Trajectory Recovery API Documentation

Last updated: February 16, 2025.

1 Preliminaries and Dependencies

The `trajectory_recovery` Python module provides an interface for evaluating datasets on the algorithms presented in D'Silva et al. in [1]. It also presents implementations of baseline work proposed by Xu et al. in [2]. The source code is available at the GitHub repository, under the MIT license. This document contains the API documentation for the `TrajectoryRecoveryA` (the original algorithm by Xu et al.) and `TrajectoryRecoveryB` (with our enhancements) classes. Note that the `TrajectoryRecovery` class is an alias of `TrajectoryRecoveryB`. The dependencies of this module, excluding the Python standard library, are:

- numpy [3]
- pandas [4]
- matplotlib [5]
- scipy [6]
- geopy [7]
- levenshtein [8]
- tqdm [9]

Throughout, we will use the following abbreviations:

- n: The number of trajectories in the dataset.
- m: The number of locations in the dataset.
- t: The number of time steps in the dataset.
- d: The number of time steps that occur in 24 hours.

If you notice any bugs or inconsistencies with the module or this document, please create an issue on the GitHub repository.

2 API Documentation

If this documentation intends to refer to the `TrajectoryRecoveryA` and `TrajectoryRecoveryB` classes simultaneously, this will be written as `TrajectoryRecovery[A|B]`.

`TrajectoryRecovery[A|B]()`, the constructor, expects all of the following arguments (unless otherwise specified) in the given order:

- `aggregated_dataset`: *pandas.DataFrame* or *numpy.ndarray*

  The aggregated dataset with exactly t rows and m columns. Rows must appear in chronological order. The dataset must begin at 00:00. The order of columns (locations) from left to right is used for the below.

- `grid`: *dict* or *list* or *numpy.ndarray*

  Location information that maps i, (the i-th location above) to a *tuple* representing its location in space. They may be mapped to cartesian coordinates, or given as latitude and longitude coordinates.

- `num_trajectories`: *int*

  n, the number of trajectories in the dataset.

- `num_locations`: *int*

  m, the number of locations in the dataset.

- `num_timesteps`: *int*

  t, the number of time steps in the dataset.

- `num_timesteps_per_day`: *int*

  d, the number of time steps that can occur in 24 hours.

- Optional: `cartesian`: *bool*

  Indicates whether the locations in the grid are mapped to cartesian coordinates, or are latitude and longitude coordinates. If this is not provided, then this is set to *True*.

`TrajectoryRecoveryA.run_algorithm()`

Runs the algorithm on the initialised aggregated dataset.

Returns *None*.

`TrajectoryRecoveryB.run_algorithm(lookback)`

Runs the algorithm on the initialised aggregated dataset.

Returns *None*.

- `lookback` : *int*

  The number of past days to consider when linking days. If a zero or negative integer is given, this is set to 1. Note that if `lookback` $= 1$, this is equivalent to the strategy used in `TrajectoryRecoveryA`. Also note that values > 7 tend to offer little-to-no benefit for accuracy.

`TrajectoryRecovery[A|B].evaluate(truth_dataset)`

Evaluates the current predictions on a given truth dataset.

Returns a *dict* containing accuracy, recovery error, and top-k uniqueness metrics for the predicted and true datasets, for all $1 \leq k \leq 5$. It also contains a list of tuples where each (i, j) means that the i-th predicted trajectory was matched with the j-th true trajectory.

- `truth_dataset`: *list[list[tuple]]*

  A 2D *list* of n true trajectories. The order of rows (trajectories) is not important, but each trajectory must be a *list* of t locations in chronological order. Each location is a *tuple* expressing the location coordinates.

`TrajectoryRecovery[A|B].visualise(timestep_range)`

Plots all the matched predicted and associated true trajectories within the given time step range.

Returns a *list* of *matplotlib.pyplot* figures.

- `timestep_range`: *tuple[int, int]*

  The range of time steps to plot, left-inclusive and right-exclusive. If no time step range is given, then the range of $[0, \min(t, d))$ is used.

`TrajectoryRecovery[A|B].gain(trajectory_1, trajectory_2)`

Calculates the gain of two trajectories.

Returns a *float* of the calculated gain.

- `trajectory_1` : *list*
- `trajectory_2` : *list*

  Each trajectory is expressed as a *list* of locations. The representation of locations (e.g. by *int* or *tuple* of coordinates) is not important, as long as it is consistent.

`TrajectoryRecovery[A|B].uniqueness(data, k)`

Calculates the top-k uniqueness of a dataset.

Returns a *float* of the calculated gain.

- `data` : *list[list]*

  A 2D *list* of n trajectories. Each trajectory is expressed as a *list* of t *tuples*, representing sequential locations. The representation of locations (e.g. by *int* or *tuple* of coordinates) is not important, as long as it is consistent.

- `k` : *int*

`TrajectoryRecovery[A|B].get_predictions(location_type)`

Returns a 2D *list* of the n predicted trajectories, where each trajectory is a *list* of t locations.

- `location_type`: *"coordinate"* or *"id"*, both as *str*

  If set to *"coordinate"*, then locations are represented as *tuples* of *floats*. If set to *"id"*, then locations are represented by their assigned IDs, as an *int*. If no argument is given, then this is *"coordinate"* by default.

`TrajectoryRecovery[A|B].get_results()`

Returns a *dict* containing the results of the most recent evaluation, including accuracy, recovery error, top-k uniqueness, and Levenshtein accuracy metrics for the predicted and true datasets, for all $1 \leq k \leq 5$. It also contains a *list* of *tuples* where each (i, j) means that the i-th predicted trajectory was matched with the j-th true trajectory.

References

[1] N. D'Silva, T. Shahi, T. Dokk Husveg, A. Sanjeeve, E. Buchholz, and S. S. Kanhere, "Demystifying trajectory recovery from ash: An open-source evaluation and enhancement," in *2024 17th International Conference on Security of Information and Networks (SIN)*, pp. 1–8, 2024.

[2] F. Xu, Z. Tu, Y. Li, P. Zhang, X. Fu, and D. Jin, "Trajectory recovery from ash: User privacy is not preserved in aggregated mobility data," in *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, International World Wide Web Conferences Steering Committee, Apr. 2017.

[3] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, Sept. 2020.

[4] T. pandas development team, "pandas-dev/pandas: Pandas," Feb. 2020.

[5] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[6] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[7] GeoPy Contributors, "GeoPy: Python Geocoding Toolbox." https://github.com/geopy/geopy.

[8] Levenshtein Contributors, "Levenshtein." https://github.com/rapidfuzz/Levenshtein.

[9] Casual Programmer's Incremental Developments, "tqdm: A Fast, Extensible Progress Bar for Python." https://github.com/tqdm/tqdm.