mgwr Documentation

Release 2.1.1

pysal developers

Jul 18, 2019

CONTENTS:

1 Installation **3**
- 1.1 Installing released version . . . 3
- 1.2 Installing development version . . . 3

2 API reference **5**
- 2.1 GWR Model Estimation and Inference . . . 5
  - 2.1.1 mgwr.gwr.GWR . . . 5
  - 2.1.2 mgwr.gwr.GWRResults . . . 8
  - 2.1.3 mgwr.gwr.GWRResultsLite . . . 11
- 2.2 MGWR Estimation and Inference . . . 12
  - 2.2.1 mgwr.gwr.MGWR . . . 12
  - 2.2.2 mgwr.gwr.MGWRResults . . . 15
- 2.3 Utility Functions . . . 19
  - 2.3.1 Kernel Specification . . . 19
  - 2.3.2 Bandwidth Selection . . . 20
  - 2.3.3 Visualization . . . 23

3 References **25**

Bibliography **27**

Index **29**

mgwr is a Python implementation of multiscale geographically weighted regression for investigating process spatial heterogeneity and scale. It incorporates the widely used approach to modeling process spatial heterogeneity - Geographically Weighted Regression (GWR) as well as the newly proposed approach - Multiscale GWR (MGWR) which relaxes the assumption that all of the processes being modeled operate at the same spatial scale. Inferences are available for both approaches.

CHAPTER

ONE

INSTALLATION

mgwr supports python 3.5 and 3.6 only. Please make sure that you are operating in a python 3 environment.

1.1 Installing released version

mgwr is available on the Python Package Index. Therefore, you can either install directly with *pip* from the command line:

```
pip install -U mgwr
```

or download the source distribution (.tar.gz) and decompress it to your selected destination. Open a command shell and navigate to the decompressed folder. Type:

```
pip install .
```

1.2 Installing development version

Potentially, you might want to use the newest features in the development version of mgwr on github - pysal/mgwr while have not been incorporated in the Pypi released version. You can achieve that by installing pysal/mgwr by running the following from a command shell:

```
pip install https://github.com/pysal/mgwr/archive/master.zip
```

You can also fork the pysal/mgwr repo and create a local clone of your fork. By making changes to your local clone and submitting a pull request to pysal/mgwr, you can contribute to the mgwr development.

CHAPTER

TWO

API REFERENCE

2.1 GWR Model Estimation and Inference

mgwr.gwr.GWR(coords, y, X, bw[, family, ...])	Geographically weighted regression.
mgwr.gwr.GWRResults(model, params, predy, S, ...)	Basic class including common properties for all GWR regression models
mgwr.gwr.GWRResultsLite(model, resid, influ, ...)	Lightweight GWR that computes the minimum diagnostics needed for bandwidth selection

2.1.1 mgwr.gwr.GWR

class mgwr.gwr.**GWR**(*coords*, *y*, *X*, *bw*, *family=<spglm.family.Gaussian object>*, *offset=None*, *sigma2_v1=True*, *kernel='bisquare'*, *fixed=False*, *constant=True*, *spherical=False*, *hat_matrix=False*)

Geographically weighted regression. Can currently estimate Gaussian, Poisson, and logistic models(built on a GLM framework). GWR object prepares model input. Fit method performs estimation and returns a GWRResults object.

Parameters

coords [array-like] n*2, collection of n sets of (x,y) coordinates of observatons; also used as calibration locations is 'points' is set to None

y [array] n*1, dependent variable

X [array] n*k, independent variable, exlcuding the constant

bw [scalar] bandwidth value consisting of either a distance or N nearest neighbors; user specified or obtained using Sel_BW

family [family object] underlying probability model; provides distribution-specific calculations

offset [array] n*1, the offset variable at the ith location. For Poisson model this term is often the size of the population at risk or the expected size of the outcome in spatial epidemiology Default is None where Ni becomes 1.0 for all locations; only for Poisson models

sigma2_v1 [boolean] specify form of corrected denominator of sigma squared to use for model diagnostics; Acceptable options are:

'True': n-tr(S) (defualt) 'False': n-2(tr(S)+tr(S'S))

kernel [string] type of kernel function used to weight observations; available options: 'gaussian' 'bisquare' 'exponential'

fixed [boolean] True for distance based kernel function and False for adaptive (nearest neighbor) kernel function (default)

constant [boolean] True to include intercept (default) in model and False to exclude intercept.

spherical [boolean] True for shperical coordinates (long-lat), False for projected coordinates (defalut).

hat_matrix [boolean] True to store full n by n hat matrix, False to not store full hat matrix to minimize memory footprint (defalut).

Examples

#basic model calibration

```
>>> import libpysal as ps
>>> from mgwr.gwr import GWR
>>> data = ps.io.open(ps.examples.get_path('GData_utm.csv'))
>>> coords = list(zip(data.by_col('X'), data.by_col('Y')))
>>> y = np.array(data.by_col('PctBach')).reshape((-1,1))
>>> rural = np.array(data.by_col('PctRural')).reshape((-1,1))
>>> pov = np.array(data.by_col('PctPov')).reshape((-1,1))
>>> african_amer = np.array(data.by_col('PctBlack')).reshape((-1,1))
>>> X = np.hstack([rural, pov, african_amer])
>>> model = GWR(coords, y, X, bw=90.000, fixed=False, kernel='bisquare')
>>> results = model.fit()
>>> print(results.params.shape)
(159, 4)
```

#predict at unsampled locations

```
>>> index = np.arange(len(y))
>>> test = index[-10:]
>>> X_test = X[test]
>>> coords_test = np.array(coords)[test]
>>> model = GWR(coords, y, X, bw=94, fixed=False, kernel='bisquare')
>>> results = model.predict(coords_test, X_test)
>>> print(results.params.shape)
(10, 4)
```

Attributes

coords [array-like] n*2, collection of n sets of (x,y) coordinates used for calibration locations

y [array] n*1, dependent variable

X [array] n*k, independent variable, exlcuding the constant

bw [scalar] bandwidth value consisting of either a distance or N nearest neighbors; user specified or obtained using Sel_BW

family [family object] underlying probability model; provides distribution-specific calculations

offset [array] n*1, the offset variable at the ith location. For Poisson model this term is often the size of the population at risk or the expected size of the outcome in spatial epidemiology Default is None where Ni becomes 1.0 for all locations

sigma2_v1 [boolean] specify form of corrected denominator of sigma squared to use for model diagnostics; Acceptable options are:

'True': n-tr(S) (defualt) 'False': n-2(tr(S)+tr(S'S))

kernel [string] type of kernel function used to weight observations; available options: 'gaussian' 'bisquare' 'exponential'

fixed [boolean] True for distance based kernel function and False for adaptive (nearest neighbor) kernel function (default)

constant [boolean] True to include intercept (default) in model and False to exclude intercept

spherical [boolean] True for shperical coordinates (long-lat), False for projected coordinates (defalut).

hat_matrix [boolean] True to store full n by n hat matrix, False to not store full hat matrix to minimize memory footprint (defalut).

n [integer] number of observations

k [integer] number of independent variables

mean_y [float] mean of y

std_y [float] standard deviation of y

fit_params [dict] parameters passed into fit method to define estimation routine

points [array-like] n*2, collection of n sets of (x,y) coordinates used for calibration locations instead of all observations; defaults to None unles specified in predict method

P [array] n*k, independent variables used to make prediction; exlcuding the constant; default to None unless specified in predict method

exog_scale [scalar] estimated scale using sampled locations; defualt is None unless specified in predict method

exog_resid [array-like] estimated residuals using sampled locations; defualt is None unless specified in predict method

Methods

fit(self[, ini_params, tol, max_iter, ...])	Method that fits a model with a particular estimation routine.
predict(self, points, P[, exog_scale, ...])	Method that predicts values of the dependent variable at un-sampled locations

mgwr.gwr.GWR

GWR.**fit**(*self*, *ini_params=None*, *tol=1e-05*, *max_iter=20*, *solve='iwls'*, *lite=False*, *pool=None*)

Method that fits a model with a particular estimation routine.

Parameters

ini_betas [array, optional] k*1, initial coefficient values, including constant. Default is None, which calculates initial values during estimation.

tol: float, optional Tolerence for estimation convergence. Default is 1.0e-5.

max_iter [integer, optional] Maximum number of iterations if convergence not achieved. Default is 20.

solve [string, optional] Technique to solve MLE equations. Default is 'iwls', meaning iteratively (re)weighted least squares.

lite [bool, optional] Whether to estimate a lightweight GWR that computes the minimum diagnostics needed for bandwidth selection (could speed up bandwidth selection for GWR) or to estimate a full GWR. Default is False.

pool [A multiprocessing Pool object to enable parallel fitting; default is None.]

Returns

: If lite=False, return a GWRResult instance; otherwise, return a GWRResultLite instance.

mgwr.gwr.GWR

GWR.**predict**(*self*, *points*, *P*, *exog_scale=None*, *exog_resid=None*, *fit_params={}*)
Method that predicts values of the dependent variable at un-sampled locations

Parameters

points [array-like] n*2, collection of n sets of (x,y) coordinates used for calibration prediction locations

P [array] n*k, independent variables used to make prediction; exlcuding the constant

exog_scale [scalar] estimated scale using sampled locations; defualt is None which estimates a model using points from "coords"

exog_resid [array-like] estimated residuals using sampled locations; defualt is None which estimates a model using points from "coords"; if given it must be n*1 where n is the length of coords

fit_params [dict] key-value pairs of parameters that will be passed into fit method to define estimation routine; see fit method for more details

df_model	
df_resid	

__init__(*self*, *coords*, *y*, *X*, *bw*, *family=<spglm.family.Gaussian object at 0x7eff0aa39828>*, *offset=None*, *sigma2_v1=True*, *kernel='bisquare'*, *fixed=False*, *constant=True*, *spherical=False*, *hat_matrix=False*)
Initialize class

2.1.2 mgwr.gwr.GWRResults

class mgwr.gwr.**GWRResults**(*model*, *params*, *predy*, *S*, *CCT*, *influ*, *tr_STS=None*, *w=None*)
Basic class including common properties for all GWR regression models

Parameters

model [GWR object] pointer to GWR object with estimation parameters

params [array] n*k, estimated coefficients

predy [array] n*1, predicted y values

S [array] n*n, hat matrix

CCT [array] n*k, scaled variance-covariance matrix

w [array] n*1, final weight used for iteratively re-weighted least sqaures; default is None

Attributes

model [GWR Object] points to GWR object for which parameters have been estimated

params [array] n*k, parameter estimates

predy [array] n*1, predicted value of y

y [array] n*1, dependent variable

X [array] n*k, independent variable, including constant

family [family object] underlying probability model; provides distribution-specific calculations

n [integer] number of observations

k [integer] number of independent variables

df_model [integer] model degrees of freedom

df_resid [integer] residual degrees of freedom

offset [array] n*1, the offset variable at the ith location. For Poisson model this term is often the size of the population at risk or the expected size of the outcome in spatial epidemiology; Default is None where Ni becomes 1.0 for all locations

scale [float] sigma squared used for subsequent computations

w [array] n*1, final weights from iteratively re-weighted least sqaures routine

resid_response [array] n*1, residuals of the repsonse

resid_ss [scalar] residual sum of sqaures

W [array] n*n; spatial weights for each observation from each calibration point

S [array] n*n, hat matrix

CCT [array] n*k, scaled variance-covariance matrix

ENP [scalar] effective number of parameters

tr_S [float] trace of S (hat) matrix

tr_STS [float] trace of STS matrix

y_bar [array] weighted mean of y

TSS [array] geographically weighted total sum of squares

RSS [array] geographically weighted residual sum of squares

R2 [float] Global r-squared value for a Gaussian model.

adj_R2 [float] Adjusted global r-squared for a Gaussian model.

aic [float] Akaike information criterion

aicc [float] corrected Akaike information criterion to account to account for model complexity (smaller bandwidths)

bic [float] Bayesian information criterio

localR2 [array] local R square

sigma2 [float] residual variance

std_res [array] standardized residuals

bse [array] standard errors of Betas

influ [array] n*1, leading diagonal of S matrix

CooksD [array] n*1, Cook's D

tvalues [array] Return the t-statistic for a given parameter estimate.

adj_alpha [array] Corrected alpha (critical) values to account for multiple testing during hypothesis testing.

deviance [array] n*1, local model deviance for each calibration point

resid_deviance [array] n*1, local sum of residual deviance for each calibration point

llf [scalar] log-likelihood of the full model; see pysal.contrib.glm.family for damily-sepcific log-likelihoods

pDev [float] Local percentage of deviance accounted for.

D2 [float] Percentage of deviance explanied.

adj_D2 [float] Adjusted percentage of deviance explanied.

mu [array] n*, flat one dimensional array of predicted mean response value from estimator

fit_params [dict] parameters passed into fit method to define estimation routine

predictions [array] p*1, predicted values generated by calling the GWR predict method to predict dependent variable at unsampled points ()

Methods

D2(self)	Percentage of deviance explanied.
ENP(self)	effective number of parameters
R2(self)	Global r-squared value for a Gaussian model.
RSS(self)	geographically weighted residual sum of squares
TSS(self)	geographically weighted total sum of squares
adj_D2(self)	Adjusted percentage of deviance explanied.
adj_R2(self)	Adjusted global r-squared for a Gaussian model.
adj_alpha(self)	Corrected alpha (critical) values to account for multiple testing during hypothesis testing.
bse(self)	standard errors of Betas
conf_int(self)	Returns the confidence interval of the fitted parameters.
cooksD(self)	Influence: leading diagonal of S Matrix
cov_params(self, cov[, exog_scale])	Returns scaled covariance parameters
critical_tval(self[, alpha])	Utility function to derive the critical t-value based on given alpha that are needed for hypothesis testing
filter_tvals(self[, critical_t, alpha])	Utility function to set tvalues with an absolute value smaller than the absolute value of the alpha (critical) value to 0.
localR2(self)	local R square
local_collinearity(self)	Computes several indicators of multicollinearity within a geographically weighted design matrix, including:
pDev(self)	Local percentage of deviance accounted for.
sigma2(self)	residual variance
spatial_variability(self, selector[, ...])	Method to compute a Monte Carlo test of spatial variability for each estimated coefficient surface.

Continued on next page

Table 3 – continued from previous page

std_res(self)	standardized residuals
summary(self)	Print out GWR summary
tr_S(self)	trace of S (hat) matrix
tvalues(self)	Return the t-statistic for a given parameter estimate.
use_t(self)	bool(x) -> bool
y_bar(self)	weighted mean of y

W	
adj_pseudoR2	
aic	
aicc	
bic	
deviance	
df_model	
df_resid	
global_deviance	
initialize	
llf	
llnull	
normalized_cov_params	
null	
null_deviance	
pearson_chi2	
predictions	
pseudoR2	
pvalues	
resid_anscombe	
resid_deviance	
resid_pearson	
resid_response	
resid_ss	
resid_working	
scale	

__init__(*self*, *model*, *params*, *predy*, *S*, *CCT*, *influ*, *tr_STS=None*, *w=None*)
Initialize self. See help(type(self)) for accurate signature.

2.1.3 mgwr.gwr.GWRResultsLite

class mgwr.gwr.**GWRResultsLite**(*model*, *resid*, *influ*, *params*)
Lightweight GWR that computes the minimum diagnostics needed for bandwidth selection

Parameters

model [GWR object] pointer to GWR object with estimation parameters

resid [array] n*1, residuals of the repsonse

influ [array] n*1, leading diagonal of S matrix

Attributes

tr_S [float] trace of S (hat) matrix

llf [scalar] log-likelihood of the full model; see pysal.contrib.glm.family for damily-sepcific log-likelihoods

mu [array] n*, flat one dimensional array of predicted mean response value from estimator

resid_ss [scalar] residual sum of sqaures

Methods

llf	
mu	
predy	
resid_ss	
tr_S	

__init__(*self*, *model*, *resid*, *influ*, *params*)
Initialize self. See help(type(self)) for accurate signature.

2.2 MGWR Estimation and Inference

mgwr.gwr.MGWR(coords, y, X, selector[, ...])	Multiscale GWR estimation and inference.
mgwr.gwr.MGWRResults(model, params, predy, ...)	Class including common properties for a MGWR model.

2.2.1 mgwr.gwr.MGWR

class mgwr.gwr.**MGWR**(*coords*, *y*, *X*, *selector*, *sigma2_v1=True*, *kernel='bisquare'*, *fixed=False*, *constant=True*, *spherical=False*, *hat_matrix=False*)
Multiscale GWR estimation and inference. See [FYK17] [YFL+19].

Parameters

coords [array-like] n*2, collection of n sets of (x,y) coordinates of observatons; also used as calibration locations is 'points' is set to None

y [array] n*1, dependent variable

X [array] n*k, independent variable, exlcuding the constant

selector [sel_bw object] valid sel_bw object that has successfully called the "search" method. This parameter passes on information from GAM model estimation including optimal bandwidths.

family [family object] underlying probability model; provides distribution-specific calculations

sigma2_v1 [boolean] specify form of corrected denominator of sigma squared to use for model diagnostics; Acceptable options are:

'True': n-tr(S) (defualt) 'False': n-2(tr(S)+tr(S'S))

kernel [string] type of kernel function used to weight observations; available options: 'gaussian' 'bisquare' 'exponential'

fixed [boolean] True for distance based kernel function and False for adaptive (nearest neighbor) kernel function (default)

constant [boolean] True to include intercept (default) in model and False to exclude intercept.

spherical [boolean] True for spherical coordinates (long-lat), False for projected coordinates (defalut).

hat_matrix [boolean] True for computing and storing covariate-specific hat matrices R (n,n,k) and model hat matrix S (n,n). False (default) for computing MGWR inference on the fly.

Examples

#basic model calibration

```
>>> import libpysal as ps
>>> from mgwr.gwr import MGWR
>>> from mgwr.sel_bw import Sel_BW
>>> data = ps.io.open(ps.examples.get_path('GData_utm.csv'))
>>> coords = list(zip(data.by_col('X'), data.by_col('Y')))
>>> y = np.array(data.by_col('PctBach')).reshape((-1,1))
>>> rural = np.array(data.by_col('PctRural')).reshape((-1,1))
>>> fb = np.array(data.by_col('PctFB')).reshape((-1,1))
>>> african_amer = np.array(data.by_col('PctBlack')).reshape((-1,1))
>>> X = np.hstack([fb, african_amer, rural])
>>> X = (X - X.mean(axis=0)) / X.std(axis=0)
>>> y = (y - y.mean(axis=0)) / y.std(axis=0)
>>> selector = Sel_BW(coords, y, X, multi=True)
>>> selector.search(multi_bw_min=[2])
[92.0, 101.0, 136.0, 158.0]
>>> model = MGWR(coords, y, X, selector, fixed=False, kernel='bisquare', sigma2_
↪v1=True)
>>> results = model.fit()
>>> print(results.params.shape)
(159, 4)
```

Attributes

coords [array-like] n*2, collection of n sets of (x,y) coordinates of observatons; also used as calibration locations is 'points' is set to None

y [array] n*1, dependent variable

X [array] n*k, independent variable, exlcuding the constant

selector [sel_bw object] valid sel_bw object that has successfully called the "search" method. This parameter passes on information from GAM model estimation including optimal bandwidths.

bw [array-like] collection of bandwidth values consisting of either a distance or N nearest neighbors; user specified or obtained using Sel_BW with fb=True. Order of values should the same as the order of columns associated with X

family [family object] underlying probability model; provides distribution-specific calculations

sigma2_v1 [boolean] specify form of corrected denominator of sigma squared to use for model diagnostics; Acceptable options are:

'True': n-tr(S) (defualt) 'False': n-2(tr(S)+tr(S'S))

kernel [string] type of kernel function used to weight observations; available options: 'gaussian' 'bisquare' 'exponential'

fixed [boolean] True for distance based kernel function and False for adaptive (nearest neighbor) kernel function (default)

constant [boolean] True to include intercept (default) in model and False to exclude intercept.

spherical [boolean] True for shperical coordinates (long-lat), False for projected coordinates (defalut).

n [integer] number of observations

k [integer] number of independent variables

mean_y [float] mean of y

std_y [float] standard deviation of y

fit_params [dict] parameters passed into fit method to define estimation routine

W [array-like] list of n*n arrays, spatial weights matrices for weighting all observations from each calibration point: one for each covariate (k)

Methods

fit(self[, n_chunks, pool])	Compute MGWR inference by chunk to reduce memory footprint.
predict(self)	Not implemented.

mgwr.gwr.MGWR

MGWR.**fit**(*self*, *n_chunks=1*, *pool=None*)

Compute MGWR inference by chunk to reduce memory footprint.

Parameters

n_chunks [integer, optional] A number of chunks parameter to reduce memory usage. e.g. n_chunks=2 should reduce overall memory usage by 2.

pool [A multiprocessing Pool object to enable parallel fitting; default is None.]

Returns

: MGWRResults

mgwr.gwr.MGWR

MGWR.**predict**(*self*)

Not implemented.

df_model	
df_resid	

__init__(*self*, *coords*, *y*, *X*, *selector*, *sigma2_v1=True*, *kernel='bisquare'*, *fixed=False*, *constant=True*, *spherical=False*, *hat_matrix=False*)

Initialize class

2.2.2 mgwr.gwr.MGWRResults

class mgwr.gwr.**MGWRResults**(*model*, *params*, *predy*, *CCT*, *ENP_j*, *w*, *R*)

Class including common properties for a MGWR model.

Parameters

model [MGWR object] pointer to MGWR object with estimation parameters

params [array] n*k, estimated coefficients

predy [array] n*1, predicted y values

S [array] n*n, model hat matrix (if MGWR(hat_matrix=True))

R [array] n*n*k, covariate-specific hat matrices (if MGWR(hat_matrix=True))

CCT [array] n*k, scaled variance-covariance matrix

w [array] n*1, final weight used for iteratively re-weighted least sqaures; default is None

Attributes

model [GWR Object] points to GWR object for which parameters have been estimated

params [array] n*k, parameter estimates

predy [array] n*1, predicted value of y

y [array] n*1, dependent variable

X [array] n*k, independent variable, including constant

family [family object] underlying probability model; provides distribution-specific calculations

n [integer] number of observations

k [integer] number of independent variables

df_model [integer] model degrees of freedom

df_resid [integer] residual degrees of freedom

scale [float] sigma squared used for subsequent computations

w [array] n*1, final weights from iteratively re-weighted least sqaures routine

resid_response [array] n*1, residuals of the repsonse

resid_ss [scalar] residual sum of sqaures

W [array-like] list of n*n arrays, spatial weights matrices for weighting all observations from each calibration point: one for each covariate (k)

S [array] n*n, model hat matrix (if MGWR(hat_matrix=True))

R [array] n*n*k, covariate-specific hat matrices (if MGWR(hat_matrix=True))

CCT [array] n*k, scaled variance-covariance matrix

ENP [scalar] effective number of parameters

ENP_j [array-like] effective number of paramters, which depends on sigma2, for each covariate in the model

adj_alpha [array] Corrected alpha (critical) values to account for multiple testing during hypothesis testing.

adj_alpha_j [array] Corrected alpha (critical) values to account for multiple testing during hypothesis testing.

tr_S [float] trace of S (hat) matrix

tr_STS [float] trace of STS matrix

R2 [float] Global r-squared value for a Gaussian model.

adj_R2 [float] Adjusted global r-squared for a Gaussian model.

aic [float] Akaike information criterion

aicc [float] corrected Akaike information criterion to account to account for model complexity (smaller bandwidths)

bic [float] Bayesian information criterio

sigma2 [float] residual variance

std_res [array] standardized residuals

bse [array] standard errors of Betas

influ [array] n*1, leading diagonal of S matrix

CooksD [array] n*1, Cook's D

tvalues [array] Return the t-statistic for a given parameter estimate.

llf [scalar] log-likelihood of the full model; see pysal.contrib.glm.family for damily-sepcific log-likelihoods

mu [array] n*, flat one dimensional array of predicted mean response value from estimator

Methods

D2(self)	Percentage of deviance explanied.
ENP(self)	effective number of parameters
R2(self)	Global r-squared value for a Gaussian model.
RSS(self)	geographically weighted residual sum of squares
TSS(self)	geographically weighted total sum of squares
adj_D2(self)	Adjusted percentage of deviance explanied.
adj_R2(self)	Adjusted global r-squared for a Gaussian model.
adj_alpha(self)	Corrected alpha (critical) values to account for multiple testing during hypothesis testing.
adj_alpha_j(self)	Corrected alpha (critical) values to account for multiple testing during hypothesis testing.
bse(self)	standard errors of Betas
conf_int(self)	Returns the confidence interval of the fitted parameters.
cooksD(self)	Influence: leading diagonal of S Matrix
cov_params(self, cov[, exog_scale])	Returns scaled covariance parameters
critical_tval(self[, alpha])	Utility function to derive the critial t-value based on given alpha that are needed for hypothesis testing
filter_tvals(self[, critical_t, alpha])	Utility function to set tvalues with an absolute value smaller than the absolute value of the alpha (critical) value to 0.

Continued on next page

Table 6 – continued from previous page

localR2(self)	local R square
local_collinearity(self)	Computes several indicators of multicollinearity within a geographically weighted design matrix, including:
pDev(self)	Local percentage of deviance accounted for.
sigma2(self)	residual variance
spatial_variability(self, selector[, ...])	Method to compute a Monte Carlo test of spatial variability for each estimated coefficient surface.
std_res(self)	standardized residuals
summary(self)	Print out MGWR summary
tr_S(self)	trace of S (hat) matrix
tvalues(self)	Return the t-statistic for a given parameter estimate.
use_t(self)	bool(x) -> bool
y_bar(self)	weighted mean of y

mgwr.gwr.MGWRResults

MGWRResults.**critical_tval**(*self*, *alpha=None*)

Utility function to derive the critial t-value based on given alpha that are needed for hypothesis testing

Parameters

alpha [scalar] critical value to determine which tvalues are associated with statistically significant parameter estimates. Default to None in which case the adjusted alpha value at the 95 percent CI is automatically used.

Returns

critical [scalar] critical t-val based on alpha

mgwr.gwr.MGWRResults

MGWRResults.**filter_tvals**(*self*, *critical_t=None*, *alpha=None*)

Utility function to set tvalues with an absolute value smaller than the absolute value of the alpha (critical) value to 0. If critical_t is supplied than it is used directly to filter. If alpha is provided than the critical t value will be derived and used to filter. If neither are critical_t nor alpha are provided, an adjusted alpha at the 95 percent CI will automatically be used to define the critical t-value and used to filter. If both critical_t and alpha are supplied then the alpha value will be ignored.

Parameters

critical [scalar] critical t-value to determine whether parameters are statistically significant

alpha [scalar] alpha value to determine which tvalues are associated with statistically significant parameter estimates

Returns

filtered [array] n*k; new set of n tvalues for each of k variables where absolute tvalues less than the absolute value of alpha have been set to 0.

mgwr.gwr.MGWRResults

MGWRResults.**local_collinearity**(*self*)

Computes several indicators of multicollinearity within a geographically weighted design matrix, including:

local condition number (n, 1) local variance-decomposition proportions (n, p)

Returns four arrays with the order and dimensions listed above where n is the number of locations used as calibrations points and p is the nubmer of explanatory variables

mgwr.gwr.MGWRResults

MGWRResults.**spatial_variability**(*self*, *selector*, *n_iters=1000*, *seed=None*)

Method to compute a Monte Carlo test of spatial variability for each estimated coefficient surface.

WARNING: This test is very computationally demanding!

Parameters

selector [sel_bw object] should be the sel_bw object used to select a bandwidth for the gwr model that produced the surfaces that are being tested for spatial variation

n_iters [int] the number of Monte Carlo iterations to include for the tests of spatial variability.

seed [int] optional parameter to select a custom seed to ensure stochastic results are replicable. Default is none which automatically sets the seed to 5536

Returns

p values [list] a list of psuedo p-values that correspond to the model parameter surfaces. Allows us to assess the probability of obtaining the observed spatial variation of a given surface by random chance.

mgwr.gwr.MGWRResults

MGWRResults.**summary**(*self*)

Print out MGWR summary

W	
adj_pseudoR2	
aic	
aicc	
bic	
deviance	
df_model	
df_resid	
global_deviance	
initialize	
llf	
llnull	
normalized_cov_params	
null	
null_deviance	
pearson_chi2	
predictions	
pseudoR2	
pvalues	
resid_anscombe	
resid_deviance	
resid_pearson	
resid_response	
resid_ss	
resid_working	
scale	

__init__(*self*, *model*, *params*, *predy*, *CCT*, *ENP_j*, *w*, *R*)
Initialize class

2.3 Utility Functions

2.3.1 Kernel Specification

mgwr.kernels.Kernel(i, data[, bw, fixed, ...])	GWR kernel function specifications.
mgwr.kernels.local_cdist(coords_i, coords, ...)	Compute Haversine (spherical=True) or Euclidean (spherical=False) distance for a local kernel.

mgwr.kernels.Kernel

class mgwr.kernels.**Kernel**(*i*, *data*, *bw=None*, *fixed=True*, *function='triangular'*, *eps=1.0000001*, *ids=None*, *points=None*, *spherical=False*)
GWR kernel function specifications.

__init__(*self*, *i*, *data*, *bw=None*, *fixed=True*, *function='triangular'*, *eps=1.0000001*, *ids=None*, *points=None*, *spherical=False*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(self, i, data[, bw, fixed, ...])	Initialize self.

mgwr.kernels.local_cdist

kernels.**local_cdist**(*coords_i*, *coords*, *spherical*) **= <function local_cdist>**

2.3.2 Bandwidth Selection

mgwr.sel_bw.Sel_BW(coords, y, X_loc[, ...])	Select bandwidth for kernel

mgwr.sel_bw.Sel_BW

class mgwr.sel_bw.**Sel_BW**(*coords*, *y*, *X_loc*, *X_glob=None*, *family=<spglm.family.Gaussian object>*, *offset=None*, *kernel='bisquare'*, *fixed=False*, *multi=False*, *constant=True*, *spherical=False*)

Select bandwidth for kernel

Methods: p211 - p213, bandwidth selection

[FBC02]: Fotheringham, A. S., Brunsdon, C., & Charlton, M. (2002). Geographically weighted regression: the analysis of spatially varying relationships.

Parameters

y [array] n*1, dependent variable.

X_glob [array] n*k1, fixed independent variable.

X_loc [array] n*k2, local independent variable, including constant.

coords [list of tuples] (x,y) of points used in bandwidth selection

family [family object/instance, optional] underlying probability model: Gaussian(), Poisson(), Binomial(). Default is Gaussian().

offset [array] n*1, the offset variable at the ith location. For Poisson model this term is often the size of the population at risk or the expected size of the outcome in spatial epidemiology Default is None where Ni becomes 1.0 for all locations

kernel [string, optional] kernel function: 'gaussian', 'bisquare', 'exponential'. Default is 'bisquare'.

fixed [boolean] True for fixed bandwidth and False for adaptive (NN)

multi [True for multiple (covaraite-specific) bandwidths] False for a traditional (same for all covariates) bandwdith; defualt is False.

constant [boolean] True to include intercept (default) in model and False to exclude intercept.

spherical [boolean] True for shperical coordinates (long-lat), False for projected coordinates (defalut).

Examples

```
>>> import libpysal as ps
>>> from mgwr.sel_bw import Sel_BW
>>> data = ps.io.open(ps.examples.get_path('GData_utm.csv'))
>>> coords = list(zip(data.by_col('X'), data.by_col('Y')))
>>> y = np.array(data.by_col('PctBach')).reshape((-1,1))
>>> rural = np.array(data.by_col('PctRural')).reshape((-1,1))
>>> pov = np.array(data.by_col('PctPov')).reshape((-1,1))
>>> african_amer = np.array(data.by_col('PctBlack')).reshape((-1,1))
>>> X = np.hstack([rural, pov, african_amer])
```

Golden section search AICc - adaptive bisquare

```
>>> bw = Sel_BW(coords, y, X).search(criterion='AICc')
>>> print(bw)
93.0
```

Golden section search AIC - adaptive Gaussian

```
>>> bw = Sel_BW(coords, y, X, kernel='gaussian').search(criterion='AIC')
>>> print(bw)
50.0
```

Golden section search BIC - adaptive Gaussian

```
>>> bw = Sel_BW(coords, y, X, kernel='gaussian').search(criterion='BIC')
>>> print(bw)
62.0
```

Golden section search CV - adaptive Gaussian

```
>>> bw = Sel_BW(coords, y, X, kernel='gaussian').search(criterion='CV')
>>> print(bw)
68.0
```

Interval AICc - fixed bisquare

```
>>> sel = Sel_BW(coords, y, X, fixed=True)
>>> bw = sel.search(search_method='interval', bw_min=211001.0, bw_max=211035.0,␣
↪interval=2)
>>> print(bw)
211025.0
```

Attributes

y [array] n*1, dependent variable.

X_glob [array] n*k1, fixed independent variable.

X_loc [array] n*k2, local independent variable, including constant.

coords [list of tuples] (x,y) of points used in bandwidth selection

family [string] GWR model type: ‘Gaussian’, ‘logistic, ‘Poisson’‘

kernel [string] type of kernel used and wether fixed or adaptive

fixed [boolean] True for fixed bandwidth and False for adaptive (NN)

criterion [string] bw selection criterion: 'AICc', 'AIC', 'BIC', 'CV'

search_method [string] bw search method: 'golden', 'interval'

bw_min [float] min value used in bandwidth search

bw_max [float] max value used in bandwidth search

interval [float] interval increment used in interval search

tol [float] tolerance used to determine convergence

max_iter [integer] max interations if no convergence to tol

multi [True for multiple (covaraite-specific) bandwidths] False for a traditional (same for all covariates) bandwdith; defualt is False.

constant [boolean] True to include intercept (default) in model and False to exclude intercept.

offset [array] n*1, the offset variable at the ith location. For Poisson model this term is often the size of the population at risk or the expected size of the outcome in spatial epidemiology Default is None where Ni becomes 1.0 for all locations

spherical [boolean] True for shperical coordinates (long-lat), False for projected coordinates (defalut).

search_params [dict] stores search arguments

int_score [boolan] True if adaptive bandwidth is being used and bandwdith selection should be discrete. False if fixed bandwidth is being used and bandwidth does not have to be discrete.

bw [scalar or array-like] Derived optimal bandwidth(s). Will be a scalar for GWR (multi=False) and a list of scalars for MGWR (multi=True) with one bandwidth for each covariate.

S [array] n*n, hat matrix derived from the iterative backfitting algorthim for MGWR during bandwidth selection

R [array] n*n*k, partial hat matrices derived from the iterative backfitting algoruthm for MGWR during bandwidth selection. There is one n*n matrix for each of the k covariates.

params [array] n*k, calibrated parameter estimates for MGWR based on the iterative backfitting algorithm - computed and saved here to avoid having to do it again in the MGWR object.

Methods

search(self[, search_method, criterion, ...])	Method to select one unique bandwidth for a gwr model or a bandwidth vector for a mgwr model.

mgwr.sel_bw.Sel_BW

Sel_BW.**search**(*self, search_method='golden_section', criterion='AICc', bw_min=None, bw_max=None, interval=0.0, tol=1e-06, max_iter=200, init_multi=None, tol_multi=1e-05, rss_score=False, max_iter_multi=200, multi_bw_min=[None], multi_bw_max=[None], bws_same_times=3, pool=None, verbose=False*)

Method to select one unique bandwidth for a gwr model or a bandwidth vector for a mgwr model.

Parameters

criterion [string] bw selection criterion: 'AICc', 'AIC', 'BIC', 'CV'

search_method [string] bw search method: 'golden', 'interval'

bw_min [float] min value used in bandwidth search

bw_max [float] max value used in bandwidth search

multi_bw_min [list] min values used for each covariate in mgwr bandwidth search. Must be either a single value or have one value for each covariate including the intercept

multi_bw_max [list] max values used for each covariate in mgwr bandwidth search. Must be either a single value or have one value for each covariate including the intercept

interval [float] interval increment used in interval search

tol [float] tolerance used to determine convergence

max_iter [integer] max iterations if no convergence to tol

init_multi [float] None (default) to initialize MGWR with a bandwidth derived from GWR. Otherwise this option will choose the bandwidth to initialize MGWR with.

tol_multi [convergence tolerence for the multiple bandwidth] backfitting algorithm; a larger tolerance may stop the algorith faster though it may result in a less optimal model

max_iter_multi [max iterations if no convergence to tol for multiple] bandwidth backfitting algorithm

rss_score [True to use the residual sum of sqaures to evaluate] each iteration of the multiple bandwidth backfitting routine and False to use a smooth function; default is False

bws_same_times [If bandwidths keep the same between iterations for] bws_same_times (default 3) in backfitting, then use the current set of bandwidths as final bandwidths.

pool [A multiprocessing Pool object to enbale parallel fitting;] default is None

verbose [Boolean] If true, bandwidth searching history is printed out; default is False.

Returns

bw [scalar or array] optimal bandwidth value or values; returns scalar for multi=False and array for multi=True; ordering of bandwidths matches the ordering of the covariates (columns) of the designs matrix, X

__init__(*self*, *coords*, *y*, *X_loc*, *X_glob=None*, *family=<spglm.family.Gaussian object at 0x7efeeeb6d080>*, *offset=None*, *kernel='bisquare'*, *fixed=False*, *multi=False*, *constant=True*, *spherical=False*)

Initialize self. See help(type(self)) for accurate signature.

2.3.3 Visualization

utils.shift_colormap(cmap[, start, ...])	Function to offset the "center" of a colormap.
utils.truncate_colormap(cmap[, minval, ...])	Function to truncate a colormap by selecting a subset of the original colormap's values
utils.compare_surfaces(*args, **kwargs)	

mgwr.utils.shift_colormap

mgwr.utils.**shift_colormap**(*cmap*, *start=0*, *midpoint=0.5*, *stop=1.0*, *name='shiftedcmap'*)

Function to offset the "center" of a colormap. Useful for data with a negative min and positive max and you want the middle of the colormap's dynamic range to be at zero

Parameters

cmap [The matplotlib colormap to be altered]

start [Offset from lowest point in the colormap's range.] Defaults to 0.0 (no lower ofset). Should be between 0.0 and *midpoint*.

midpoint [The new center of the colormap. Defaults to] 0.5 (no shift). Should be between 0.0 and 1.0. In general, this should be 1 - vmax/(vmax + abs(vmin)) For example if your data range from -15.0 to +5.0 and you want the center of the colormap at 0.0, *midpoint* should be set to 1 - 5/(5 + 15)) or 0.75

stop [Offset from highets point in the colormap's range.] Defaults to 1.0 (no upper ofset). Should be between *midpoint* and 1.0.

Returns

new_cmap [A new colormap that has been shifted.]

mgwr.utils.truncate_colormap

mgwr.utils.**truncate_colormap**(*cmap*, *minval=0.0*, *maxval=1.0*, *n=100*)

Function to truncate a colormap by selecting a subset of the original colormap's values

Parameters

cmap [Mmatplotlib colormap to be altered]

minval [Minimum value of the original colormap to include in the truncated colormap]

maxval [Maximum value of the original colormap to include in the truncated colormap]

n [Number of intervals between the min and max values for the gradient of the truncated colormap]

Returns

new_cmap [A new colormap that has been shifted.]

mgwr.utils.compare_surfaces

mgwr.utils.**compare_surfaces**(**args*, ***kwargs*)

CHAPTER

THREE

REFERENCES

BIBLIOGRAPHY

[BKW80] D. A. Belsey, E. Kuh, and R. E. Welsch. *Regression Diagnostics: Identifying Influential Data and Sources of Collinearity*. Wiley, New York, 1980.

[BFC99] Chris Brunsdon, A Stewart Fotheringham, and Martin Charlton. Some notes on parametric significance tests for geographically weighted regression. *Journal of Regional Science*, 39(3):497–524, 1999.

[BFC08] Chris Brunsdon, A Stewart Fotheringham, and Martin Charlton. Geographically weighted regression: a method for exploring spatial nonstationarity. *Encyclopedia of Geographic Information Science*, pages 558, 2008.

[dSF16] Alan Ricardo da Silva and A. Stewart Fotheringham. The multiple testing issue in geographically weighted regression. *Geographical Analysis*, 48(3):233–247, 2016. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/gean.12084, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/gean.12084, doi:10.1111/gean.12084.

[FB99] A Stewart Fotheringham and Chris Brunsdon. Local forms of spatial analysis. *Geographical Analysis*, 31(4):340–358, 1999.

[FBC02] A. Stewart Fotheringham, Chris Brunsdon, and Martin Charlton. *Geographically Weighted Regression: The Analysis of Spatially Varying Relationships*. John Wiley & Sons, February 2002. ISBN 978-0-470-85525-6.

[FO16] A. Stewart Fotheringham and Taylor M. Oshan. Geographically weighted regression and multicollinearity: dispelling the myth. *Journal of Geographical Systems*, 18(4):303–329, 2016. URL: http://dx.doi.org/10.1007/s10109-016-0239-5, doi:10.1007/s10109-016-0239-5.

[FYK17] A. Stewart Fotheringham, Wenbai Yang, and Wei Kang. Multiscale geographically weighted regression (mgwr). *Annals of the American Association of Geographers*, 107(6):1247–1265, 2017. URL: http://dx.doi.org/10.1080/24694452.2017.1352480, arXiv:http://dx.doi.org/10.1080/24694452.2017.1352480, doi:10.1080/24694452.2017.1352480.

[HFCC10] P. Harris, A. S. Fotheringham, R. Crespo, and M. Charlton. The Use of Geographically Weighted Regression for Spatial Prediction: An Evaluation of Models Using Simulated Data Sets. *Mathematical Geosciences*, 42(6):657–680, June 2010. URL: http://link.springer.com/article/10.1007/s11004-010-9284-7, doi:10.1007/s11004-010-9284-7.

[NFBC05] T Nakaya, AS Fotheringham, Chris Brunsdon, and Martin Charlton. Geographically weighted poisson regression for disease association mapping. *Statistics in Medicine*, 24(17):2695–2717, 2005.

[OF17] Taylor M. Oshan and A. Stewart Fotheringham. A Comparison of Spatially Varying Regression Coefficient Estimates Using Geographically Weighted and Spatial-Filter-Based Techniques: A Comparison of Spatially Varying Regression. *Geographical Analysis*, June 2017. URL: http://doi.wiley.com/10.1111/gean.12133, doi:10.1111/gean.12133.

[Whe07] David C. Wheeler. Diagnostic Tools and a Remedial Method for Collinearity in Geographically Weighted Regression. *Environment and Planning A*, 39(10):2464–2481, October 2007. URL: http://epn.sagepub.com/content/39/10/2464, doi:10.1068/a38325.

[YFL+19] Hanchen Yu, Alexander Stewart Fotheringham, Ziqi Li, Taylor Oshan, Wei Kang, and Levi John Wolf. Inference in multiscale geographically weighted regression. *Geographical Analysis*, 2019. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/gean.12189, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/gean.12189, doi:10.1111/gean.12189.

INDEX

Symbols

__init__() (*mgwr.gwr.GWR method*), 8
__init__() (*mgwr.gwr.GWRResults method*), 11
__init__() (*mgwr.gwr.GWRResultsLite method*), 12
__init__() (*mgwr.gwr.MGWR method*), 14
__init__() (*mgwr.gwr.MGWRResults method*), 19
__init__() (*mgwr.kernels.Kernel method*), 19
__init__() (*mgwr.sel_bw.Sel_BW method*), 23

C

compare_surfaces() (*in module mgwr.utils*), 24
critical_tval() (*mgwr.gwr.MGWRResults method*), 17

F

filter_tvals() (*mgwr.gwr.MGWRResults method*), 17
fit() (*mgwr.gwr.GWR method*), 7
fit() (*mgwr.gwr.MGWR method*), 14

G

GWR (*class in mgwr.gwr*), 5
GWRResults (*class in mgwr.gwr*), 8
GWRResultsLite (*class in mgwr.gwr*), 11

K

Kernel (*class in mgwr.kernels*), 19

L

local_cdist (*mgwr.kernels attribute*), 20
local_collinearity() (*mgwr.gwr.MGWRResults method*), 17

M

MGWR (*class in mgwr.gwr*), 12
MGWRResults (*class in mgwr.gwr*), 15

P

predict() (*mgwr.gwr.GWR method*), 8
predict() (*mgwr.gwr.MGWR method*), 14

S

search() (*mgwr.sel_bw.Sel_BW method*), 22
Sel_BW (*class in mgwr.sel_bw*), 20
shift_colormap() (*in module mgwr.utils*), 23
spatial_variability() (*mgwr.gwr.MGWRResults method*), 18
summary() (*mgwr.gwr.MGWRResults method*), 18

T

truncate_colormap() (*in module mgwr.utils*), 24