Multi-threaded/-processed Requests to Cloud Services for Intelligent Address Standardization

PP103 Minisymposterium: Software Productivity and Sustainability for CSE and Data Science, SIAM CSE 2019

ILLINOIS INSTITUTE OF TECHNOLOGY

Aleksei Sorokin (asorokin@hawk.iit.edu), Andy Liu (aliu2@imsa.edu), Sou-Cheng T. Choi (schoi32@iit.edu)

Department of Applied Mathematics, Illinois Institute of Technology, USA

Introduction

Motivation:

Driving to college for the first time, I input "3333 Soth Wabash Avenue, Chicago" into Google Maps [7] and received an exact address to which directions were provided. How did Google Maps know the exact address I was referring to? The address I typed in contained a misspelling of "South" and missed data-fields such as state name and zip code. Despite these human errors (noise), Google Maps still identified the exact address. Here lies the basis of *intelligent address standardization*: mapping a noisy address to a clean address that consists of address components such as street number, street name, and city name.

Background:

The Python address-standardization software [1] and study [2] compare the accuracies of different web or cloud services for geocoding and address standardization. The program introduces noise to a clean address before requesting location identification from a web service. A web service's back-end machine-learning models try to identify a user-requested address that often does not contain all correct address components known to the service. A web service's accuracy is tested by counting how many data fields from the clean address match the service's evaluation of the corresponding noisy address. Testing many addresses on a web service gives a more precise average accuracy value of the service.

Contribution:

We added multithreading and multiprocessing options for evaluating a web service's accuracy over a large number of addresses. Compatible web services include Geocoder [4], Data Science Toolkit [5], and usaddress [6] (we did not evaluate Google Maps in this study as the service started charging for batch requests in July 2018).

Sample Data

The dataset used was sourced from OpenAddresses [3], a database of international addresses. We have sampled over 10 million addresses from the American Midwest and Northeast, as shown on the map below. Addresses in this dataset contained six key fields of interest: city, street, street number, region, postcode, zip+4.

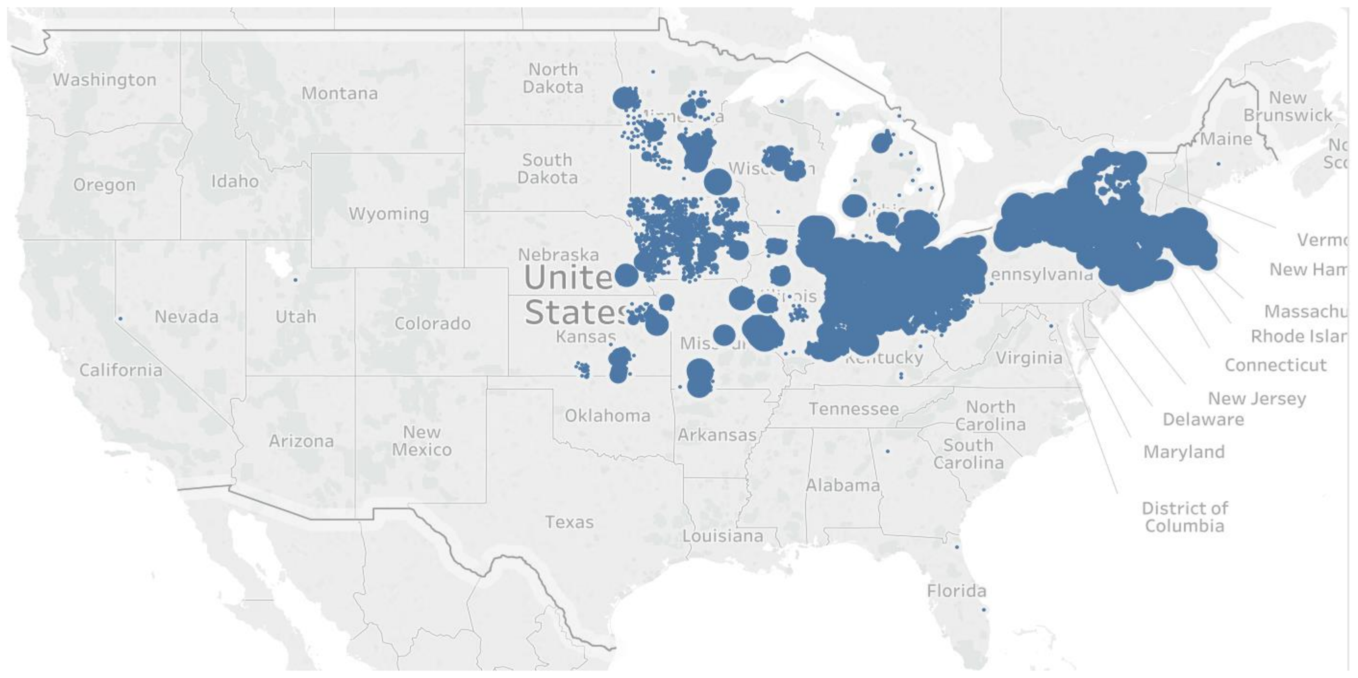

Figure 1: Visualization of a sample of OpenAddresses dataset we used; larger circles represent zip codes with a larger number of addresses in the dataset. This figure was created using Tableau [8].

References

[1] Choi, S.C.T., and Lin, Y.H. *Comparison of Public-Domain Software and Services for Probabilistic Record Linkage and Address Standardization*. Python software, 2017, https://github.com/schoi32/prl-splncs. Accessed 23 Feb. 2019.

[2] Choi, S.C.T., Lin Y., and Mulrow E. Comparison of Public-Domain Software and Services for Probabilistic Record Linkage and Address Standardization. *Towards Integrative Machine Learning and Knowledge Extraction*. Springer, 2017, pp. 51—66.

[3] *OpenAddresses*. http://openaddresses.io. Accessed 23 Feb. 2019.

[4] *geocoder.us*. http://206.220.230.164. Accessed 23 Feb. 2019.

[5] Warden, P. *The Data Science Toolkit*. http://www.datasciencetoolkit.org. Accessed 23 Feb. 2019.

[6] Gregg F., Deng C., Batchkarov M., and Cochrane, J. *usaddress*. Python library, 2014, https://github.com/datamade/usaddress. Accessed 23 Feb. 2019.

[7] *Google Maps*. https://www.google.com/maps. Accessed 23 Feb. 2019.

[8] *Tableau*. https://www.tableau.com. Accessed 23 Feb. 2019.

User Arguments

Variable parameters for a single trial of web-service evaluation

- n = Number of addresses
- m = Web service (Geocoder, Data Science Toolkit, usaddress)
- g = Noise level (between 0 and 1)
- p = Processing type (Serial, Multithreading, Multiprocessing)
- t = Number of tasks (Irrelevant for serial processing)

Serial Processing

Evaluating addresses one-by-one in sequential order

General Process:

1. Read in User Arguments
2. Load a clean address from the Sample Data
3. Introduce noise to the clean address
4. Issue a RESTful-API call to the cloud service asking for identification of the noisy address
5. Wait for the cloud service to respond (Elicits Parallel Processing)
6. Extract relevant data fields from the response address
7. Count matching data fields between the clean address and response address
8. Update total right and wrong counts for each data field
9. Repeat steps 2-8 on each address

Strength: Low memory usage
Weakness: Slow runtime (see step 5)

Parallel Processing

Evaluating batches of addresses in parallel tasks

Approach:

1. Implement multithreading and multiprocessing using the Python packages `threading` and `multiprocessing`
2. Track overall accuracy by aggregating task totals

Multithreading vs. Multiprocessing:

- Multithreading creates tasks (threads) in a shared memory space and switches between tasks when downtime is incurred. On regular architecture, all threads are executed in one process. Some machines distribute the threads to multiple cores.
- Multiprocessing maps tasks to separate processes that are executed simultaneously in independent memory spaces.

General Process:

1. Read in User Arguments
2. Load addresses from the Sample Data
3. Split addresses into batches
4. Create tasks: Each task is responsible for evaluating a batch of addresses
5. Start all tasks simultaneously. For each task:
   a) perform serial processing on its associated batch of addresses
   b) write total counts to a global list of task results
6. Wait for all tasks to terminate
7. Aggregate counts from the global list of task results
8. Calculate accuracy based on aggregated totals

Strength: Fast runtime
Weakness: High memory usage (see Multithreading vs Multiprocessing)

Performance Results

Cloud Service Comparison:

Table 1: Comparing web-services' average response time for one address and accuracy for 100,000 clean and noisy addresses

Cloud Service	Single Response Time (second)	Accuracy (%) with g=0 (clean)	Accuracy (%) with g=0.1 (noisy)
Geocoder	1.7×10^{-1}	52.1	36.8
Data Science Toolkit	5.5×10^{-2}	54.8	41.1
usaddress	4.0×10^{-4}	99.2	95.0

Processing Type Comparison:

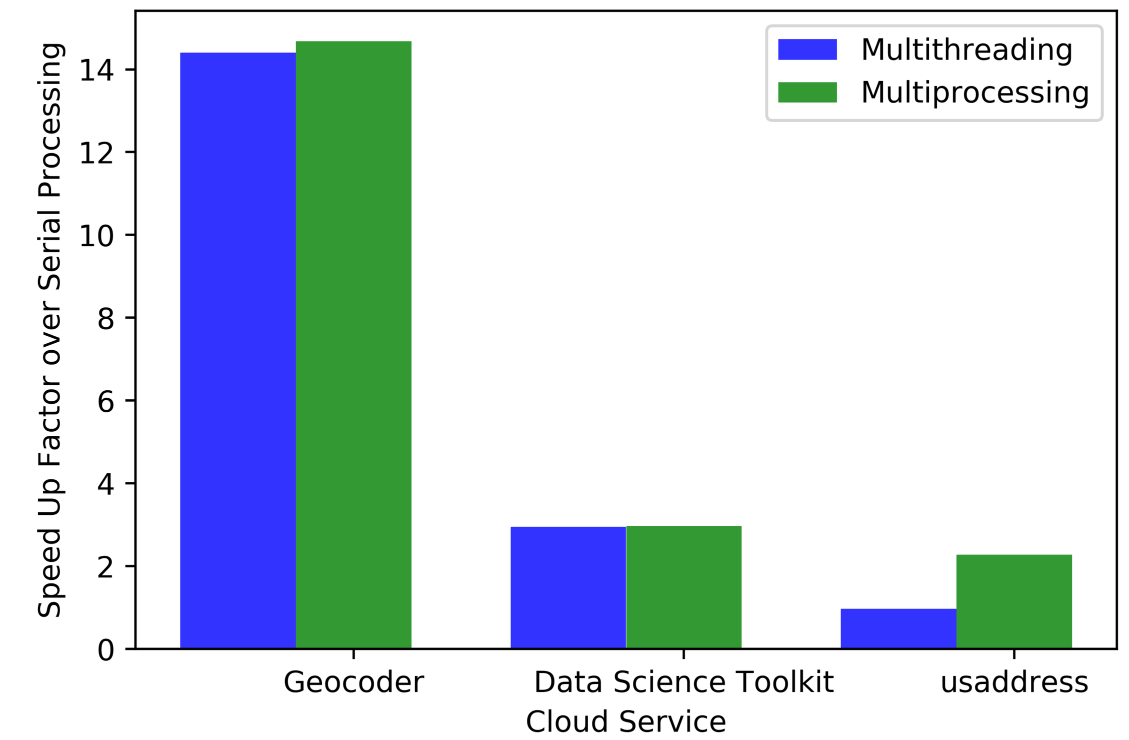

Figure 2: Test Parameters: n=100,000 (Addresses) ; g=0 (Clean); t=15 (Tasks)

Field-Level Accuracy Comparison:

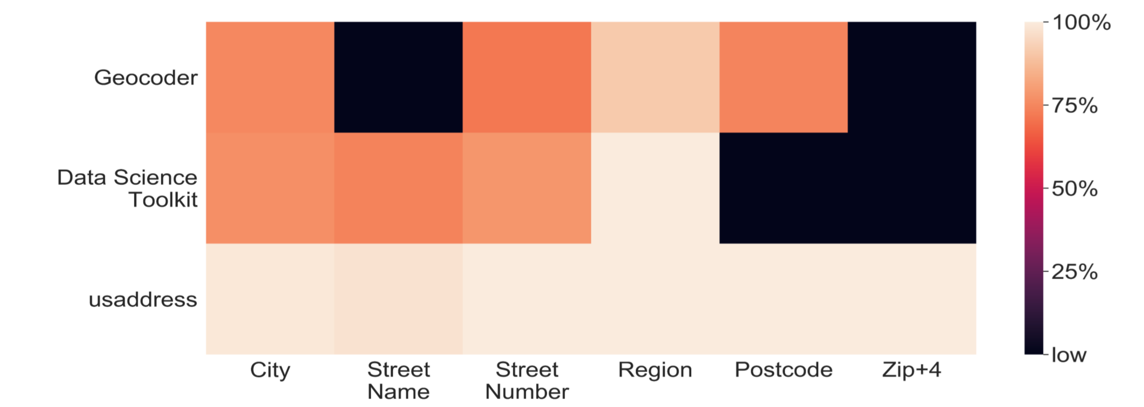

Figure 3: usaddress achieved close to 100% accuracy for all fields. Note: Geocoder does not return zip+4. Data Science Toolkit does not return Postcode or zip+4.

usaddress on One Million Addresses:

Table 2: Parameters: n=1,000,000; g=0; p=3 (Multiprocessing) ; t=15 (Tasks)

Runtime (sec)	Accuracy (%)						
4 Cores	Overall	City	Street Name	Street Number	Region	Postcode	Zip+4
280	97.3	98.6	87.3	99.4	100	100	100

Conclusions

- Of the 3 cloud services tested, usaddress yielded the fastest, most accurate responses
- Parallel processing is advantageous when evaluating with a slow web service
- For fast web services, multiprocessing can be more advantageous than multithreading

Ongoing/Future Work

- Automating processing-type selection and number of threads or processes
- Developing convolutional neural networks (CNNs) with long short-term memory (LSTM) for improved address standardization
- Writing a paper on final results and findings

Acknowledgements

- We thank American Family Insurance and eMALI.IO Ltd. for sponsoring this research
- We also thank Jack Huang and Lek-Heng Lim from the University of Chicago for helpful discussion