aceso Documentation

Release 0.1.0

Brian Lewis

Aug 05, 2019

Contents

CHAPTER 1

Introduction

Aceso is a lightweight Python package for measuring spatial accessibility. The primary models it implements are the Two-Step Floating Catchment Area (2SFCA) model and variants thereof, including gravity models, 3SFCA, and MSFCA.

CHAPTER 2

Installation

Aceso is available on PyPI and can be installed using `pip`.

View the source code on GitHub.

CHAPTER 3

Contents

3.1 Usage

The diverse and ever-growing ecosystem of measures of potential spatial accessibility is too complicated for this short document or its inexpert author to do justice. Each model has strengths, peculiarities, and shortcomings. Aceso was designed to make it easy to experiment with different choices of model and model parameters.

Accordingly, this documentation assumes that users have already selected a model from the menagerie when they arrive at Aceso. That said, experience shows that the default gravity model with raised cosine decay is often a good starting point.

3.1.1 Model initialization

All model classes should specify a decay function that operates on 1d-arrays and parameters for that function in dictionary format. Certain functions can be specified by name, including `'uniform'`, `'raised_cosine'`, `'gaussian'`, and `'parabolic'`. Other functions can be passed directly as callables.

For concreteness: 2SFCA uses the uniform decay function. Enhanced Two-Step Floating Catchment Area (E2SFCA) models would define a piecewise function over different distance ranges.

Where relevant, decay parameters should be measured in the same units as the travel impedances (kilometers, miles, minutes, etc.).

3.1.2 Model usage

All model classes support the method `calculate_accessibility_scores(distance_matrix, demand_array, supply_array)`. Here, the entry of `distance_matrix` in row i and column j corresponds to the distance (or time) between demand location i and supply location j. Both `demand_array` and `supply_array` will default to arrays of ones of the correct length if not provided explicitly.

In one standard use pattern, demand locations are stored in a GeoPandas GeoDataFrame object. Access scores can then be calculated and added to the dataframe as a new column:

```
model = aceso.TwoStepFCA(radius=60.0)
gdf['access_score'] = model.calculate_accessibility_scores(
    distance_matrix=distance_matrix,
    demand_array=gdf['population'].values
)
```

3.1.3 Notes

- Aceso is agnostic about the source of `distance_matrix` and does not currently implement any methods to calculate it. Great-circle distance can be calculated efficiently using the haversine formula, which has a very fast implementation in the cHaversine package. Retrieving matrices of driving times is more complicated and may involve calls to external routing APIs.
- See *Contributing* if there are other models you'd like to see supported.

3.2 Models of spatial interaction

Classes to calculate gravity-based measures of potential spatial accessibility.

These measures assign accessibility scores to demand locations based on their proximity to supply locations. The main model used here is a gravitational model using non-standard decay functions.

References

Luo, W. and Qi, Y. (2009) An enhanced two-step floating catchment area (E2SFCA) method for measuring spatial accessibility to primary care physicians. Health and Place 15, 11001107.

Luo, W. and Wang, F. (2003) Measures of spatial accessibility to health care in a GIS environment: synthesis and a case study in the Chicago region. Environment and Planning B: Planning and Design 30, 865884.

Wang, F. (2012) Measurement, optimization, and impact of health care accessibility: a methodological review. Annals of the Association of American Geographers 102, 11041112.

Wan, Neng & Zou, Bin & Sternberg, Troy. (2012). A 3-step floating catchment area method for analyzing spatial access to health services. International Journal of Geographical Information Science. 26. 1073-1089. 10.1080/13658816.2011.624987.

3.2.1 Gravity Model

class aceso.**GravityModel**(*decay_function*, *decay_params={}*, *huff_normalization=False*, *suboptimality_exponent=1.0*)

Represents an instance of a gravitational model of spatial interaction.

Different choices of decay function lead to the following models:

- Standard gravity models
- Two-Step Floating Catchment Area (2SFCA) models
- Enhanced 2SFCA (E2SFCA) models
- Three-Step FCA (3SFCA) models
- Modified 2SFCA (M2SFCA) models

- Kernel Density 2SFCA (KD2SFCA) models

Initialize a gravitational model of spatial accessibility.

Parameters

- **decay_function** (*callable or str*) – If str, the name of a decay function in the `decay` module. Some available names are 'uniform', 'raised_cosine', and 'gaussian_decay'.

  If callable, a vectorized numpy function returning demand dropoffs by distance.

- **decay_params** (*mapping*) – Parameter: value mapping for each argument of the specified decay function. These parameters are bound to the decay function to create a one-argument callable.

- **huff_normalization** (*bool*) – Flag used to normalize demand through Huff-like interaction probabilities.

  Used in 3SFCA to curtail demand over-estimation.

- **suboptimality_exponent** (*float*) – Used in M2SFCA to indicate the extent to account for network suboptimality in access. This parameter allows for the differentiation between two scenarios:

  1. Three demand locations each at a distance of 1.0 mile from the sole provider;
  2. Three demand locations each at a distance of 2.0 miles from the sole provider.

  Values greater than 1.0 for this parameter will result in accessibility scores whose weighted average is less than the overall supply.

static _bind_decay_function_parameters (*decay_function*, *decay_params*)

Bind the given parameters for the decay function.

Returns A one-argument callable that accepts one-dimensional numpy arrays.

Return type callable

_calculate_demand_potentials (*distance_matrix*, *demand_array*)

Calculate the demand potential at each input location.

Returns An array of demand at each supply location.

Return type array

_calculate_interaction_probabilities (*distance_matrix*)

Calculate the demand potential at each input location.

Parameters **distance_matrix** (*np.ndarray (float)*) – A matrix whose entry in row i, column j is the distance between demand point i and supply point j.

Returns A 2D-array of the interaction probabilities between each demand point and supply point.

Return type array

calculate_accessibility_scores (*distance_matrix*, *demand_array=None*, *supply_array=None*)

Calculate accessibility scores from a 2D distance matrix.

Parameters

- **distance_matrix** (*np.ndarray (float)*) – A matrix whose entry in row i, column j is the distance between demand point i and supply point j.

- **demand_array** (*np.array(float) or None*) – A one-dimensional array containing demand multipliers for each demand location. The length of the array must match the number of rows in distance_matrix.
- **supply_array** (*np.array(float) or None*) – A one-dimensional array containing supply multipliers for each supply location. The length of the array must match the number of columns in distance_matrix.

Returns An array of access scores at each demand location.

Return type array

3.2.2 Two-Step Floating Catchment Area (2SFCA)

class aceso.**TwoStepFCA**(*radius*)

Bases: `aceso.gravity.GravityModel`

Represents an instance of the standard Two-Step Floating Catchment Area (2SFCA) model.

Initialize a 2SFCA model with the specified radius.

Parameters **radius** (*float*) – The radius of each floating catchment. Pairs of points further than this distance apart are deemed mutually inaccessible. Points within this radius contribute the full demand amount (with no decay).

3.2.3 Three-Step Floating Catchment Area (3SFCA)

class aceso.**ThreeStepFCA**(*decay_function*, *decay_params*)

Bases: `aceso.gravity.GravityModel`

Represents an instance of the Three-Step Floating Catchment Area (3SFCA) model.

In 3SFCA, the presence of nearby options influences the amount of demand pressure each demand location places on other supply locations. A demand location with many nearby options will not exert the same demand on faraway supply locations as a demand location at the same distance that has no nearby alternatives.

This model is designed to account for this observation and reduce the demand over-estimation that may take place with ordinary 2SFCA.

References

Wan, Neng & Zou, Bin & Sternberg, Troy. (2012). A 3-step floating catchment area method for analyzing spatial access to health services. International Journal of Geographical Information Science. 26. 1073-1089. 10.1080/13658816.2011.624987.

Initialize a gravitational model of spatial accessibility using Huff-like normalization.

Parameters

- **decay_function** (*callable or str*) – If str, the name of a decay function in the `decay` module. Some available names are 'uniform', 'raised_cosine', and 'gaussian_decay'.

  If callable, a vectorized numpy function returning demand dropoffs by distance.
- **decay_params** (*mapping*) – Parameter: value mapping for each argument of the specified decay function. These parameters are bound to the decay function to create a one-argument callable.

3.3 Decay functions

A suite of decay functions to simulate demand dropoff as distance increases.

All decay functions operate on one-dimensional numpy arrays.

`aceso.decay.`**`gaussian_decay`**(*distance_array*, *sigma*)

Transform a measurement array using a normal (Gaussian) distribution.

Some sample values. Measurements are in multiple of `sigma`; decay value are in fractions of the maximum value:

measurement	decay value
0.0	1.0
0.7582	0.75
1.0	0.60647
1.17	0.5
2.0	0.13531

`aceso.decay.`**`get_decay_function`**(*name*)

Return the decay function with the given name.

Parameters **`name`** (*`str`*) – The name of the requested decay function.

Available names:

- `'uniform'`
- `'raised_cosine'`
- `'gaussian'`
- `'parabolic'`

`aceso.decay.`**`parabolic_decay`**(*distance_array*, *scale*)

Transform a measurement array using the Epanechnikov (parabolic) kernel.

Some sample values. Measurements are in multiple of `scale`; decay value are in fractions of the maximum value:

measurement	decay value
0.0	1.0
0.25	0.9375
0.5	0.75
0.75	0.4375
1.0	0.0

`aceso.decay.`**`raised_cosine_decay`**(*distance_array*, *scale*)

Transform a measurement array using a raised cosine distribution.

Some sample values. Measurements are in multiple of `scale`; decay value are in fractions of the maximum value:

measurement	decay value
0.0	1.0
0.25	0.853553
0.5	0.5
0.75	0.146447
1.0	0.0

`aceso.decay.`**`uniform_decay`**(*distance_array*, *scale*)

Transform a measurement array using a uniform distribution.

The output is 1 below the scale parameter and 0 above it.

Some sample values. Measurements are in multiple of `scale`; decay value are in fractions of the maximum value:

measurement	decay value
0.0	1.0
0.25	1.0
0.5	1.0
0.75	1.0
1.0	1.0

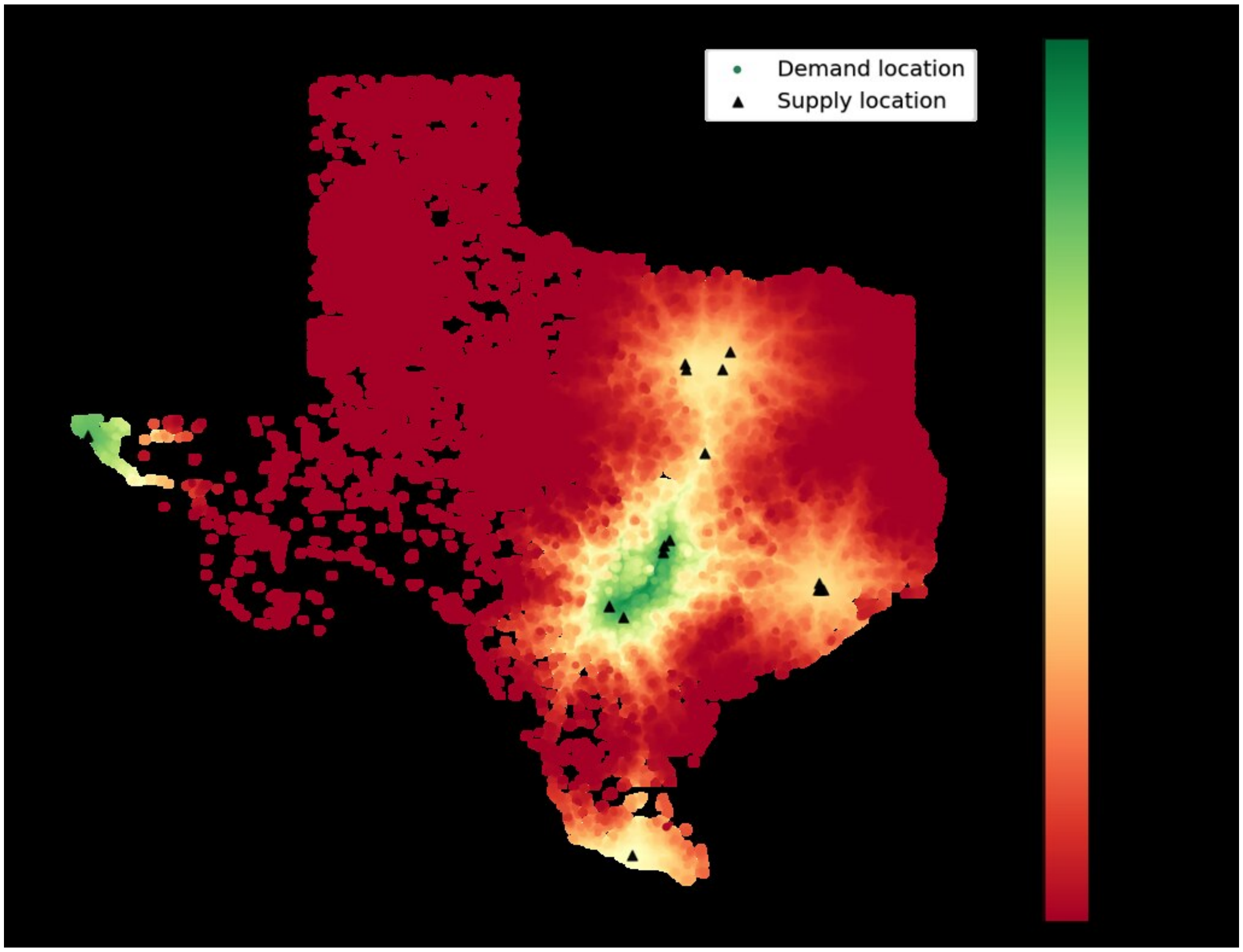
3.4 Contributing

Feature suggestions are welcome, especially ones concerned with alternative models beyond the ones currently implemented. Kindly use GitHub issues to make any such request.

All contributors are expected to follow the code of conduct.

CHAPTER 4

Sample Output



CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index