GR5243 Project 4 Doubly Robust Estimation

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

%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]:

```
# create a function to tune for best hyperparameters for each data set

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 [4]:

```
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 [5]:

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

Low Dimensional Case

In [6]:

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

```
The best tuned coefficient of regularization strength is 0.3 with a testing accuracy of
0.8
```

In [7]:

```
# calculate propensity score for low dimensional case
PS_low = propensity_score(lowdim_data, C=0.1, plot = False)
```

In [8]:

```
# 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 [9]:

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

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

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

In [10]:

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

In [11]:

```
# 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 = (2.0901 - DR_low_ETA)/2.0901
DR_low_end = time.time()
DR_low_time = DR_low_end - DR_low_start
```

In [12]:

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

```
Doubly robust estimation method for low dimensional dataset:
 ETA = -6.68
 Accuracy = 4.19
 DR running time = 0.63
```

High Dimensional Case

In [13]:

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

```
The best tuned coefficient of regularization strength is 0.01 with a testing accuracy of
0.71
```

In [14]:

```
# calculate propensity score for high dimensional case
PS_high = propensity_score(highdim_data, C=0.1, plot = False)
```

In [15]:

```
# 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 [16]:

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

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

In [17]:

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

In [18]:

```
# 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 = (-54.8558 - DR_high_ETA)/-54.8558
DR_high_end = time.time()
DR_high_time = DR_high_end - DR_high_start
```

In [19]:

```
# 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 = -34.42
 Accuracy = 0.37
 DR running time = 70.89
```