Problem Set #5

Mahyar Ebrahimitorki
Mahyar.Ebrahimitorki@grad.moore.sc.edu

ECON833 — October 19, 2021

1 MSE

1.1 Interpretation

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

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

which does not consider transfers and is calculated using Gale-Shapely (GS) Algorithm and

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

which considers transfers using Becker-Shapely-Subik (BSS) algorithm. We also define two different payoff functions for our desired relationship between variables of interest. Also two different payoff functions provided in the prompt are defined as follows:

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

in which x_1bm is the number of stations owned by parent company of the buyer and y_1tm is the population in range of the target in market m, x_2bm is an indicator for corporate ownership, and $distance_btm$ is the distance (in miles) between the buyer and target. The second form of payoff function is

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

where HHi_tm is the Hindahl-Hirschman Index measuring market concentration.

Estimation results for these four models and algorithms are provided in the results section. Table 1 shows the results for estimated parameters using Gale-Shapely (GS) Algorithm and first value function. Based on the results we can see that the interaction between corporate ownership and the population has positive effect on the companies' value but it has negative impact on companies' value when we consider the price. In both methods, GS and BSS, the coefficients of distance (Betas) are positive and relatively large. The further away is the target, the more is the payoff. Thus, Companies are more interested in expanding their ownership in farther locations. That is, one mile farther increases the value of targets for company 6.9 units of payoff in GS and 9.4 Dollars in BSS algorithm.

In the Table2, the interaction between the number of stations owned by the parent company and population (Sigma) has positive effect on companies' value and it's magnitude increased when we have transfers in the algorithm. The effect of interaction between ownership and population is positive and stronger without transfer and is not consistent with BSS algorithm results. This inconsistency in two algorithms is slimier to the first value function estimation's results. Furthermore, market concentration's coefficient (Gamma), based on GS algorithm, is positive. But based on BSS algorithm, it is negative which means the higher concentrated markets attract the mergers less. Distance has positive effects on companies' value

based on both methods which is consistent with first value function results. In general we can say, the distance has significant effect on merging value of company due to large magnitude and consistency in both algorithm.

1.2 Results

Table 1: Estimated Parameters for value function 1, without and with transfers, using GS and BSS algorithm

Algorithm	Alpha	Beta
Gale-Shapely (GS)	7.26329	6.9054
Becker-Shapely-Subik (BSS)	-2.1352	9.47624

Table 2: Estimated Parameters for value function 2, without and with transfers, using GS and BSS algorithm

Algorithm	Sigma	Alpha	Gamma	Beta
Gale-Shapely (GS)	0.729169	7.3894	1.08678	4.95609
Becker-Shapely-Subik (BSS)	2.41002	-4.62312	-3.21928	9.72082

1.3 Executable Script

```
import numpy as np
import pandas as pd
from pandas import DataFrame as df
from pandas import ExcelWriter
from pandas import ExcelFile
import itertools
import scipy.optimize as opt
from scipy.optimize import minimize
from scipy.optimize import differential_evolution
import geopy
from geopy import distance
# 1.reading
df = pd.read_csv("radio_merger_data.csv")
#2.Convert scale of price and population to 1000 dollars and 1000 people
    respectively
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]
cf2007 = counterfact[counterfact['year']==2007]
cf2008 = counterfact[counterfact['year']==2008]
dfy = dict(iter(df.groupby('year', as_index = False)))
dfy[2007].describe().to_csv("describe2007.csv")
dfy[2008].describe().to_csv("describe2008.csv")
B = ['year', 'buyer_id', 'buyer_lat', 'buyer_long', 'num_stations_buyer', '
    corp_owner_buyer']
T = ['target_id', 'target_lat', 'target_long', 'price', 'hhi_target', '
    population_target']
data = [pd.DataFrame(dfy[2007]), pd.DataFrame(dfy[2008])]
counterfact = pd.DataFrame()
counterfact = pd.DataFrame([x[B].iloc[i].values.tolist() + x[T].iloc[j].values.
    tolist() for x in data for i in range(len(x)) for j in range(len(x)) if i!= j],
    columns = B + T)
counterfact.head()
# 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, using Gale-Shapely
    Algorithm
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)
#Score function definition for model 2 , without transfers, using Gale-Shapely
    Algorithm
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=100000)
#Score function definition for model 1 , with transfers, using Becker-Shapely-Subik
    Algorithm
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)
#Score function definition for model 2 , with transfers, using Becker-Shapely-Subik
    Algorithm
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)
from tabulate import tabulate
d = [GS1_dif.x ,BSS1_dif.x]
print(tabulate(d,headers=["Alpha ","Beta"],tablefmt="github"))
dd=  [GS2_dif.x,BSS2_dif.x]

print(tabulate(dd,headers=["Sigma "," Alpha ","Gamma ","Beta" ],tablefmt="github"))
from tabulate import tabulate
d = [GS1_dif.x ,BSS1_dif.x]
print(tabulate(d,headers=["Alpha ","Beta"],tablefmt="latex"))
dd=  [GS2_dif.x,BSS2_dif.x]

print(tabulate(dd,headers=["Sigma "," Alpha ","Gamma ","Beta" ],tablefmt="latex"))
```