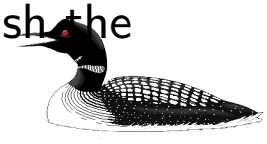
```
s_glyphs <- l_glyph_add_serialaxes(cities_map,
                                   data = minority_only,
                                   sequence = names(minority_only),
                                   scaling = "variable",
                                   showArea = TRUE,
                                   label = "serial_axes")
cities_map['glyph'] <- s_glyphs
```

Zoom and pan the map to explore spatial patterns.

Go to the inspector for the map.

- select the “Glyphs” tab
- select the glyph labelled “serial_axes”
- change the scaling to “observation” with layout “radial”

Have the map and the original serial axes both displayed on your screen with the same scaling and axes layout. Now pan and zoom over the map and brush the points.

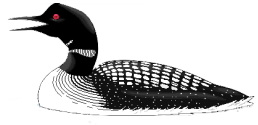


Ad hoc glyphs - `l_make_glyphs()`

Sometimes, it is handy to construct your own glyphs. In `loon` any plot that can be drawn in `R` can appear as an `image` glyph in `loon`.

```
barplot_imgs <- l_make_glyphs(data = lapply(1:nrow(minority_only),
                                            FUN = function(i) minority_only[i,]),
                              # draw a coloured bar plot
                              draw_fun = function(m) {
                                  par(mar=c(1,1,1,1)*.5)
                                  mat <- as.numeric(m/max(m))
                                  barplot(height = mat,
                                          beside = FALSE,
                                          ylim = c(0,1),
                                          col = rainbow(length(mat)),
                                          axes= FALSE,
                                          axisnames=FALSE)},
                              width=120,
                              height=120)
# Can view the images using loon's viewer
img_vwr <- l_imageviewer(barplot_imgs)
# add the glyphs
barplot_glyphs <- l_glyph_add_image(cities_map, barplot_imgs, "barplot")
cities_map['glyph'] <- barplot_glyphs
```

These images contain the same information as the radial images with "observation" scaling.

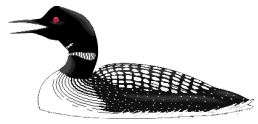


A higher dimensional example - `frey` faces

In the `frey` dataset to be found in the `RnavGraphImageData` package, we have 1,965 images of a young Brendan Frey's face. For example:

- Each pixel has a grey value in [0,1].
- Each image is a 28×20 pixel image.

So each image is a point in 560 dimensions.



A higher dimensional example - `frey` faces

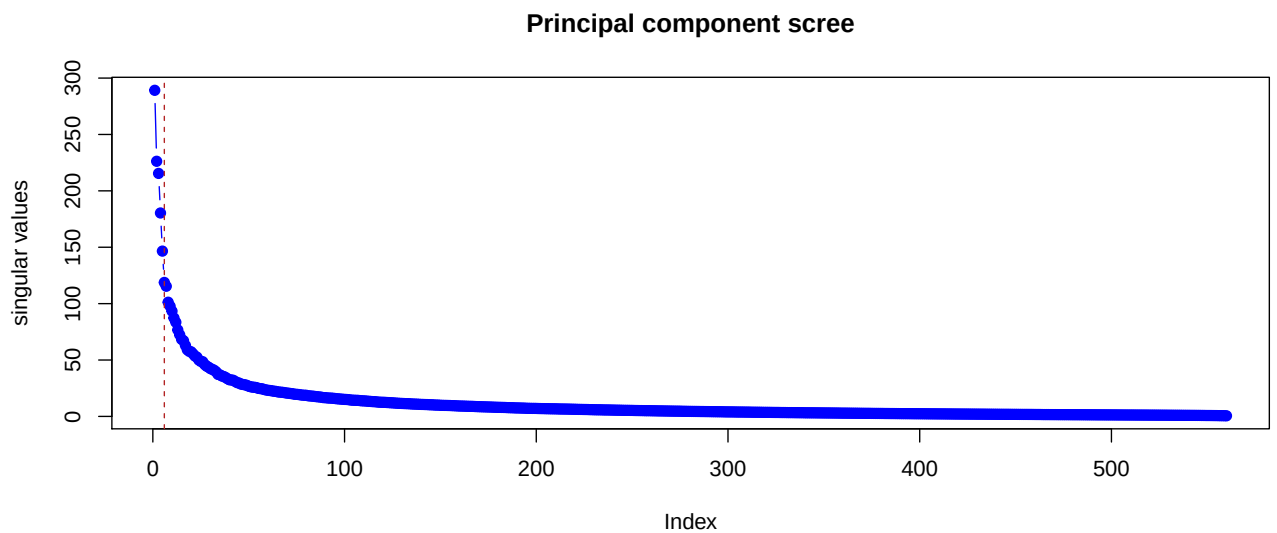
The entire data set is had and viewed as follows:

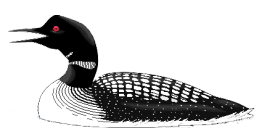
```
library(RnavGraphImageData)
data(frey)
# Here are the images
frey.imgs <- l_image_import_array(frey, 28,20,
                                  img_in_row = FALSE, rotate = 90)
frey_imgs_vwr <- l_imageviewer(frey.imgs)
```

The question is whether we can position these in a smaller dimensional space ($ << 560$) which might then be explored.

A standard approach is to use principal component analysis:

```
data <- t(frey)  # images were in columns
# get the principal components
frey_pc <- prcomp(data)
```

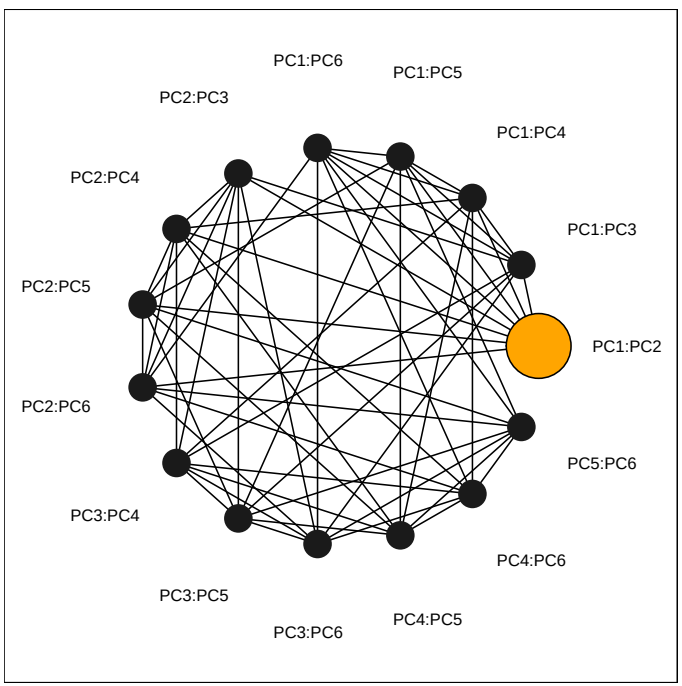




Principal components - `frey` faces

Let's just look at the first 6 principal components. In 'loon' a 'navgraph' can be used to explore the 6-dimensional space:

```
frey_pc_coords <- frey_pc$x[, 1:6]
# create a navigation graph
frey_pc_nav  <- l_navgraph(frey_pc_coords, linkingGroup="frey")
frey_glyphs  <- l_glyph_add_image(frey_pc_nav$plot,
                                  images=frey.imgs,
                                  label="frey faces")
```



A 'navgraph' is an interactive graph whose nodes are pairs of variates (e.g. scatterplots) and whose edges represent a 3 dimensional space (if the nodes share a variate) and a 4 dimensional space (if the nodes do not). A "navigator" is moved along the graph to effect a change in the scatterplot.

Select the "navigator" and move it about. Holding the <Shift> key down shows the next available nodes.

Local linear embedding - `frey` faces

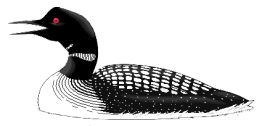
Local linear embedding (or LLE) is just one dimension reduction method which tries to preserve local structure at the cost of global structure.

It also has the strange side effect of producing lines which makes it useful when looking at projections (since lines are preserved).

```
library(dimRed) # dimension reduction package
# Number of neighbours
k_nbhrs <- 12
# Target dimension
n_dims <- 20
# Dimension reduction via LLE ... this may take a while
lle20 <-  embed(data, "LLE", knn = k_nbhrs, ndim = n_dims)
lle20data <- lle20@data@data
l_serialaxes(lle20data, linkingGroup = "frey")
```

Reducing to 20 dimensions means $\binom{20}{2}$ or 190 possible scatterplots to examine.

In 'loon' we can take advantage of measures of interestingness of each plot (e.g. via `scagnostics`) to narrow down the set of plots we might examine.



Finding interesting scatterplots - `l_ng_plots()` and `scagnostics2d()`

Scagnostics are "scatterplot diagnostics" first proposed by John and Paul Tukey in 1985. A graph-theoretic implementation is available from the `scagnostics` package.

```
# Get a set of "closures" which will calculate scagnostics.
lle_scags2d <- scagnostics2d(lle20data)
# Now a scatterplot matrix of THE PLOTS
lle_nav <- l_ng_plots(measures=lle_scags2d, linkingGroup="frey", size = 1)

lle_gl <- l_glyph_add_image(lle_nav$plot,
                            images=frey.imgs,
                            label="frey faces")

lle_nav$plot["linkingGroup"] <- "frey"
lle_nav$plots$x2y1["size"] <- 1
```

Explore the data using these tools.

Note that

- measures can be recalculated on subsets of the data
- you can produce your own measures (see `l_help()` section on graphs)