Data Science – Project IoT using GPS, Data Visualization, and Anomaly Detection

The Internet of Things (IoT) involves the growing trend of having multiple small devices that all collect small amounts of information. Each IoT device uses a small processor. Most of these devices do not even have an operating system on them. The "smarts" is distributed – some of the pre-processing happens on the device, and some of it comes later on when the data is assimilated and analyzed.

Arrow corporate hires many people to integrate IoT devices which help control power for companies. The buildings they work on get scattered with small IoT devices. The devices monitor heat, light, and water usage, people presence, ventilation, sunlight, and cooling. By combining information from many sources, they form a larger picture, which tells more of a story then any individual device could. They are now promising to save money for customers, as a blanket statement. This level of IoT is beyond having a computer in your toaster, it is called Industrial IoT, or IIoT for short.

The data you are provided is GPS data, collected using an Arduino micro-controller. The Arduino has only 32K of memory, and 2K of stack. That's **kilo**bytes! It has much less then a megabyte of information. Yet the device can record information fro a GPS device to a micro-SD card.

Each GPS track is usually just one trip from Penfield to RIT, or from RIT to Penfield, or around Monroe County on an errand. During each trip, I try to drive a different path. During one trip, I do not necessarily stop at any particular traffic stop light. If the light is green, I drive through it. In fact, as I approach a red traffic light, I slow the car down and wait for a traffic light to change to green, so that by the time I get there it has changed and I anc roll through it. This is to avoid having to stop, and waste fuel. This keeps my gas mileage up.

Some trips are good trips, and I never have to stop for any traffic lights. Some trips are terrible trips, and I have to stop for all of the traffic lights. This is an example of the sampling problem. (This leads to the concept of the Bias/Variance problem, which we will talk about later on.)

With enough trips back and forth, I will eventually need to stop for most of the traffic lights.

Traffic lights that I do not have to stop for, over the course of ten or twenty trips are probably ones that I can easily avoid stopping at because they are predominantly green in the direction I need to drive.

(continued)

You are provided with a set of GPS data tracks in which Dr. K. Given the GPS data please achieve the following goals:

1. **GPS_to_KML.py** GPS_Filename.txt KML_Filename.kml
   Write a program to convert one GPS data file into one KML file that can be viewed in Google Earth.
   During the process you need to clean the data by filtering out and ignoring any redundant, erroneous or anomalous data.

   For example:
   a. If the vehicle is parked, you do not need multiple data points at that same location.
   b. If the vehicle is traveling in a straight line, you could ignore some points.
   c. The Arduino sometimes burps, and writes two GPS sentences to the same line of the data file.
      You must detect and ignore these anomalies.
      Otherwise it looks like the car jumps from one side of the planet to the other side.
   d. The Arduino sometimes loses its mind, and starts recording GPS values that jump all over the place.
      This especially happens if it loses connection to the antenna. You may need to ignore meaningless junk.
   e. When the trip first starts up, the GPS device is not moving. Do not worry about the data points when the vehicle has not started moving yet.
   f. When the vehicle parks, the GPS device stops moving. Do not worry about these non-moving data points at the end of a drive.
   g. The "heading" or "direction" information from the GPS is only valid if the car is moving. If the car stops at a stop sign or a traffic light, the "direction" is meaningless. So, you can only use this information if the speed of the vehicle is above a minimum speed.
   h. You need to convert the GPS sentences. You cannot rely on the comments in the code.
   i. Generate a KML file which can be examined in Google Earth. Use a yellow path. An example KML file will be provided, and discussion of this was given in the lecture on writing a program which writes a program.

2. **GPS_to_CostMap.py** *.txt KML_Mapfile.kml
   Imagine that you are UPS. You are given GPS tracks around Rochester from all of the trucks driving around to deliver packages, and need to assimilate them all into one map for planning purposes.
   Given a set of GPS tracks (*.txt files), produce one KML map of all places that a vehicle might have to:
   a. Stop for a sign or traffic light. (in yellow)
   b. Make a left turn. (in **red**, for a port side turn)
   c. Make a right turn. (in **green**, for a starboard side turn)

   If the vehicle stops and turns, then mark it as a turn.

   Locate in the data any left or right turns, or stops for traffic lights.
   You must define what a turn is, and then design a classifier to locate it in the data.
   For a simplistic example, a turn might be a 90 degree turn within ten seconds.
   That is probably too simplistic. You will probably have to do better.

   A stop might be indicated by the vehicle slowing down below 10 miles per hour, staying less then two minutes, and then accelerating back up again. **You** pick all of these parameters, and must design the classifier.

   You may use any data in the RMC or GGA sentences of the GPS file.
   [Do not use any debugging lines that Dr. Kinsman put in the GPS file if they are present. (lng, lat… in the file.)]

   You will need to agglomerate stops and turns across many paths into one stop or turn in your resulting map.

   Your turn detector might accidentally detect many stops at the same location, and you will need to agglomerate many separate detections into one example at that location. (continued)

The GPS location can vary from track to track. This is caused by the GPS satellites drifting over time. You will need to agglomerate detections at different locations into the same position on your output map.

3. Creating Pins:

   a. Yellow Pins: Label all the stops, at each long and lat, with a yellow pin in KML.

```
Example default yellow Placemark:
<Placemark>
        <description>Stop Light</description>
        <Point>
          <coordinates>-77.6805,43.0867,0.0</coordinates>
        </Point>
  </Placemark>
```

   b. Example Red Placemark:

```
# The Hex values are the colors: Alpha, Blue, Green, Red.  Delete this line.
<Placemark>
      <description>Red PIN for A Stop</description>
      <Style id="normalPlacemark">
        <IconStyle>
          <color>ff0000ff</color>
          <Icon>
            <href>http://maps.google.com/mapfiles/kml/paddle/1.png</href>
          </Icon>
        </IconStyle>
      </Style>
      <Point>
          <coordinates>-77.6800,43.0867,0.0</coordinates>
      </Point>
    </Placemark>
```

4. **Write-up** your results, with the following information in it. I will be reading this from the point of view of a company that is considering hiring you.
   a. Title, followed by names of all contributors.

   b. One Paragraph abstract which describes what you did.

   c. Overview section, which describes what is in the rest of the report. Give an overview of how you solved the problems.

   d. Team composition: What role(s) did each team member have.
   How did you balance the load across members?

   e. Converting GPS to KML.
   What issues did you have converting the files. How did you do data cleaning and anomaly detection? What issues where in the files?

   f. Stop Detection:
   What classifier, and what features did you use to detect a stop or stop light?
   What issues were there with it?
   How did you ignore the case when the vehicle was stopped to run an errand – such as drop off a book at the library or go shopping?

   g. Turn Detection:
   What classifier, and what features did you use to detect a left turn?
   How about a right turn? Where there any issues?

h. Assimilating across tracks:
Describe how you designed and wrote your program. What design pattern did you use? What intermediate data structures did you use? Did you build intermediate databases?

Did you hold all the information in memory at once? Did you parse all of the files at once?

How did you handle assimilation across different tracks?

What parameters did you use?

i. Results of a path file:
Show an example screen shot of your path KML file.

j. Results of a hazard file:
Show an example screen shot of your resulting turn and stop KML file.

k. Screen shot of a *.kml file you created, showing

l. Discussion section:
What problems did you find with the approach?
Did you have to do any noise removal or signal processing?

m. A summary conclusion of what you learned overall,
and how this might be useful for some commercial application.

5. **Turn into the GROUP dropbox:**
   a. Your well commented programs.
   b. An example GPS_Path.kml path file.
   c. An example GPS_Hazards.kml file of stops and turns.
   d. Your write-up in PDF format, and your resulting kml file.

6. **GRADING RUBRIC:**
   a. Well commented code, which an outside reader can read and get the gist of what is happening.
   If you use any special python judo, explain what you are doing in comments.
   For example, a novice programmer might not know what the python **zip** command does, or the **any,** or **all** statements in python. (15%)
   b. Ability to convert one GPS file to one KML file showing the path traveled. (30%)
   c. Ability to convert a set of GPS file to one KML map of stops and turns: (30%)
      i. Too many misses or false alarms for stops signs is penalized 10%
      ii. Too many misses or false alarms for left turns is penalized 10%
      iii. Too many misses or false alarms for right turns is penalized 10%

   d. Writeup with all parts, as specified above. (25 %)

7. **BUILDING COST FUNCTIONs**

NOT FOR 2019 but read this and understand how a cost function can be constructed:

Finding the optimum track. Of the routes provided, determine the best route to use.
This route includes the travel time from the starting location to the final location.
In order to define "best" we need a formal definition. It is neither just the fastest, nor just the shortest.

The cost function to evaluate the routes is
Cost = (Travel Time (mins) / 30 (mins)) + (1/10) (Maximum Velocity (mph) / 60 (mph)).

Notice how this is designed:

a. The total cost function is composed of two parts: the objective function and the regularization.

b. The objective function is what we really want to minimize – the travel time to work.

c. The regularization is something that would be *nice* to minimize, but is not strictly required.
The regularization helps break ties if two different trips take the same amount of time.

Notice also that the regularization is less then the objective function to keep the regularization part from dominating the objective function. That way you cannot pick the slowest trip in by default.

d. The usual travel times is 30 minutes, so the travel time is normalized by 30.
The usual maximum velocity is about 60 MPH, so the max velocity is normalized by 60.
This way the two normalized components of the objective function and the regularization are *commensurate* – or *reasonably comparable*.

Then the factor of 1/10 assures that the regularization does not dominate the objective function.

You might convert MPH to knots to do the math in knots instead of MPH, for easier computing.

e. The final computed cost has no units on it.