Data Science *for the* **Public Good**

Developing and Evaluating a Toolkit for Geocoding and Reverse Geocoding

Anjali Mehta, Annie Xie,
Jianing Cai, Marijke van der Geer,
Nakshatra Yalagach, Prashanth Wagle,
Steve Zhou, Trinity Chamblin

Supervised by: Dr. Alan Wang.

January 24, 2024

Abstract

This document presents an in-depth exploration of geocoding and reverse geocoding, crucial processes in geographic information systems (GIS) that convert between location-based data and human-readable addresses. Geocoding transforms address information, such as street and city names, into geographical coordinates (latitude and longitude), enabling accurate map representations and spatial analysis. Its primary goal is to link textual location data with spatial representation, which is fundamental for applications in navigation, location-based services, real estate, and urban planning. Conversely, reverse geocoding converts geographic coordinates back into human-readable addresses, providing context to spatial data and enhancing user experience in various applications by displaying detailed location information. This document aims to establish guidelines for creating effective and accurate geocoding and reverse geocoding tools, emphasizing their importance in integrating spatial information into diverse applications and enhancing the utility of geospatial data.

1 Introduction

Geocoding and reverse geocoding are fundamental processes in the field of geographic information systems (GIS) that play a crucial role in converting between location-based data and human-readable addresses. These techniques serve as essential tools for integrating spatial information into various applications, enabling efficient mapping, location-based services, and data analysis.

Geocoding is the process of transforming descriptive address information, such as street names, city names, and postal codes, into corresponding geographic co-

ordinates, namely latitude and longitude. This conversion allows addresses to be accurately represented on a map, facilitating spatial analysis and visualization.

The purpose of geocoding is to bridge the gap between textual location data and its spatial representation. By creating a geocoding tool, users can efficiently convert a vast amount of address data into geographic coordinates, making it possible to overlay and analyze diverse datasets on digital maps. Geocoding finds extensive applications in navigation systems, location-based services, real estate analysis, and urban planning, among many others.

On the other hand, reverse geocoding performs the inverse process by converting geographic coordinates (latitude and longitude) back into human-readable addresses. This allows users to identify the location corresponding to a set of coordinates, providing valuable context to spatial data.

The intended use of a reverse geocoding tool is to enable applications and systems to display meaningful location information to end-users. By employing reverse geocoding, location-aware applications can provide users with detailed addresses, city names, postal codes, and other location-based information based on their current or specified coordinates. This functionality is vital for enhancing user experience, aiding navigation, and displaying relevant information in location-based services, mapping applications, and social media platforms.

1.1 Our Tools

1.1.1 Geocoding Tool

This Python program performs batch geocoding using the Census API.[3] It takes a CSV file as input, splits it into smaller chunks due to the API's row limit of 10,000, and then performs geocoding on each chunk separately. The geocoded results are saved as separate CSV files in the specified output directory. The script also allows resuming the process from where it left off, making it efficient for large datasets.

The command line arguments allow users to customize the input file, output directory, and control the verbosity and behavior of the geocoding process. The program can be executed using the following parameters.[2]

```
python script_name.py -i input_file.csv -o output_directory -v -f
```

`-i` or `--input_file`: The path to the input CSV file.

`-o` or `--output_dir`: The directory where the geocoded results will be saved.

`-v` or `--verbose`: (Optional) Shows debugging outputs if provided.

`-f` or `--force`: (Optional) Forces the script to override pre-existing output files.

Access Geocode Tool Here

1.1.2 Reverse Geocoding Tool

This Python program performs reverse geocoding using the OpenStreetMap Nominatim API. The input data was downloaded from the National Address Database, an extensive resource for geographic data [4]. Taking a CSV and a SHP file as

input, this tool populates the data frame in the CSV with an 'address' column, utilizing latitude and longitude coordinates. For compatibility with larger datasets, this tool reverse-geocodes data into smaller chunks separately. The remaining data frame with filled addresses is then saved as a separate CSV file in a specified output directory.[1]

The command line arguments allow users to customize the input CSV file, the input SHP file, and the output directory. The program can be executed using the following parameters.

```
python reverse_geo_CLI.py --cf county.csv --sf shape.shp --of output.csv
```

`-cf`: The path to the input CSV file that contains county data.

`-sf`: The path to the input SHP file that contains the shape file for the corresponding county data.

`-of`: The path to the output CSV file that contains the reverse-geocoded county data.

Access Reverse Geocode Tool Here

1.2 Definitions and Acronyms

Term	Definition
Geolocator	A device or software application that utilizes the GPS technology to determine and provide the precise geographic location of an object, person, or vehicle in real-time.
Centroid	A geometric concept that represents the "center of mass" or "center of gravity" of an object or a geometric shape. It is a point that summarizes the distribution of mass or points within the shape in a way that balances the shape evenly.
Shape File	A widely used geospatial data format that contains both geometric and attribute data. It consists of multiple files with extensions such as .shp (geometric data), .shx (index), and .dbf (attribute data). The .shp file stores the spatial features like points, lines, or polygons, while the .dbf file contains tabular information with attributes associated with each shape. This format is used in Geographic Information Systems (GIS) to represent and exchange geographic data.

2 Rubric

2.1 Well-written Code

The following QOS(Quality of Service) parameters demonstrate why our tools are well-written.

Reverse Geocode and Geocoding Tool:

- **Usability:** Both tools are equipped with Command Line Interface to enhance its usability. This command-line functionality provides users with a flexible and efficient way to control the tool, automate tasks, and integrate it into their workflows seamlessly.
- **Readability:** The tool is easy to read and understand. It uses clear and descriptive variable and function names, follows consistent indentation, and includes comments where necessary to explain complex logic or algorithms. The functions are abstracted into individual simple units. Docstrings are added to every function in the geocoding tools. Constants and configurations have been abstracted.
- **Reusability:** The code is divided into smaller, reusable modules or functions. As changes to one module are less likely to impact others, the code is easy to reuse and modify. Since the .py files are modules, individual functions can be imported easily into other codes.
- **Testability:** The code is designed with testability in mind. The Performance section outlines the process for our testing stage. The geocoding tool is testable as the function has a single point of entry and well-defined inputs and outputs. Post-completion of running the reverse geocode tool, the output is tested to check if there are any missing addresses. Any additional test cases can be formulated to validate the code.
- **Efficiency:** The tools are optimized for performance and resource usage. The geocoding tool batch geocodes the input in O(n) time. The tools ensure that the program runs smoothly and responds quickly to user interactions.

2.2 Justification

Geocode Tool

	Inputs	Outputs
Command-Line	*arg[1] Input File* Path to the .csv file to be geocoded. --- *arg[2] Output Destination:* Path to the directory to save the geocoded data chucks --- *arg[3] Force Flag:* If True, overwite existing .csv files	One or more geocoded file chunks
Function	*chunk_files(input_file, output_dir):* splits input into smaller chuncks. --- *batch_geocode(chunk_names, force):* takes a list of chunk names, sends each chunk to the Census geocoding API. --- *batch_geocode_util(input_file, output_dir, force):* number of queries to run at once	One or more geocoded file chunks

arg[1] Input File: This input is essential as it provides the path to the input .csv file that contains the data to be geocoded. The function needs this file to extract the street address column and perform the geocoding process.

arg[2] Output Destination: This input is necessary to specify the directory where the geocoded data chunks will be saved as separate .csv files. The function requires this information to organize and store the geocoded data.

arg[3] Force Flag: The Force Flag input is provided to control the behavior when saving the geocoded data. If set to True, the program will automatically overwrite existing .csv files in the output directory. This option gives users the flexibility to manage previously saved data in the output directory.

chunk_files(input file, output_dir): This function is designed to split the input .csv file into smaller chunks and save each chunk as a separate .csv file in the specified output directory. It returns a list of names of the saved chunk files, enabling further processing or reference.

batch_geocode(chunk names, force): This function takes a list of chunk names (files containing smaller subsets of data) and sends each chunk to the FCC (Federal Communications Commission) geocoding API. It then saves the geocoded results as separate .csv files in the output directory. The "force" parameter allows users to control whether to overwrite existing files during the geocoding process.

batch_geocode_util(input file, output dir, force): This function is a utility function that wraps the batch_geocode function, providing a higher-level interface for users to perform the geocoding process. It takes the input file, output directory, and force flag, along with an optional parameter for the number of queries to run at once (batch size). It then returns one or more geocoded file chunks as output.

Reverse-Geocode Tool

	Inputs	**Outputs**
Command-Line	*path(str):* path to file to reverse-geocode --- *filename(str):* name of file to reverse-geocode --- *save_as(str):* name to save new, reverse-geocoded file under --- ***args:* function arguments (outlined below)	none – saves a new csv file containing the original data and the reverse-geocoded column
Function	*df(dataframe):* dataframe containing the data to reverse-geocode --- *latitude(str):* name of latitude column in data --- *longitude(str):* name of longitude column in data --- *batch(int)(optional; default = **):* number of queries to run at once; defaults to max possible before hitting rate-limit	*df(dataframe):* contains the original data and the reverse-geocoded column

path(str): This input is used to provide the path to the file that contains the geographical data for reverse-geocoding. It allows the function to access the required data and perform the reverse-geocoding process.

filename(str): Similar to the 'path' input, this input provides the name of the file containing the geographical data. It complements the 'path' input and ensures

the correct file is accessed for reverse-geocoding.

save_as(str): This input specifies the name under which the new reverse-geocoded file will be saved. After reverse-geocoding, the function creates a new file with the reverse-geocoded data, and this input helps to assign a meaningful name to the new file.

****args:** This input represents function arguments that are outlined below.
This parameter allows for additional flexibility in passing optional arguments to the function.

none: This output means that when no specific arguments are provided for the function, it will save a new CSV file containing both the original data and the additional column with the reverse-geocoded information.

df(data frame): This input expects a DataFrame containing the geographical data to be reverse-geocoded. The function will operate on this DataFrame to add the reverse-geocoded information.

latitude(str): This input indicates the name of the column in the DataFrame that holds the latitude values. It helps the function to identify the correct column for latitude data.

longitude(str): Similar to 'latitude', this input specifies the name of the column in the DataFrame that holds the longitude values.

batch(int): Reverse geocoding can involve making API queries, and specifying a batch size allows the function to control the number of queries executed at once. By limiting the batch size, the function can avoid rate-limiting issues with the reverse geocoding service. The default value of ** means that if this parameter is not explicitly provided, the function will use the maximum possible batch size.

2.3 Performance and Limitations

Performance:

1. Usability:

The geocoding tool takes CSV file, batches it into chunks of a max size of 10,000 and geocodes each of these batches and saves it into csv files which is straightforward.

The reverse geocoding algorithm takes in a data frame and automatically outputs the same data frame with updated information. There is nothing else that requires users to edit or add on, which makes it easier to use and opens to users from all programming levels. Besides, the output is automatically added on the initial data frame, which saves users time to combine the result with the original table again and makes it straightforward to observe the address this algorithm generated based on the coordinating longitude and latitude.

2. Runtime speed: The geocoding algorithm is entirely dependent on the API and takes around 30s to geocode a file of 10,000 rows, which is approximately 333.333 rows/second. Note that thand it is entirely dependent on the API. The reverse geocoding algorithm took around 8.5 minutes to fill in 1008 rows, which is approximately 1.976 rows/second. Specifically, generating missing points took the most part of the time (8m 24.3s).

3. Accuracy (Reverse Geocoding):

Output Lat	**Google Lat**	**Lat Diff**	**Output Long**	**Google Long**	**Long Diff**
40.811162	40.811074	0.000088	-73.855841	-73.856235	0.000394
40.857552	40.858042	-0.00049	-73.847622	-73.846503	-0.001119
40.870227	40.857435	0.012792	-73.880354	-73.863655	-0.016699
40.808179	40.808568	-0.000389	-73.928064	-73.928095	0.000031
40.816928	40.816737	0.000191	-73.861661	-73.861616	-0.000045
40.811335	40.816668	-0.005333	-73.926864	-73.923128	-0.003736
		0.00114317			**-0.003529**

To check the algorithms' accuracy, we used Google geocoding API to cross-check the result we got. We used the address that our algorithm generated and passed this address into the Google API, which will return me a latitude and a longitude. Then, we compared the latitude and longitude Google gave us with the original latitude and longitude. Next, we randomly chose six results to examine. Although it is worth noting that the largest difference could be 0.013 for latitudinal and 0.017 for longitudinal, the overall latitude difference is 0.00114317 and the longitude difference is 0.003529, which are roughly close to zero.

Limitations:

1. Usability: For geocoding, the input format is rigid, it has to have Street address, City, State, ZIP Code, and so on. For reverse geocoding, the data frame

passed into the algorithm must contain columns named exactly 'latitude' and 'longitude', which to some extent limit the flexibility of the tool.

2. Runtime speed: The geocoding tool has a linear runtime despite it gecoding in batches, as it is a function of the API that it uses rather than the tool itself. The reverse-geocoding algorithm used two main for-loops, which, in theory, makes it an $O(n^2)$ runtime. When the number of missing addresses is big enough, it will potentially slow down the speed of the program significantly.

3. General: The geocoding algorithm is dependent on the database of the Geocoder tool which may be imperfect. It supports only CSV files for now. Exception handling is something that we need to take care of in the future. The reverse-geocoding algorithm is a black box algorithm, which highly dependent on the Nominatim API. If the author of Nominatim API makes changes, users may need to check the Nominatim API by themselves for the latest version.

References

[1] G. Contributors, "GeoPandas: Python tools for geographic data," *Zenodo*, 2021. [Online]. Available: https://doi.org/10.5281/zenodo.592855

[2] Python Software Foundation, "urllib - url handling modules in python," https://docs.python.org/3/library/urllib.html, Year of Access.

[3] U.S. Census Bureau, "U.S. Census API," https://www.census.gov/data/developers/data-sets.html, Year of Access.

[4] U.S. Department of Transportation, "National address database (nad)," https://www.transportation.gov/gis/national-address-database, Year of Access.