Display San Francisco Cab Data

Table of Contents

Introduction

This an exploration in using the Google Maps API by generating HTML & JavaScript code in R that is then displayed in a Web browser. We will use the data from http://cabspotting.org. (Note we can collect our own in real time.)

We'll grow this example incrementally.

1. We start by displaying a single cab and showing its path. cab1.html

2. Next we add markers on the map that shows where a passenger was picked up and another marker type for where a passenger was let off. We add HTML content to the marker so that when the viewer clicks on the marker it is displayed in a popup window on the map. The information gives information about what type of event this was (drop-off or pickup), the date and time of the event, the passenger number (in sequence) and the total number of passengers in this period, and the duration of the ride. cab3.html

3. Next we'll break the path into different sub-paths corresponding to separate driver shifts or different passengers. cab4.html

4. We move then to providing a choice menu or a selection list that allows the viewer to select which cab(s) to display and then we display those polylines. See cab5.html & cabShiftToggle.html

5. We'll use polyline encoding for efficiency and greater control of appearance when zooming.

6. We also show how we can keep the data in an XML format and separate from the JavaScript code. The JavaScript reads that at run-time and generates the objects. See data.html

A Single Cab

We start with a single cab.

```
                                                                           R
f = "new_enyenewl.txt"
a = read.table(f, header = FALSE, col.names = c("lat", "long", "occupied", "time")
                                         colClasses = c("numeric", "numeric", "int
a[[3]] = as.logical(a[[3]])
class(a[[4]]) = c("POSIXt", "POSIXct")
```

We'll start by creating the JavaScript code that creates the polyline. We do this by creating an array of GLatLng objects.

```
                                                                           R
tmp = sprintf("new GLatLng(%.4f, %.4f)", a$lat, a$long)
```

```
                                                                           R
cc = paste("new GPolyline([", paste(tmp, collapse = ",\n\t"),
                         "], ",
                        dQuote("#FF0000"), ",", 2,
                        ")", sep = "\n")
```

Now we have to center the map at the "center" of the path.

```
                                                                           R
zoom = 11
sprintf("map.setCenter(new GLatLng(%.4f, %.4f), %d)",
            mean(range(a$lat)), mean(range(a$long)), zoom)
```

Adding markers

The idea is that we will determine where the cab picked up or dropped off a passenger. Let's reverse the order of the rows in our cab data frame. This will allow us to think of time as increasing with row.

```
                                                                           R
b = a[nrow(a):1,]
```

Now we find the rows when the occupied status changes

```
                                                                           R
w = which(diff(b$occ) != 0)
```

If we add 1 to this, we have the corresponding row in **b**. So now we have the location for the marker. We also need to know if each of these is occupied or unoccupied. This depends on the starting value.

```
                                                                           R
labels = c("occupied", "unoccupied")
if(b[1, "occupied"])
 labels = rev(labels)
status = factor(labels[rep(c(1,2), length = length(w))],  levels = labels)
```

So now we can add the markers

```
                                                                           R
```

```
tmp = sprintf("\tmap.addOverlay(new GMarker(new GLatLng(%.4f, %.4f)));", b[w, "lat
paste(tmp, collapse = "\n")
```

The above produces markers for each event. But we want to color code them as being a drop-off or pick-up. Also, we want to allow the viewer to click on the marker and get information about the particular event.

Markers with Event Information

R

```
icons = c("occupied" = "http://gmaps-samples.googlecode.com/svn/trunk/markers/red/
          "unoccupied" = "http://gmaps-samples.googlecode.com/svn/trunk/markers/gr

html = sprintf("Cab: <a href='http://cabspotting.org/cab.xml.php?cab=%s&m=45'>%s</
                 names(cabCounts)[1], names(cabCounts)[1],
                 seq(along = w),
                 length(w),
                 c("occupied" = "drop-off", unoccupied = "pick-up")[as.character(s
                 as.character(a$time)[w + 1]
                )

tmp = sprintf('\tmap.addOverlay(createMarker(new GLatLng(%.4f, %.4f), "%s", "%s"))
                   b[w, "lat"], b[w, "long"], html, icons[as.character(status)])
cat(paste(tmp, collapse = "\n"))
```

Using XML

A different way to present the markers (without color) for the location of the drop-offs and pickups is to use generic JavaScript code and specify the marker location separately using an XML document. Building on what we have from the previous section (i.e. the row number in **b** for the pick-up and drop-off events)

R

```
m = newXMLNode("markers")
invisible(sapply(w, function(i, p)
                      newXMLNode("marker",
                                  attrs = c(lat = b[i, "lat"], lng = b[i,"long"]),
                                  parent = p),
                  m))
saveXML(m, "~/Books/XMLTechnologies/Rpackages/R2GoogleMaps/inst/sampleDocs/markers
```

Then we can open data.html. This has been partially modified (from the original version downloaded from the Google Map examples) to center the map on the region of interest and specify a different data file. We would do this programmatically in R with

R

```
doc = htmlParse("~/Books/XMLTechnologies/Rpackages/R2GoogleMaps/inst/sampleDocs/da
body = getNodeSet(doc, "//body")[[1]]
xmlAttrs(body) = c(onload = sprintf("initialize('markers.xml', %.4f, %.4f)",
                                    mean(range(b$lat)), mean(range(b$long))))
```

By day

We can display the path the cab took broken up by day/shift. We find the shifts by finding the observations in the data where there is a gap of an hour or more. We can create a new variable which identifies the shift number with the following code:

R

```
i = diff(b$time) > 60^2
shift = c(0, cumsum(i))
```

Now we can operate on each shift using *by*() and create the code that creates the poly lines for that, e.g.

R

```
colors = substring(rainbow(length(unique(shift))), 1, 7)
k = by(cbind(b, color = colors[shift + 1]), shift, makePolyline)
```

We want the value of the **shift** in each group to select the color.

We can define *makePolyline*() something like the following based on what we did for the entire day.

R

```
makePolyline =
function(data, var = character())
{
  color = as.character(data$color)[1]
  tmp = sprintf("new GLatLng(%.4f, %.4f)", data$lat, data$long)

  tmp = paste("new GPolyline([", paste(tmp, collapse = ",\n\t"),
                                     "], ",
                                       dQuote(color), ",", 2,
                        ")", sep = "\n")

  if(length(var))
    tmp = paste(var, tmp, sep = " = ")
  sprintf("map.addOverlay(%s);", tmp)
}
```

Selecting Driver Shifts within a Cab or Different Cabs

Here we look at how we can allow the viewer to control what is displayed in the view. We'll break a single cab's data into different shifts as we did before.

R

```
d = readCabTrace("/Users/duncan/Data/cabspottingdata/new_ugthfu.txt")
i = diff(d$time) > 60^2
d$shift = c(0, cumsum(i))
d$colors = as.character(substring(rainbow(length(unique(d$shift))), 1, 7))[d$shift
```

Now we generate the code. We assign each GPolyline to an element of a JavaScript array which we will call polylines.

R
```
k = by(d, d$shift, function(x) makePolyline(x, paste("polylines[", x$shift[1], "]"
```

We need to add this code to the initialize and also to define the polylines variable. We write the code in **k** to a function drawPaths in a separate file so that we can easily include it in our HTML document. We then change the initialize function to call this, passing it the GMap2 object.

R
```
cat("function drawPaths(map)", "{", k,  "}", sep = "\n", file = "drawPaths.js")
```

Now we can add the form and checkboxes to the HTML document. We add one for each shift and we have its onMouseUp method call the JavaScript function toggle, passing it the relevant overlay, i.e. element of polylines (remembering we use 0-based counting JavaScript), and whether to show or hide the overlay.

R
```
form = newXMLNode("form")
dl = newXMLNode("dl", parent = form)
invisible(
sapply(unique(d$shift),
        function(i, p)
          newXMLNode("dt", newXMLNode("input", paste("Shift", i), attrs = c(type =
                                              checked = "1",
                                              onMouseUp = sprintf("toggleOverlay(polylin
                      parent = p),
       dl))
```
We should also add a reset button to the form.

R
```
newXMLNode("input", attrs = c(type="button", value="Reset", onclick="reset()"), pa
```

Now we add this to the HTML document along with references to the JavaScript files drawPath.js and toggle.js.

The result is cabShiftToggle.html

High-level Functions

In the examples above, we have glossed over how we add the content to the HTML documents. We now work through some of these examples using high-level functions in R that generate the code from R data objects and construct the HTML file and its contents to display the map.

The Simple Polyline

R
```
center = c(mean(range(b$lat)), mean(range(b$long)))
code = addOverlay(gpolyline(b))
d = googleMapsDoc(code, center, zoom = 11, dim = c(750, 700), file = "simplePolyli
```

Markers

R

```
w = which(diff(b$occ) != 0)
labels = c("occupied", "unoccupied")
if(b[1, "occupied"])
 labels = rev(labels)
status = factor(labels[rep(c(1,2), length = length(w))],  levels = labels)
```

R
```
code = gmarker(b[w, "lat"], b[w, "long"], addOverlay = TRUE)
d = googleMapsDoc(code, center, zoom = 11, dim = c(750, 700), file = "simplePolyli
```

Markers with Information

The R2GoogleMaps package doesn't do much to help here with generating the code as it is quite customized, using a a different JavaScript function to create the marker that has icons and such. The package does help in creating the document and bringing in the necessary JavaScript code.

As we did above, we need to compute some variables that go into the content, specifically the HTML for each marker.

R
```
w = which(diff(b$occ) != 0)

icons = c("occupied" = "http://gmaps-samples.googlecode.com/svn/trunk/markers/red/
          "unoccupied" = "http://gmaps-samples.googlecode.com/svn/trunk/markers/gr

labels = c("occupied", "unoccupied")
if(b[1, "occupied"])
 labels = rev(labels)
status = factor(labels[rep(c(1,2), length = length(w))],  levels = labels)

html = sprintf("Cab: <a href='http://cabspotting.org/cab.xml.php?cab=%s&m=45'>%s</
                 "oilrag", "oilrag",
                 seq(along = w),
                 length(w),
                 c("occupied" = "drop-off", unoccupied = "pick-up")[as.character(s
                 as.character(b$time)[w + 1]
                )
```

Now we can create the HTML document and the map and specify the extra JavaScript file to be included.

R
```
code = sprintf('\tmap.addOverlay(createMarker(new GLatLng(%.4f, %.4f), "%s", "%s")
                   b[w, "lat"], b[w, "long"], html, icons[as.character(status)])
d = googleMapsDoc(code, c(mean(range(b$lat)), mean(range(b$lon))),
                   zoom = 11, file = "infoMarkers.html", scripts = "../javascript/
```

Using XML

The function *markerData*() creates data in the format we want for specifying marker locations via XML. We use this as

R

```
markerData(b[w,], file = "markers.xml")
```

Now we have to use this data in an HTML document that creates a map and displays the markers. We need to include the genericMarkerData.js script in our file as that knows how to read our XML data. It provides an initialize function that will create the map and the markers, and it also provides a separate function (makeMarkers) that we can call to create the markers ourselves.

R
```
googleMapsDoc('makeMarkers("markers.xml", map);',
               c(mean(range(b$lat)), mean(range(b$lon))),
              zoom = 12,
              file = "mydata.html", scripts = "../javascript/genericMarkerData.js"
```

If we wanted to use the initialize function from genericMarkerData.js and let it add the controls, specify the zoom, etc. we can. Instead of providing our own code to `googleMapsDoc()` , we want to change the onload code that is evaluated when the document is loaded. We want to avoid building our own initialize function, so we specify an empty string for the code we provide and make it an *`AsIs`* object to say "treat this as the entire function". So our call is

R
```
googleMapsDoc(I(''),
              onload = sprintf('initialize("markers.xml", %.4f, %.4f)', mean(range
              zoom = 12,
              file = "mydata1.html",
              scripts = "../javascript/genericMarkerData.js"
            )
```

and we get the same result, except we have different controls and zoom levels that are hard-coded into the initialize function in genericMarkerData.js. We could of course make that function richer and have it accept optional parameters.

By day

Real-time Data

The cabspotting site has a real time feed and API. We can find out which cabs have data available within the last m minutes.

R
```
http://cabspotting.org/cab.xml.php?cab=ugthfu&m=480
```

The following functions perform the download and format the data.

R
```
library(XML)
library(RCurl)

getCabs =
function(m = 10)
{
  x = getForm('http://cabspotting.org/cabs.xml.php',  m = m)
  doc = xmlParse(x, asText = TRUE)
  structure(as.integer(xmlSApply(xmlRoot(doc), xmlGetAttr, "updates")),
            names = xmlSApply(xmlRoot(doc), xmlGetAttr, "id"))
}

getCabInfo =
function(m = 60, cabs = names(getCabs(m)), combine = TRUE)
{
  ans = lapply(cabs, getOneCab, m)

  if(combine)
    do.call("rbind", ans)
  else {
    names(ans) = cabs
    ans
  }
}

getOneCab =
function(id, m = 60)
{
  ans = getForm("http://cabspotting.org/cab.xml.php", cab = id, m = m)
  doc = xmlParse(ans)
  tmp = xmlSApply(xmlRoot(doc), xmlAttrs)
  tmp = as.data.frame(t(tmp), row.names = 1:ncol(tmp))
  names(tmp) = c("cab", "lat", "long", "status", "time")
  tmp$time = as.POSIXct(as.numeric(as.character(tmp$time)), origin = "1970-01-01")
  tmp
}
```