Tooltips in ggplot

« February 18, 2014 | r »

R has few adequate facilities for interactive visualization. This table presents an overview of many of them. Joe Fruehwald has had some success with using googleVis to display diachronic data on sound change in Philadelphia. However, in general interactive graphs are not widely used in linguistics. This post presents a case study of a certain kind of interactive graphics, with accompanying source code.

Updates to this post:

- Feb. 18, 2014: clarify Flash problem
- Feb. 18, 2014: fix handling of duplicated data rows

Introduction

We'd like to examine Hilary Prichard's data on the Great Vowel Shift in northern England. Specifically, we'd like to look at the Middle English /u/, and whether it has lowered to /au/. In doing so, we encounter tension between two forces. The first is to represent the broad pattern as faithfully as possible. The second is to not lose detail in our visualization. We'll tackle the problem by using tooltips.

For example, Joe Fruehwald's page linked above may or may not be working; I could not get the graphs to load on my computer. This presents perhaps an unintentional

There are a couple desiderata for this endeavor. First of all, we want the graphic to be as high-quality as possible. Secondly, we want to keep the output to a single file. High-quality interactive visualizations can be made with the aid of a web server or running R process, but we want our solution to be downloadable by anyone and openable without installing or running additional software, or connecting to the internet. These considerations mean that we will use the SVG format. This is a graphics format which opens and displays in all modern web browsers. It is indefinitely zoomable without loss of resolution. It also can be easily converted to a PDF for use in publication (the interactivity, however, is lost).

lesson in the perils of making complicated visualizations stably available! **Update:** this turned out to be a problem with Flash on my computer.

PreliminaRies

We need to load some packages:

```
library(mapdata)
library(ggplot2)
library(dplyr)
library(stringr)
library(proto)
library(gridSVG)
library(ggmap)
library(grid)
library(RColorBrewer)
```

If you get errors about packages not being found, use `install.packages` to install them, then try again.

Let’s peek at the data, which is available from GitHub:

```
u.data <- read.csv("HOUSE.csv")
u.data[50:55,]
```

TownCode	Town	Latitude	Longitude	How	House	Clouds	About	Dro
5.1	Coniston	54.368	-3.073	aʊ	aʊ	aʊ	aʊ	aʊ
5.2	Cartmel	54.199	-2.951	aʊ	aʊ	aʊ	aʊ	ʊ
5.2	Cartmel	54.199	-2.951	aʊ	aʊ	aʊ	aʊ	ʊ
5.3	Yealand	54.17555	-2.76596	aʊ	aʊ	aʊ	aʊ	ʊ
5.3	Yealand	54.17555	-2.76596	aʊ	uː	aʊ	aʊ	ʊ
5.4	Dolphinholme	53.975	-2.738	aʊ	aʊ	aʊ	aʊ	ʊ

Note that some sample locations are represented with two, possibly diverging, rows. These rows don’t represent duplicate samples. Rather, they exist to represent variation in the data. So, in Yealand “house” can be pronounced as either /haʊs/ or /huːs/, whereas the other four words are pronounced uniformly. The individual vowels are

Update: The discussion of the coding of variation in the data was added after consulting with Hilary. The map was also changed to represent the data as variation, and not (incorrectly) as multiple samples.

very closely transcribed. We should set up a variable which contains all the non-lowered variants of the vowel, to aid in assigning them to the binary categories of (not) lowered:

```
conservative.vowels <- c("ᵊuː","ᵓuː","ᵘuː","uː'
                         "ʊ") ## upsilon only i
```

Be careful: the colon-resembling things are not a colon, but rather the IPA length diacritic. Don’t try to retype this code yourself, but rather copy it into your R session, or download the script from Github.

We’ll also set up some functions which translate from one row of the data frame to a string indicating which vowels are not lowered. We’ll use HTML formatting, since that’s what our tooltips will allow.

```
singleSummary <- function(vowels, word) {
    s <- vowels %in% conservative.vowels
    if (length(s) > 1 && !all(s) && any(s)) {
        return (paste0(word, " (varies)"))
    } else if (all(s)) {
        return (word)
    } else {
        return (NA)
    }
}
summaryString <- function(house, how, about, cl
    r <- ""
    x <- singleSummary(house, "House")
    if (!is.na(x)) r <- paste0(r, x, "<br />")

    x <- singleSummary(how, "How")
    if (!is.na(x)) r <- paste0(r, x, "<br />")

    x <- singleSummary(about, "About")
    if (!is.na(x)) r <- paste0(r, x, "<br />")

    x <- singleSummary(clouds, "Clouds")
    if (!is.na(x)) r <- paste0(r, x, "<br />")

    x <- singleSummary(drought, "Drought")
    if (!is.na(x)) r <- paste0(r, x, "<br />")

    ## Strip a trailing <br />
    r <- str_sub(r, end = -7)
    r <- ifelse(r == "", "(none)", r)
    return (r)
}
```

Reformatting the data

The first step of making a plot is almost always to reformat the data. I’ll be using the dplyr package for this. It’s a relatively new package, written by Hadley Wickham, who also wrote ggplot. Like ggplot, the learning curve is steep, but once you master the system it’s very intuitive and powerful. I won’t dwell on this code; ask me if you’d like to learn more or use this R command to read the introduction: `vignette("introduction", package = "dplyr")`.

```
u.data <- tbl_df(u.data)
u.plot.data <- u.data %.%
    group_by(Town, Latitude, Longitude) %.%
    summarise(NUnLowered =
              sum(any(House %in% conservative.v
                  any(How %in% conservative.vow
                  any(About %in% conservative.v
                  any(Clouds %in% conservative.
                  any(Drought %in% conservative
              UnLoweredWds = summaryString(
                 House, How, About, Clouds, Dro
              N = n())
u.plot.data$N <- u.plot.data$N * 5
head(as.data.frame(u.plot.data[20:25,]))
```

Note that this code counts variable tokens as instances of the conservative (unlowered) variant for calculating the number of unlowered tokens per locality. Other alternatives would be to not count them (i.e. treat them as innovative), or count them with half the weight of an unvarying token.

We also need to read in some boundary data; this luckily is very easy:

```
mapdata <- map_data("worldHires", "UK")
```

Tooltips

We also need a function to enable tooltips on our map. I've set this up so that it fits in to the usual ggplot syntax as closely as possible. Instead of using (for this example) `geom_point`, we instead need to call `geom_tooltip`. And we need to add a `tooltip` key to our `aes`, which specifies, in HTML format, the tooltip we want. We also add a `real.geom` argument which indicates the ggplot geom we wish to decorate with the tooltip.

HTML wizards: this will be displayed inside a `span` element, so you can't use block-level HTML formatting. Bold, underline, and italics are fine.

The function to do this packs a lot of punch. The rest of this post will be dedicated to explaining it carefully, but if you're just here for the cool map, you should skip to the next section.

```
geom_tooltip <- function (mapping = NULL, data
                          position = "identity'
    rg <- real.geom(mapping = mapping, data = c
                    position = position, ...)

    rg$geom <- proto(rg$geom, { ## ②
        draw <- function(., data, ...) {
            grobs <- list()
            for (i in 1:nrow(data)) {
                grob <- .super$draw(., data[i,]
                grobs[[i]] <- garnishGrob(grob,
                                          `data
            }
            ggplot2:::ggname("geom_tooltip", gT
                children = do.call("gList", gro
                ))
        }
        required_aes <- c("tooltip", .super$rec
    })

    rg ## ⑤
}
```

We begin at ① by calling the `real.geom`. This function returns a ggplot layer, which has a geom as well as several other properties. At ②, we replace the geom slot of that object with something new. Ggplot internally uses a protypal object orientation package (similar to the OO in Javascript). This allows us to patch the `draw` method while leaving the rest of the object relatively undisturbed. At ③, we must loop through each data point and draw it separately. Otherwise, we would get the tooltip assigned to groups of points (which share ggplot's `group` attribute), rather than the desired individual assignment. At ④, we use a function from the `gridSVG` package to add an attribute, which will be translated into svg and picked up by our tooltip javascript. (It has no effect on the native R graph.) We add `tooltip` to the required aesthetics, and finally at ⑤ we return the modified object.

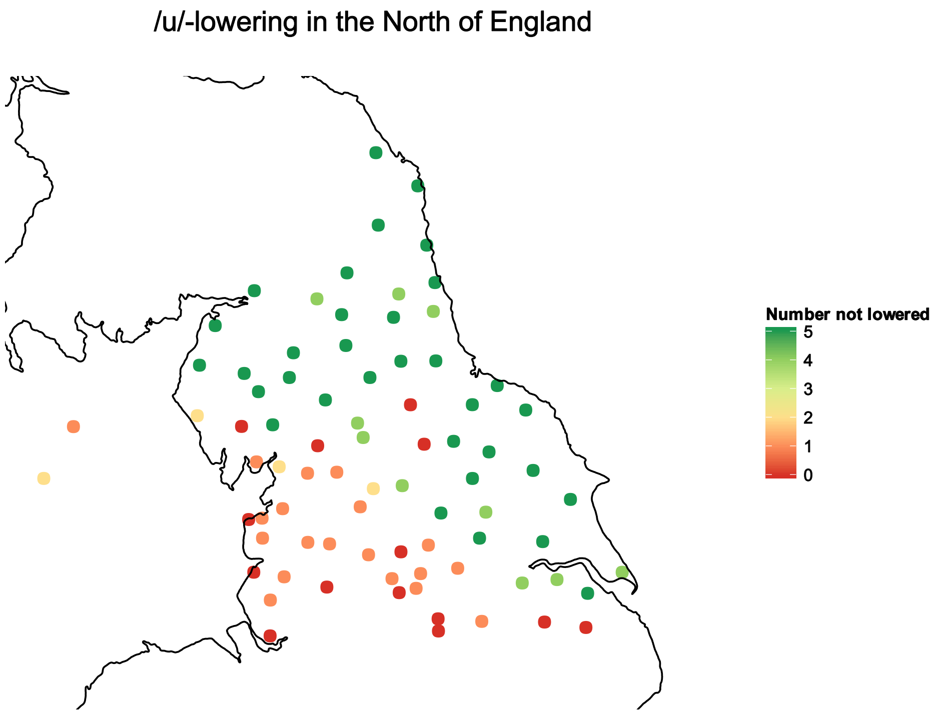
We also need some javascript code to animate the tooltips. The first code block was taken from Simon Potter's website, and modified by me. A version can also be found in the gridDebug library. It is presumably GPL2+. It's long and not of general interest, so I won't include it here. I think it's a pretty good example of the tricks that are needed to embed HTML in SVG programmatically, though, so if that's your cup of tea you should take a gander.

Building the map

OK, we are ready to build the map. We'll use the `gridsvg` function. Everything between that and the corresponding `dev.off` call will be exported to the SVG file. There are a few tricks here, but nothing other than the tooltip business is any different than how we would ordinarily build this map in ggplot.

```
gridsvg(svgDest,exportJS="inline",addClasses=TR
ggplot(u.plot.data, aes(x = Longitude, y = Lati
    geom_tooltip(aes(tooltip = paste0(
                         "<b>Words not lowered
                         ":</b><br />",
                         UnLoweredWds),
                     color = NUnLowered,
                     size = 3),
                 real.geom = geom_point) +
    geom_polygon(aes(x = long, y = lat, group =
                 color = "black", fill = NA) +
    coord_map(xlim=c(-5,1), ylim=c(53,56)) +
    theme_nothing(legend = TRUE) +
    theme(plot.title = element_text(size = 16,
          plot.margin = unit(c(0.5, 0.1, 0.1, 0
    ggtitle("/u/-lowering in the North of Engla
    scale_color_gradientn("Number not lowered",
                          colours=brewer.pal(6,
    scale_size_continuous(guide = FALSE)
grid.script(jscript)
dev.off()
```

We've assigned the javascript code above to the `jscript` variable; it will be included in the generated SVG file. Et voilà:



The color indicates the overall distribution of lowering (red) and conservative non-lowered variants (green). Hovering your mouse over a point will show you which words were not lowered by speakers at that location.

You can download the SVG file yourself. All the code from this post is available on GitHub, as is the data. Enjoy!

— Aaron Ecay

Comments Community 1 Login

Recommend Sort by Oldest

Start the discussion…

Be the first to comment.