GR5243 Project 4 Doubly Robust Estimations(scaled)

Group3 - Zi Fang

In [1]:

```
import pandas as pd
import numpy as np
import time
from matplotlib import style
from matplotlib import pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
%matplotlib inline

# set seed
random_state = 2021
```

In [2]:

```
lowdim_data = pd.read_csv('../data/lowDim_dataset.csv')
highdim_data = pd.read_csv('../data/highDim_dataset.csv')
```

In [3]:

```
# function to scale the datasets
def scaled_data(data):
    x = data.drop(['A','Y'], axis = 1)
    y = data[["A"]]

    data_columns = data.columns.drop(['Y','A'])

    x_scaled = StandardScaler().fit_transform(x)

    data_scaled = pd.DataFrame(x_scaled, index = data.index, columns = data_columns)

    data_scaled['A'] = data['A']
    data_scaled['Y'] = data['Y']

    display(data_scaled.head())

    return data_scaled
```

In [4]:

```
# scale the high dimentional dataset
highdim_scale_data = scaled_data(highdim_data)
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	
0	-1.015114	0.482748	-1.161393	0.303352	1.487812	-1.171070	-1.423520	1.686961	1.203321	0.38
1	-1.015114	-2.071474	-1.650640	-1.477143	-0.512424	-1.171070	0.204290	0.524392	1.203321	0.80

	V1	V2	V3	V4	V5	V6	V7	V8	V9	
2	-1.015114	-2.071474	0.795598	-1.922267	-0.876103	-0.415004	-0.880917	0.669713	1.203321	-0.03
3	0.985111	-2.071474	-1.324475	-1.477143	-1.239782	-1.171070	-0.700049	0.698777	-0.831034	-0.45
4	0.985111	0.482748	-0.019815	0.971038	0.214934	0.492274	2.012968	0.756906	1.203321	0.38

5 rows × 187 columns

In [5]:

```python
# scale the low dimentional dataset
lowdim_scale_data = scaled_data(lowdim_data)
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V
0	-0.502205	-0.352816	-0.257883	-0.266592	-0.34195	-0.465776	-0.266412	-0.649809	-0.35092	-0.159
1	-0.502205	-0.352816	-0.257883	-0.266592	-0.34195	-0.465776	-0.266412	1.034631	-0.35092	-0.159
2	-0.502205	-0.352816	-0.257883	-0.266592	-0.34195	-0.465776	-0.266412	-0.649809	-0.35092	-0.159
3	3.441468	-0.352816	-0.257883	-0.266592	-0.34195	-0.465776	-0.266412	-0.649809	-0.35092	-0.159
4	-0.253654	0.209949	-0.150877	-0.081459	-0.34195	0.445723	0.162041	0.493204	1.15826	-0.071

5 rows × 24 columns

In [6]:

```python
def best_param(data, random_state, param_grid, cv=10):
    '''
    Purpose: to find the best parameter "C" (coefficient of regularization strength) fo

    Parameters:
    data - dataset to best tested on
    random_state - set seed
    param_grid - set of parameter values to test on
    cv - number of folds for cross-validation

    '''


    x = data.drop(['A','Y'], axis = 1)
    y = data[['A']].values.ravel()

    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25, random_st


    model_cv = GridSearchCV(LogisticRegression(penalty='l1',solver = 'liblinear'), para
    model_cv.fit(x_train, y_train)

    print("The best tuned coefficient of regularization strength is",model_cv.best_para
          "with a testing accuracy of", model_cv.score(x_test, y_test))

    return model_cv.best_params_.get('C')
```

In [7]:

```python
def propensity_score(data, C=0.1, plot = True):
    '''
    Purpose: to estimate propensity score with L1 penalized logistic regression

    Parameters:
    data - dataset to estimate on
    C - coeficient of regularization strength
    plot - print out visualization to show distribution of propensity scores

    Returns:
    1. ps for Propensity Score
    2. Visualization plot to show distribution of propensity scores

    '''


    T = 'A'
    Y = 'Y'
    X = data.columns.drop([T,Y])

    ps_model = LogisticRegression(random_state=random_state, penalty='l1',
                                  solver='liblinear').fit(data[X], data[T])

    ps = ps_model.predict_proba(data[X])[:,1] # we are interested in the probability of

    if plot:
        df_plot = pd.DataFrame({'Treatment':data[T], 'Propensity Score':ps})

        sns.histplot(data=df_plot, x = "Propensity Score", hue = "Treatment", element =
        plt.title("Distribution of Propensity Score by Treatment Group", size=20)
        plt.show()

    return ps
```

In [8]:

```
# setting parameters
param_grid = {"C":[0.01,0.05,0.1,0.3,0.5,0.7,1]}
```

Low Dimensional Case

In [9]:

```
# use 10-fold cross-validation to tune for the best parameter for logistic regression
DR_low_start = time.time()
c_low = best_param(lowdim_scale_data, random_state=random_state, param_grid=param_grid)
```

```
The best tuned coefficient of regularization strength is 0.3 with a testing accuracy of
0.792
```

In [10]:

```
# calculate propensity score for low dimensional case
PS_low = propensity_score(lowdim_scale_data, C = c_low, plot = False)
```

In [11]:

```
# reload data, add propensity score column and divide data into treat and control group
lowdim_data = pd.read_csv('../data/lowDim_dataset.csv')
lowdim_data['PS_low'] = pd.Series(PS_low, index=lowdim_data.index)
lowdim_treat = lowdim_data[lowdim_data.A == 1].reset_index(drop = True)
lowdim_control = lowdim_data[lowdim_data.A == 0].reset_index(drop = True)
```

In [12]:

```
# fit regression models to treat and control group
```

```
xlow_treat = lowdim_treat.drop(['A','Y','PS_low'],axis=1)
ylow_treat = lowdim_treat['Y']
lr_low_treat = LinearRegression().fit(xlow_treat, ylow_treat)

xlow_control = lowdim_control.drop(['A','Y','PS_low'],axis=1)
ylow_control = lowdim_control['Y']
lr_low_control = LinearRegression().fit(xlow_control, ylow_control)
```

In [13]:

```
# make prediction based on trained models and construct a full dataset
xlow = lowdim_data.drop(['A','Y','PS_low'],axis=1)
lowdim_data['mtreat'] = lr_low_treat.predict(xlow)
lowdim_data['mcontrol'] = lr_low_control.predict(xlow)
```

In [14]:

```
# perform Doubly Robust Estimation algorithm
DR_low_1 = 0
DR_low_0 = 0
for i in range(len(lowdim_data)):
    DR_low_1 = (DR_low_1 + lowdim_data['A'][i]*lowdim_data['Y'][i]
                -(lowdim_data['A'][i]-lowdim_data['PS_low'][i])*lowdim_data['mtreat'][i
    DR_low_0 = (DR_low_0 + (1-lowdim_data['A'][i])*lowdim_data['Y'][i]
                +(lowdim_data['A'][i]-lowdim_data['PS_low'][i])*lowdim_data['mcontrol']
DR_low_ETA = (DR_low_1 - DR_low_0)/len(lowdim_data)
DR_low_accu = 1 - abs((DR_low_ETA -2.0901)/2.0901)
DR_low_end = time.time()
DR_low_time = DR_low_end - DR_low_start
```

In [15]:

```
# print the ETA, accuracy and algorithm running time results
print(f'Doubly robust estimation method for low dimensional dataset:\n ETA = {DR_low_ET
```

```
Doubly robust estimation method for low dimensional dataset:
 ETA = -6.60
 Accuracy = -3.16
 DR running time = 0.63
```

High Dimensional Case

In [16]:

```
# use 10-fold cross-validation to tune for the best parameter for logistic regression
DR_high_start = time.time()
c_high = best_param(highdim_scale_data, random_state=random_state, param_grid=param_gri
```

```
The best tuned coefficient of regularization strength is 0.05 with a testing accuracy of
0.716
```

In [17]:

```
# calculate propensity score for high dimensional case
PS_high = propensity_score(highdim_scale_data, C = c_high, plot = False)
```

In [18]:

```
# reload data, add propensity score column and divide data into treat and control group
highdim_data = pd.read_csv('../data/highDim_dataset.csv')
highdim_data['PS_high'] = pd.Series(PS_high, index=highdim_data.index)
highdim_treat = highdim_data[highdim_data.A == 1].reset_index(drop = True)
highdim_control = highdim_data[highdim_data.A == 0].reset_index(drop = True)
```

In [19]:
```
# fit regression model to treat and control group
xhigh_treat = highdim_treat.drop(['A','Y','PS_high'],axis=1)
yhigh_treat = highdim_treat['Y']
lr_high_treat = LinearRegression().fit(xhigh_treat, yhigh_treat)

xhigh_control = highdim_control.drop(['A','Y','PS_high'],axis=1)
yhigh_control = highdim_control['Y']
lr_high_control = LinearRegression().fit(xhigh_control, yhigh_control)
```

In [20]:
```
# make prediction based on trained models and construct a full dataset
xhigh = highdim_data.drop(['A','Y','PS_high'],axis=1)
highdim_data['mtreat'] = lr_high_treat.predict(xhigh)
highdim_data['mcontrol'] = lr_high_control.predict(xhigh)
```

In [21]:
```
# perform Doubly Robust Estimation algorithm
DR_high_1 = 0
DR_high_0 = 0
for i in range(len(highdim_data)):
    DR_high_1 = (DR_high_1 + highdim_data['A'][i]*highdim_data['Y'][i]
                -(highdim_data['A'][i]-highdim_data['PS_high'][i])*highdim_data['mtreat
    DR_high_0 = (DR_high_0 + (1-highdim_data['A'][i])*highdim_data['Y'][i]
                +(highdim_data['A'][i]-highdim_data['PS_high'][i])*highdim_data['mcontr
DR_high_ETA = (DR_high_1 - DR_high_0)/len(highdim_data)
DR_high_accu = 1 - abs((DR_high_ETA -(-54.8558))/(-54.8558))
DR_high_end = time.time()
DR_high_time = DR_high_end - DR_high_start
```

In [22]:
```
# print the ETA, accuracy and algorithm running time result
print(f'Doubly robust estimation method for high dimensional dataset:\n ETA = {DR_high_
```

```
Doubly robust estimation method for high dimensional dataset:
 ETA = -34.67
 Accuracy = 0.63
 DR running time = 14.56
```