Problem Set #4

Mahyar Ebrahimitorki
mahyar_ebrahimi_torki@yahoo.com

Econ 815 — October 20, 2019

1 MSE

1.1 Interpretation

In this problem set, after reading, cleaning, and preparing the data, I define two indicator function as follows:

$$\hat{\beta} = \arg\max Q(\beta) = \sum_{y=1}^{Y} \sum_{b=1}^{M_y-1} \sum_{b'=b+1}^{M_y} \mathbb{1}\left[f(b,t|\beta) + f\left(b',t'|\beta\right) \geq f\left(b',t|\beta\right) + f\left(b,t'|\beta\right)\right] \tag{1}$$

$$\begin{aligned} \hat{\beta} &= \arg\max Q(\beta) \\ &= \sum_{y=1}^{Y} \sum_{b=1}^{M_y-1} \sum_{b'=b+1}^{M_v} \mathbb{1}\left[f(b,t|\beta) - f\left(b,t'|\beta\right) \geq p_{bt} - p_{b't'} \wedge f\left(b',t'|\beta\right) - f\left(b',t|\beta\right) \geq p_{b'}t' - p_{bt}\right] \end{aligned} \tag{2}$$

where the first one is used for the case of GS algorithm and not considering transfers and the second one is used when we apply the transfers and use BSS algorithm. Also two different payoff functions provided in the prompt are defined as follows:

$$f_m(b,t) = x_{1bm}y_{1tm} + \alpha x_{2bm}y_{1tm} + \beta distance_{btm} + \varepsilon_{btm} \tag{3}$$

$$f_m(b,t) = \delta x_{1bm}y_{1tm} + \alpha x_{2bm}y_{1tm} + \gamma HHI_{tm} + \beta distance_{btm} + \varepsilon_{btm} \tag{4}$$

Estimation results for these four models and algorithms are provided in the results section. We can see that the sign of the coefficients in different models and algorithms are consistent with each other. Specifically, the coefficient for distance is positive in all models meaning that the further away is the target, the more is the payoff. Companies are more interested in expanding their ownership in farther locations.

1.2 Results

```
'''
Model 1, without transfers:
 [-2.87660604  6.82904737]
Model 2, without transfers:
 [ 0.6424948  -5.88855166  0.94272259  4.3610747 ]
EModel 1, with transfers:
 [-1.81009295  9.45846765]
Model 2, with transfers:
 [ 2.46167079 -4.62838499 -1.85642906  9.72866347]
'''
```

1.3 Executable Script

```python
# Mahyar Ebrahimitorki
#mahyar_ebrahimi_torki@yahoo.com

import numpy as np
import pandas as pd
from pandas import DataFrame as df
import itertools
import scipy.optimize as opt
from scipy.optimize import minimize
from scipy.optimize import differential_evolution
import geopy
from geopy import distance

# Read and Clean data
df = pd.read_excel("radio_merger_data.xlsx")

df.price = df.price/1000
df.population_target = df.population_target/1000

# Defining counterfactual dataset for each year:
df2007 = df[df['year']==2007]
df2008 = df[df['year']==2008]

#The number of all possible combinations:
num_comb = len(list(itertools.product(df2007.buyer_id.tolist(), df2007.target_id.tolist()))) + len(list(itertools.product(df2008.buyer_id.tolist(), df2008.target_id.tolist())))
# Any possible combination (match) between the buyer and target (including real/factual and counterfactual data):
B2007 = df2007.loc[:, ['year', 'buyer_id', 'buyer_lat', 'buyer_long', 'num_stations_buyer', 'corp_owner_buyer']]
B2008 = df2008.loc[:, ['year', 'buyer_id', 'buyer_lat', 'buyer_long', 'num_stations_buyer', 'corp_owner_buyer']]
T2007 = df2007.loc[:, ['target_id', 'target_lat', 'target_long', 'price', 'hhi_target', 'population_target']]
T2008 = df2008.loc[:, ['target_id', 'target_lat', 'target_long', 'price', 'hhi_target', 'population_target']]
def cartesian_prod7(B2007, T2007):
    return(B2007.assign(key=1).merge(T2007.assign(key=1), on='key').drop('key', 1))
combinations_2007 = cartesian_prod7(B2007, T2007)
def cartesian_prod8(B2008, T2008):
    return(B2008.assign(key=1).merge(T2008.assign(key=1), on='key').drop('key', 1))
combinations_2008 = cartesian_prod8(B2008, T2008)

# To drop common observations between "any possible combination of buyer and target" (combination_2007 and Combination_2008) and "the factual data in df2007 and df2008"
on = ['year', 'buyer_id', 'buyer_lat', 'buyer_long', 'num_stations_buyer', 'corp_owner_buyer', 'target_id', 'target_lat', 'target_long', 'price', 'hhi_target', 'population_target']
cf2007 = (combinations_2007.merge(df2007[on], on=on, how='left', indicator=True).query('_merge == "left_only"').drop('_merge', 1))

on = ['year', 'buyer_id', 'buyer_lat', 'buyer_long', 'num_stations_buyer', 'corp_owner_buyer', 'target_id', 'target_lat', 'target_long', 'price', 'hhi_target', 'population_target']
cf2008 = (combinations_2008.merge(df2008[on], on=on, how='left', indicator=True).query('_merge == "left_only"').drop('_merge', 1))

# Counterfacttual dataset is counterfact and factual dataset is df
counterfact = pd.concat([cf2007, cf2008])

# define distance
def dist(d):
    '''
    This function calculates distance between two points in miles
    b_loc = The coordinates for buyer's location
    t_loc = The coordinates for target's location
    '''
    b_loc = (d['buyer_lat'], d['buyer_long'])
```

```
    t_loc = (d['target_lat'], d['target_long'])
    return distance.distance(b_loc, t_loc).miles

df['distance'] = df.apply(dist, axis=1)
counterfact['distance'] = counterfact.apply(dist, axis=1)

#Score function definition for model 1 without transfers
def score1_GS(params, m, n):
    '''
    Payoff function for mergers to be used in the indicator function
    Indicator == 1 if f(b,t) + f(b',t') > f(b',t) + f(b,t')
    Indicator == 0 otherwise
    '''
    f_b_t = m['num_stations_buyer'] * m['population_target'] + params[0] * m['
        corp_owner_buyer'] * m['population_target'] + params[1] * m['distance']
    f_cb_ct = n['num_stations_buyer'] * n['population_target'] + params[0] * n['
        corp_owner_buyer'] * n['population_target'] + params[1] * n['distance']
    f_cb_t = n['num_stations_buyer'] * m['population_target'] + params[0] * n['
        corp_owner_buyer'] * m['population_target'] + params[1] * m['distance']
    f_b_ct = m['num_stations_buyer'] * n['population_target'] + params[0] * m['
        corp_owner_buyer'] * n['population_target'] + params[1] * m['distance']
    L = f_b_t + f_cb_ct
    R = f_cb_t + f_b_ct
    indicator=(L>R)
    total_payoff = indicator.sum()
    return -total_payoff

bounds = [(-10,10), (-10,10)]
GS1_dif = differential_evolution(score1_GS, bounds, args=(df, counterfact), strategy
    ='best1bin', maxiter=10000)

def score2_GS(params, m, n):

    '''
    Payoff function for mergers to be used in the indicator function
    Indicator == 1 if f(b,t) + f(b',t') > f(b',t) + f(b,t')
    Indicator == 0 otherwise
    '''
    f_b_t = params[0] * m['num_stations_buyer'] * m['population_target'] + params[1]
         * m['corp_owner_buyer'] * m['population_target'] + params[2] * m['
        hhi_target'] + params[3] * m['distance']
    f_cb_ct = params[0] * n['num_stations_buyer'] * n['population_target'] + params
        [1] * n['corp_owner_buyer'] * n['population_target'] + params[2]*n['
        hhi_target'] + params[3] * n['distance']
    f_cb_t = params[0] * n['num_stations_buyer'] * m['population_target'] + params
        [1] * n['corp_owner_buyer'] * m['population_target'] + params[2] * m['
        hhi_target'] + params[3] * m['distance']
    f_b_ct = params[0] * m['num_stations_buyer'] * n['population_target'] + params
        [1] * n['corp_owner_buyer'] * n['population_target'] + params[2] * m['
        hhi_target'] + params[3] * m['distance']

    L = f_b_t + f_cb_ct
    R = f_cb_t + f_b_ct
    indicator=(L>R)
    total_payoff = indicator.sum()
    return -total_payoff

bounds = [(-5,5), (-10,10), (-10, 10), (-10,10)]
GS2_dif = differential_evolution(score2_GS, bounds, args=(df, counterfact), strategy
    ='best1bin', maxiter=10000)

def score1_BSS(params, m, n):
    '''
    Payoff function for mergers to be used in the indicator function
    Indicator == 1 if f(b,t) + f(b',t') > f(b',t) + f(b,t')
    Indicator == 0 otherwise
    '''
    f_b_t = m['num_stations_buyer'] * m['population_target'] + params[0] * m['
        corp_owner_buyer'] * m['population_target'] + params[1] * m['distance']
    f_cb_ct = n['num_stations_buyer'] * n['population_target'] + params[0] * n['
        corp_owner_buyer'] * n['population_target'] + params[1] * n['distance']
```

```
    f_cb_t = n['num_stations_buyer'] * m['population_target'] + params[0] * n['
        corp_owner_buyer'] * m['population_target'] + params[1] * m['distance']
    f_b_ct = m['num_stations_buyer'] * n['population_target'] + params[0] * m['
        corp_owner_buyer'] * n['population_target'] + params[1] * m['distance']
    L1 = f_b_t - f_b_ct
    R1 = m['price'] - n['price']
    L2 = f_cb_ct - f_cb_t
    R2 = n['price'] - m['price']
    indicator= ((L1 >= R1) & (L2 >= R2))
    total_payoff = indicator.sum()
    return -total_payoff

bounds = [(-5,5), (-10,10)]
BSS1_dif = differential_evolution(score1_BSS, bounds, args=(df, counterfact),
    strategy='best1bin', maxiter=10000)

def score2_BSS(params, m, n):
    '''
    Payoff function for mergers to be used in the indicator function.
    Indicator == 1 if f(b,t) + f(b',t') > f(b',t) + f(b,t')
    Indicator == 0 otherwise.
    '''
    f_b_t = params[0] * m['num_stations_buyer'] * m['population_target'] + params[1]
         * m['corp_owner_buyer'] * m['population_target'] + params[2] * m['
        hhi_target'] + params[3] * m['distance']
    f_cb_ct = params[0] * n['num_stations_buyer'] * n['population_target'] + params
        [1] * n['corp_owner_buyer'] * n['population_target'] + params[2]*n['
        hhi_target'] + params[3] * n['distance']
    f_cb_t = params[0] * n['num_stations_buyer'] * m['population_target'] + params
        [1] * n['corp_owner_buyer'] * m['population_target'] + params[2] * m['
        hhi_target'] + params[3] * m['distance']
    f_b_ct = params[0] * m['num_stations_buyer'] * n['population_target'] + params
        [1] * n['corp_owner_buyer'] * n['population_target'] + params[2] * m['
        hhi_target'] + params[3] * m['distance']

    L1 = f_b_t - f_b_ct
    R1 = m['price'] - n['price']
    L2 = f_cb_ct - f_cb_t
    R2 = n['price'] - m['price']
    indicator= ((L1 >= R1) & (L2 >= R2))
    total_payoff = indicator.sum()
    return -total_payoff

bounds = [(-5,5), (-10,10), (-10, 10), (-10,10)]
BSS2_dif = differential_evolution(score2_BSS, bounds, args=(df, counterfact),
    strategy='best1bin', maxiter=10000)

print('Model 1, without transfers: \n', GS1_dif.x)
print('Model 2, without transfers: \n', GS2_dif.x)
print('EModel 1, with transfers: \n', BSS1_dif.x)
print('Model 2, with transfers: \n', BSS2_dif.x)
```