Count Aggregation Scripting

June 5, 2015

Michael Lockwood

Ron Ignacio

The Pierce County Office of the County Engineer Division utilizes count information in decision making and modeling for future projects. This important information must be accurate and timely. Traffic operations saw the need to improve the system from its previous baseline of ten (10) minutes per intersection per week of data to less than one (1) second with extended functionality.

Baseline System

The past method required importing files into Excel. For several files with large disk space needs, like the puck count data, the import time becomes unmanageable. Traffic engineers have used VB scripting to process the aggregation of counts from hours or in some cases 10-second intervals to meaningful daily and weekly summations. Each week required about 10 minutes to process and would force the engineer to initialize the file before running the script. This is tedious and time consuming.

When errata were discovered or the script required revisions to fit different formats of data, the engineers had to invest their time in further VB scripting. Despite expertise in VB scripting they were unable to remove the overhead of processing in Excel. Long processing times remained and their time became a sunk cost. It was also difficult to reprogram and rerun because the turn around on intermediate results was not timely.

Improved System

The improved system was instead developed with the open source scripting language Python; the preferred scripting language for Pierce County GIS. Two scripts have been developed. One called counts.py aggregates count information for pucks and is able to process files from each of the puck stations. No manual work of importing files is required and results appear in one (1) second. The second called count\_location.py aggregates count location data stored in hourly increments. It has the intelligence to correct errors in the recording of hours and has data structures established that could learn from the data to automatically correct counts. It builds its structures in one (1) second and can export several files in milliseconds. Both scripts were developed with Object Oriented Programming (OOP) frameworks in mind to allow very specific querying and addressing of data.

We spent time manually reviewing the results for error analysis. We noted that the data itself errs every Sunday at 1:00am causing some problems but nothing that the script cannot handle. The scripts consistently produce the desired results that match gold standards with perfect precision and recall.

Comparison

Table 1: Compare Baseline and Improved Systems

|  |  |
| --- | --- |
| **Baseline** | **Improved** |
| 10 minutes per week processing time | < 1 second per week processing time |
| Store all data in files with ~14MB per day | Store data in class structures with ~20KB per day |
| Manually process each period | Automatically process each period |