

Binomial_MGWR_univariate_check

May 14, 2020

Notebook Outline:

- Section 0.0.1
- Section 0.0.2
- Section 0.0.3
- Section 0.0.4
- Section 0.0.5
 - Section 0.0.6
 - Section 0.0.7

0.0.1 Set up Cells

```
In [1]: import sys
        sys.path.append("C:/Users/msachde1/Downloads/Research/Development/mgwr")

In [2]: import warnings
        warnings.filterwarnings("ignore")
        import pandas as pd
        import numpy as np

        from mgwr.gwr import GWR
        from spglm.family import Gaussian, Binomial, Poisson
        from mgwr.gwr import MGWR
        from mgwr.sel_bw import Sel_BW
        import multiprocessing as mp
        pool = mp.Pool()
        from scipy import linalg
        import numpy.linalg as la
        from scipy import sparse as sp
        from scipy.sparse import linalg as spla
        from spreg.utils import spdots, spmultiply
        from scipy import special
        import libpysal as ps
        import seaborn as sns
        import matplotlib.pyplot as plt
        from copy import deepcopy
        import copy
        from collections import namedtuple
        import spglm
```

0.0.2 Fundamental equations for Binomial MGWR

$$l = \log_b(p/(1-p)) = (\sum \beta_k x_{k,i}) \quad (1)$$

$$(2)$$

where $x_{k,1}$ is the k th explanatory variable in place i , β_k s are the parameters and p is the probability such that $p = P(Y = 1)$.

By exponentiating the log-odds:

$$p/(1-p) = b^{0+1x_1+2x_2}$$

It follows from this - the probability that $Y = 1$ is:

$$p = (b^{0+1x_1+2x_2}) / (b^{0+1x_1+2x_2} + 1) = 1 / (1 + b^{-0+1x_1+2x_2})$$

0.0.3 Example Dataset

Clearwater data - downloaded from link: <https://sgsup.asu.edu/sparc/multiscale-gwr>

```
In [3]: data_p = pd.read_csv("C:/Users/msachde1/Downloads/logistic_mgwr_data/landslides.csv")
        coords = list(zip(data_p['X'], data_p['Y']))
        y = np.array(data_p['Landslid']).reshape((-1,1))
        elev = np.array(data_p['Elev']).reshape((-1,1))
        slope = np.array(data_p['Slope']).reshape((-1,1))
        SinAspct = np.array(data_p['SinAspct']).reshape(-1,1)
        CosAspct = np.array(data_p['CosAspct']).reshape(-1,1)
        X = np.hstack([elev, slope, SinAspct, CosAspct])
        x = slope

        X_std = (X-X.mean(axis=0))/X.std(axis=0)
        x_std = (x-x.mean(axis=0))/x.std(axis=0)
        y_std = (y-y.mean(axis=0))/y.std(axis=0)
```

0.0.4 Helper functions

Hardcoded here for simplicity in the notebook workflow

Please note: A separate `bw_func_b` will not be required when changes will be made in the repository

```
In [6]: kernel='bisquare'
        fixed=False
        spherical=False
        search_method='golden_section'
        criterion='AICc'
        interval=None
        tol=1e-06
        max_iter=200
        X_glob=[]

In [7]: def gwr_func(y, X, bw, family=Gaussian(), offset=None):
        return GWR(coords, y, X, bw, family, offset, kernel=kernel,
```

```

        fixed=fixed, constant=False,
        spherical=spherical, hat_matrix=False).fit(
            lite=True, pool=pool)

def bw_func_b(coords,y, X):
    selector = Sel_BW(coords,y, X,family=Binomial(),offset=None, X_glob=[],
        kernel=kernel, fixed=fixed,
        constant=False, spherical=spherical)
    return selector

def bw_func_p(coords,y, X):
    selector = Sel_BW(coords,y, X,family=Poisson(),offset=off, X_glob=[],
        kernel=kernel, fixed=fixed,
        constant=False, spherical=spherical)
    return selector

def bw_func(coords,y,X):
    selector = Sel_BW(coords,y,X,X_glob=[],
        kernel=kernel, fixed=fixed,
        constant=False, spherical=spherical)
    return selector

def sel_func(bw_func, bw_min=None, bw_max=None):
    return bw_func.search(
        search_method=search_method, criterion=criterion,
        bw_min=bw_min, bw_max=bw_max, interval=interval, tol=tol,
        max_iter=max_iter, pool=pool, verbose=False)

```

0.0.5 Univariate example

GWR model with independent variable, x = slope

```
In [14]: bw_gwbr=Sel_BW(coords,y_std,x_std,family=Binomial(),constant=False).search()
```

```
In [15]: gwbr_model=GWR(coords,y_std,x_std,bw=bw_gwbr,family=Binomial(),constant=False).fit()
```

```
In [16]: bw_gwbr
```

```
Out[16]: 198.0
```

MGWR Binomial loop with one independent variable, x = slope

Edited multi_bw function - original function in <https://github.com/pysal/mgwr/blob/master/mgwr/search.py>

```
In [17]: def multi_bw(init,coords,y, X, n, k, family=Gaussian(),offset=None, tol=1e-06, max_iter=
        verbose=False):

    if multi_bw_min==[None]:
```

```

        multi_bw_min = multi_bw_min*X.shape[1]

    if multi_bw_max==[None]:
        multi_bw_max = multi_bw_max*X.shape[1]

    if isinstance(family,spgml.family.Poisson):
        bw = sel_func(bw_func_p(coords,y,X))
        optim_model=gwr_func(y,X,bw,family=Poisson(),offset=offset)
        err = optim_model.resid_response.reshape((-1, 1))
        param = optim_model.params
        #This change for the Poisson model follows from equation (1) above
        XB = offset*np.exp(np.multiply(param, X))

    elif isinstance(family,spgml.family.Binomial):
        bw = sel_func(bw_func_b(coords,y,X))
        optim_model=gwr_func(y,X,bw,family=Binomial())
        err = optim_model.resid_response.reshape((-1, 1))
        param = optim_model.params
        #This change for the Binomial model follows from equation above
        XB = 1/(1+np.exp(-1*np.multiply(optim_model.params,X)))
        #print(XB)

    else:
        bw=sel_func(bw_func(coords,y,X))
        optim_model=gwr_func(y,X,bw)
        err = optim_model.resid_response.reshape((-1, 1))
        param = optim_model.params
        XB = np.multiply(param, X)

    bw_gwr = bw
    XB=XB

    if rss_score:
        rss = np.sum((err)**2)
        iters = 0
        scores = []
        delta = 1e6
        BWs = []
        bw_stable_counter = np.ones(k)
        bws = np.empty(k)

    try:
        from tqdm.auto import tqdm #if they have it, let users have a progress bar
    except ImportError:

        def tqdm(x, desc=''): #otherwise, just passthrough the range
            return x

```

```

for iters in tqdm(range(1, max_iter + 1), desc='Backfitting'):
    new_XB = np.zeros_like(X)
    params = np.zeros_like(X)

    for j in range(k):
        temp_y = XB[:, j].reshape((-1, 1))
        temp_y = temp_y + err
        temp_X = X[:, j].reshape((-1, 1))

        #The step below will not be necessary once the bw_func is changed in the
        if isinstance(family, spglm.family.Poisson):

            bw_class = bw_func_p(coords, temp_y, temp_X)

        elif isinstance(family, spglm.family.Binomial):

            bw_class = bw_func_b(coords, temp_y, temp_X)

        else:

            bw_class = bw_func(coords, temp_y, temp_X)

        if np.all(bw_stable_counter == bws_same_times):
            #If in backfitting, all bws not changing in bws_same_times (default 3)
            bw = bws[j]
        else:
            bw = sel_func(bw_class, multi_bw_min[j], multi_bw_max[j])
            if bw == bws[j]:
                bw_stable_counter[j] += 1
            else:
                bw_stable_counter = np.ones(k)

        #Changed gwr_func to accept family and offset as attributes
        optim_model = gwr_func(temp_y, temp_X, bw, family, offset)
        err = optim_model.resid_response.reshape((-1, 1))
        param = optim_model.params.reshape((-1, ))
        new_XB[:, j] = optim_model.predy.reshape(-1)
        params[:, j] = param
        bws[j] = bw

    num = np.sum((new_XB - XB)**2) / n
    den = np.sum(np.sum(new_XB, axis=1)**2)
    score = (num / den)**0.5
    XB = new_XB

```

```

    if rss_score:
        predy = np.sum(np.multiply(params, X), axis=1).reshape((-1, 1))
        new_rss = np.sum((y - predy)**2)
        score = np.abs((new_rss - rss) / new_rss)
        rss = new_rss
        scores.append(deepcopy(score))
        delta = score
        BWs.append(deepcopy(bws))

    if verbose:
        print("Current iteration:", iters, ",SOC:", np.round(score, 7))
        print("Bandwidths:", ', '.join([str(bw) for bw in bws]))

    if delta < tol:
        break

    print("iters = "+str(iters))
    opt_bws = BWs[-1]
    print("opt_bws = "+str(opt_bws))
    return (opt_bws, np.array(BWs), np.array(scores), params, err, bw_gwr)

```

Running the function with family = Binomial()

```

In [18]: bw_mgwbr = multi_bw(init=None, coords=coords, y=y_std, X=x_std, n=239, k=x.shape[1], fa

iters = 1
opt_bws = [198.]

```

Running without family and offset attributes runs the normal MGWR loop

```

In [18]: bw_mgwr = multi_bw(init=None, coords=coords, y=y_std, X=x_std, n=262, k=x.shape[1])

iters = 1
opt_bws = [125.]

```

0.0.6 Parameter check

Difference in parameters from the GWR - Binomial model and MGWR Binomial model

```

In [21]: (bw_mgwbr[3]==gwbr_model.params).all()

Out[21]: True

```

The parameters are identical

0.0.7 Bandwidths check

```
In [22]: bw_gwbr
```

```
Out[22]: 235.0
```

```
In [23]: bw_mgwbr[0]
```

```
Out[23]: array([235.])
```

The bandwidth from both models is the same