Spatial Statistics for Modeling Safety

Introduction

This document gives an example for pulling data from Open Street Map, building features, incorporating survey data, and training a model to predict safety using these GPS-built features. An excellent tutorial for the R package osmar can be found at `https://journal.r-project.org/archive/2013/RJ-2013-005/RJ-2013-005.pdf`; this work builds on their code.

```
# load packages
library(osmar) # pull data from OSM
library(sp) # manage OSM data
library(lme4) # train generalized linear model

# set OSM source
src = osmsource_api(url = "https://api.openstreetmap.org/api/0.6/")
```

Data Generation

To protect the privacy of our survey participants in Nairobi, we demonstrate this method using a set of locations from Stanford University. Researchers who want to input their own locations should search and then pull the node IDs and latitude/longitude information from `https://www.openstreetmap.org/`.

```
# OSM Data for Stanford locations (OSM node ID, lat, lon)
data = data.frame(rbind(c(6394391907,  37.4316091, -122.1636112),
                        c(4297881424, 37.4278318, -122.1671190),
                        c(1434767635, 37.4342028, -122.1620981),
                        c(287483377, 37.4268792, -122.1703894),
                        c(2411555515, 37.4289640, -122.1725380)))
colnames(data) = c("loc.id", "lat", "lon")
rownames(data) = c("Alumni center", "Hoover tower", "Stanford stadium", "Memorial church",
                   "Sequoia hall")

# Load generated survey responses (make sure to set working directory with setwd())
survey_data = read.csv("survey_data.csv")
```

Feature Construction

In this section, we use OSM to build features for each location. We give a few examples of features; researchers could choose any features they want and any way to measure them (the number of street lamps within 50 meters, distance to the nearest street lamp, indicator for whether a street lamp is within 50 meters, etc.).

Understanding OSM Data

To pull data from OSM, we give a bounding box, saying we want all locations tagged within that bounding box. We can give either box coordinates (left, bottom, right, top), with corner_bbox, or the coordinates (longitude, latitude) for the center of the box, along with desired width and height, using center_bbox. Data pulls begin by creating the bounding box, and then pulling all data inside of it. We will use Stanford's alumni center as an example.

```
bb = center_bbox(data[1, ]$lon, data[1, ]$lat, 500, 500)
ua = get_osm(bb, source = src)
```

Let's begin by looking at a summary of the data contained here. There are nodes (bus stops, unique locations - essentially any point) and ways (rivers, streets, outlines of buildings - think lines that connect points).

```
summary(ua$nodes)
```

```
osmar$nodes object
1217 nodes, 298 tags

..$attrs data.frame:
    id, visible, timestamp, version, changeset, user, uid, lat, lon
..$tags data.frame:
    id, k, v

Bounding box:
         lat       lon
min 37.42690 -122.1728
max 37.44339 -122.1588

Key-Value contingency table:
            Key                 Value Freq
1       highway              crossing   40
2      crossing                marked   21
3       highway                  stop   21
4          kerb               lowered   16
5      crossing          uncontrolled   16
6  traffic_sign                  stop   14
7       barrier                  gate    9
8     direction               forward    8
9      operator Stanford University    8
10          bus                   yes    8
```

```
summary(ua$ways)
```

```
osmar$ways object
177 ways, 762 tags, 1493 refs

..$attrs data.frame:
    id, visible, timestamp, version, changeset, user, uid
..$tags data.frame:
    id, k, v
..$refs data.frame:
    id, ref

Key-Value contingency table:
            Key           Value Freq
```

```
1       highway         footway   71
2  tiger:county Santa Clara, CA   33
3       highway         service   29
4       footway        sidewalk   27
5      building             yes   23
6      maxspeed          25 mph   21
7    tiger:cfcc             A41   21
8        oneway             yes   21
9       bicycle             yes   20
10      highway        tertiary   19
```

Observe that each node is tagged with a key and value. This is what lets us find different categories of locations. Suppose, for example, that we want to know how many bus stops are near (say, within 100 meters of) the alumni center.

```
# find all bus stop IDs
ids = find(ua, node(tags(k == "bus")))
```

That step can take some work; you'll need to look at the data and see how items were actually tagged. Sometimes keys will be most helpful, and sometimes values. Regular expressions can come in handy here as well. We can take a look at the relevant nodes here:

```
# subset our data down to just those IDs
elements = subset(ua, node_ids = ids)
head(elements$nodes$tags)
```

```
            id       k                      v
122 5686257463   bench                    yes
123 5686257463     bin                    yes
124 5686257463     bus                    yes
125 5686257463 highway               bus_stop
126 5686257463    name Stanford Visitor Center
127 5686257463 network             Marguerite
```

We can see here how the bus stops are coded. Also note that some IDs (which correspond to a specific GPS location) have multiple tags; bus stops have benches, and are tagged with the network (Marguerite shuttle) and bus operator (Stanford University). Conveniently, when we look for latitude and longitude of the points, osmar only returns results for the number of specific geographic locations.

```
# Find longitudes and latitutes
lons = elements$nodes$attrs$lon
lats = elements$nodes$attrs$lat

cat(paste("Sanity check: OSM has", length(elements$nodes$tags$id), "unique ID tags but",
          length(lons), "unique locations."))
```

```
Sanity check: OSM has 54 unique ID tags but 8 unique locations.
```

Now we can find all nearby bus stops.

```
# Use "distm" function to get distance in meters
distances = distm(cbind(lons, lats), c(data[1,]$lon, data[1,]$lat))
val = sum(distances < 100)
cat(paste("There are", val, "bus stops within 100 meters of the alumni center."))
```

```
There are 2 bus stops within 100 meters of the alumni center.
```

We can also use OSM data for some nice, quick visualizations. The sp package is extremely helpful here. We do this by casting nearby objects as polygons for easy plotting; the tutorial linked at the top of this document details many options for plotting as well.

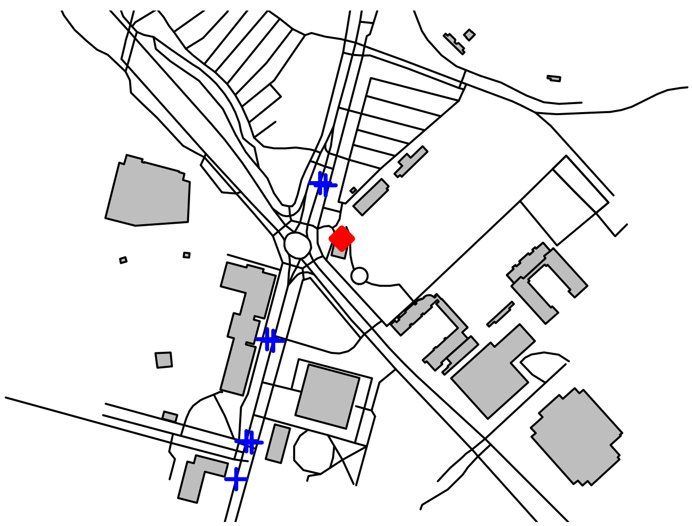
```
# find buildings
bg_ids = find(ua, way(tags(k == "building")))
bg_ids = find_down(ua, way(bg_ids))
bg_poly = as_sp(subset(ua, ids = bg_ids), "polygons")

# find streets
street_ids = find(ua, way(tags(k == "highway")))
street_ids = find_down(ua, way(street_ids))
street_lines = as_sp(subset(ua, ids = street_ids), "lines")

# create sp object for alumni center
alumni_center = subset(ua, node_ids = data[1,]$loc.id)
alumni_sp = as_sp(alumni_center, "points")

# create sp object for bus stops
bus_sp = as_sp(elements, "points")

plot(bg_poly, col = "gray")
plot(street_lines, add = TRUE, col = "black")
plot(bus_sp, add = TRUE, col = "blue", lwd = 2, pch = 3)
plot(alumni_sp, add = TRUE, col = "red", lwd = 3, pch = 9)
```



Building covariates

For this example, our features are of the form "number of X within 400 meters." So, the following function finds all OSM elements tagged with a certain keywords, within a maximum distance.

```
# function that finds locations matching a set of keywords within a max allowed distance
nearby.elements = function(ua, keywords, point, max.distance = 400){
  ids = find(ua, node(tags(v %in% keywords)))
  if(is.na(sum(ids))){
    val = 0
  } else{
    elements = subset(ua, node_ids = ids)
    lons = elements$nodes$attrs$lon
```

```
    lats = elements$nodes$attrs$lat
    distances = distm(cbind(lons, lats), point)
    val = sum(distances < max.distance)
  }
  return(val)
}
```

The main function here takes a bounding box and location id, and returns for that location the number of nearby trees, water fountains, bicycle parking spots, and bus stops.

```
build.features = function(bb, loc.id){
  # get data for the area from OSM
  ua = get_osm(bb, source = src)

  # get current location as a point
  current.loc = subset(ua, node_ids = find(ua, node(attrs(id == loc.id))))
  current.loc = as_sp(current.loc, "points")
  current.lat = current.loc$lat
  current.lon = current.loc$lon
  point = c(current.lon, current.lat)

  # number of trees within 400 meters
  trees = nearby.elements(ua, "tree", point)

  water = nearby.elements(ua, c("drinking_water"), point)
  bike.parking = nearby.elements(ua, c("bicycle_parking"), point)
  bus.stops = nearby.elements(ua, c("bus_stop", "Marguerite"), point)

  return(c(trees, water, bike.parking, bus.stops))
}

features = data.frame(t(sapply(1:nrow(data), function(i){
  # bounding box size is set to 500; user specific choice
  bb = center_bbox(data[i, 3], data[i, 2], 500, 500)
  build.features(bb, data[i, 1])
})))

colnames(features) = c("trees", "water", "bike.parking", "bus.stops")
features$location = 1:5
features
```

```
  trees water bike.parking bus.stops location
1     6     3            2        10        1
2    20     1            9        26        2
3    33     0            0         3        3
4    74     2           21        14        4
5    80     1           10        17        5
```

Data Analysis

Incorporating Survey Data

When using OSM data, if the linked surveys need to be kept private they should be stored separately (here, in a separate CSV file). We can match the location data with survey responses by checking the location of each row in the survey responses, and pulling the features for that location. Then our rows for survey data and location features will match exactly, without merging into one dataset and compromising data privacy.

```
# Make data frame where location rows (GPS) match location rows (surveys)
location.features = data.frame(t(sapply(survey_data$location, function(loc){
  unlist(features[features$location == loc, c("trees", "water", "bike.parking", "bus.stops")])
})))
```

Generalized Linear Mixed Model

Now we are ready to train a model! Here we train a generalized linear mixed model, which regresses reported safety on circumstances (being alone or at night and geospatial features (number of nearby trees, water fountains, bike parking areas, and bus stops). We also include a random effect for each individual, coded as (1 | survey_data$id), which tells our model that we might expect variation due to individual survey responses, but are not interested in modeling that variance.

```
# Train GLM with random effects for each individual (survey_data$id)
mod = lmer(survey_data$safety ~ 1 + (1|survey_data$id) + survey_data$circumstance +
             location.features$trees + location.features$water +
             location.features$bike.parking + location.features$bus.stops)
```

Let us consider the model we have trained. Note that since we generated data randomly, we would not expect significant results for any location features. We did add negative random variables for the responses alone and at night, so it is reasonable to expect significant features there. We have also included our data generating file in this repository for interested readers.

```
summary(mod)
```

```
Linear mixed model fit by REML ['lmerMod']
Formula:
survey_data$safety ~ 1 + (1 | survey_data$id) + survey_data$circumstance +
    location.features$trees + location.features$water + location.features$bike.parking +
    location.features$bus.stops

REML criterion at convergence: 1372.9

Scaled residuals:
     Min       1Q   Median       3Q      Max
-2.79352 -0.62893  0.00234  0.65032  2.54735

Random effects:
 Groups         Name        Variance Std.Dev.
 survey_data$id (Intercept) 2.9896   1.7290
 Residual                   0.8906   0.9437
Number of obs: 450, groups:  survey_data$id, 30

Fixed effects:
                                 Estimate Std. Error t value
```

```
(Intercept)                      3.9744264  0.3612888  11.001
survey_data$circumstancenight  -1.0789030  0.1089704  -9.901
survey_data$circumstancetoday   1.1098806  0.1089704  10.185
location.features$trees         0.0016688  0.0027748   0.601
location.features$water        -0.0279945  0.0546194  -0.513
location.features$bike.parking  0.0096594  0.0123611   0.781
location.features$bus.stops    -0.0009348  0.0074015  -0.126

Correlation of Fixed Effects:
                  (Intr) srvy_dt$crcmstncn srvy_dt$crcmstnct lctn.ftrs$t
srvy_dt$crcmstncn -0.151
srvy_dt$crcmstnct -0.151  0.500
lctn.ftrs$t       -0.342  0.000             0.000
lctn.ftrs$w       -0.316  0.000             0.000             0.574
lctn.ftrs$bk.      0.278  0.000             0.000            -0.828
lctn.ftrs$bs.     -0.321  0.000             0.000             0.456
                  lctn.ftrs$w lctn.ftrs$bk.
srvy_dt$crcmstncn
srvy_dt$crcmstnct
lctn.ftrs$t
lctn.ftrs$w
lctn.ftrs$bk.     -0.570
lctn.ftrs$bs.      0.281      -0.606
```