Data Wrangling with MongoDB

Final Project Report

Thomas Odell Wood

odell.wood@gmail.com

Introduction:

I chose to use the Edmonton metro data for my final project because I know someone who lives there. Edmonton also happens to be the northern-most metro area in the Americas that has OSM data available from mapzen.com and I am fascinated by the Artic. Edmonton also has a reasonably small dataset which allowed for easier processing than my original choice of Seattle, my home.

Problems encountered with the map:

A. The dataset for an entire metro region was too large to process at once on my laptop. More than once my laptop would simply freeze and need re-booting despite my reliable Linux operating system.

B. The key "p.o.box:kr" was making it through the problematic character regular expression and causing errors.

C. The key "type" which was normally associated with the tag of the element being processed was actually a key in the OSM data which sometimes contained the value "water" that was getting through the data reshaping procedure show in data.py.

D. The address information for the edmonton region was included in the "ways" elements.

E. When auditing the street types, there were a few types of street that weren't encountered when auditing the Chicago OSM data, such as the popular street name "Way."

Solutions for problems encountered with map:

A. I split the edmonton.osm file into “nodes” and “ways” files and ran the submission.py script twice. I then edited the function “process_map” so that it no longer wrote the output to a JSON file, but instead added the reshaped data element directly to mongoDB.

B. I added a line of code to check if the “.” character was contained in any string about to be used as a mongoDB key.

C. The “type” key was not allowed to creep into any of the reshaped elements with a simple “if” statement.

D. I had made sure that the part of the logical structure of “reshape_element” was set up to ensure that any element that contained address data had it included despite the fact that only “nodes” elements from the OSM Chicago dataset had addresses as fields.

E. To correct the street names in Edmonton, I expanded upon the mapping from street abbreviations to the full-length name of the street by editing the “update_name” function in the submitted script.

Statistics:

<*see *.json files inside edmonton_results directory*>

Opportunities in the data:

I realized that it should be easy enough to implement a routine which could tell me which Starbuck's coffee shop was the most geographically isolated from all the other Starbuck's using the $near operator from mongoDB. However, while I was having difficulties dealing with how the

$maxDistance opertor handles input in the form of a solid angle, I realized that I could simple use the formula

$$distance = \arccos\left(\sin(lat_x) * \sin(lat_y) + \cos(lat_x) * \cos(lat_y) * \cos(long_x - long_y)\right) * 6371.0$$

to calculate the pairwise distances between the Starbuck's in Edmonton using the longitude and latitude of two Starbuck's coffee shops. Querying the database for all positions proximal to a single Starbuck's and then filtering those results down to Starbuck's locations seemed somewhat inefficient to me, so I decided to go the more direct route that did not require any additional queries. Using the pairwise distances between Starbuck's locations, the script loneliest_franchise.py takes the minimum of the pairwise distances for each Starbuck's and then take the maximum of that set to find the loneliest Starbuck's in Edmonton.

Pseudocode for *loneliest_franchise.py*:

```
def loneliest_franchise( franchise_regex, db, coll )

   # Fetch all locations whose name matches franchise_regex.
   franchise_results := db.coll.find({ "name" : franchise_regex})

   # Initialize a mapping for distances.
   pairwise_distance := {}
   a_big_number := infinity
   for store1 in franchise_results:
     pairwise_distance[store1] := {}
     for store2 in franchise_results:
       if store1 == store2:
         pairwise_distance[store1][store2] := a_big_number
       else:
         pairwise_distance[store1][store2] := dist(store1, store2)

   # Find the closest same franchise for each store.
   nearest_neighbor := {}
```

```
for store in pairwise_distance:
  nearest_neighbor[store] := min(pairwise_distance[store])

# Find the store with furthest nearest neighbor.
loneliest_franchise := max(nearest_neighbor)

return loneliest_franchise
```

I then used argparse to set up the script to find the loneliest franchise for any string loneliest_franchise.py takes as input. The actul code can be found in the edmonton_results folder that was turned in with the rest of the assignment.

Conclusion:

The data in the Open Street Map available for downloads from mapzen.com has numerous potential uses for both consumers as well as business leaders, as shown in the simple program loneliest_franchies.py. It has been my pleasure to present my work to you, the reader, and I anticipate hearing your critique of my project. Thank you and have a wonderful day.