ADS_algorithm_5

November 30, 2022

0.1 Prejudice

Prejudice means a statistical dependence between a sensitive variable, S, and the target variable, Y, or a non-sensitive variable, X.

There are three types of prejudices: ## Direct prejudice

Direct prejudice is the use of a sensitive variable in a prediction model.

To eliminate direct prejudice, we can remove the sensitive variable from the model.

0.2 Indirect prejudice

Indirect prejudice is statistical dependence between a sensitive variable and a target variable.

To remove this indirect prejudice, we must use a prediction model that satisfies the condition $Y \perp\!\!\!\perp S$.

We can quantify the degree of indirect prejudice using the following equation where PI refers to the (indirect) prejudice index and $\mathcal{D}$ is the data set.

$$\mathrm{PI} = \sum_{(y,s)\in\mathcal{D}} \hat{\Pr}[y,s] \ln \frac{\hat{\Pr}[y,s]}{\hat{\Pr}[y]\hat{\Pr}[s]}$$

The application of the normalization technique for mutual information leads to a *normalized prejudice index* (NPI)

$$\mathrm{NPI} = \frac{\mathrm{PI}}{\sqrt{\mathrm{H}(Y)\mathrm{H}(S)}}$$

where $\mathrm{H}(\,\cdot\,)$ is the entropy function.

0.3 Latent prejudice

Latent prejudice is a statistical dependence between a sensitive variable, S, and a non-sensitive variable, X.

Removal of potential prejudice is achieved by making X and Y independent from S simultaneously.

0.4 Underestimation

Underestimation is the state in which a learned model is not fully converged due to the finiteness of the size of a training data set.

Despite that a prediction model without indirect prejudice can learn to make a fair determination, this is only the case if we have an "infinitely large" training data set. In general, training sets are finite and limited to small quantities of data, hence the model could output even more unfair determinations than that observed in the training sample distribution.

To quantify the degree of underestimation, we assess the resultant difference between the training sample distribution over $\mathcal{D}$, $\tilde{\Pr}$ using the underestimation index (UEI) which is calculated using the Hellinger distance:

$$\mathrm{UEI} = \sqrt{\frac{1}{2} \sum_{(y,s)\in\mathcal{D}} \left(\sqrt{\tilde{\Pr}[y,s]} - \sqrt{\hat{\Pr}[y,s]} \right)^2} = \sqrt{1 - \sum_{(y,s)\in\mathcal{D}} \sqrt{\hat{\Pr}[Y,S]\tilde{\Pr}[Y,S]}}$$

where $\hat{\Pr}$ is the distribution of the learned model.

0.5 Negative Legacy

Negative legacy is unfair sampling or labeling in the training data.

For example, if a bank has been refusing credit to minority people without assessing them, the records of minority people are less sampled in a training data set.

0.6 General Framework

Given a training data set $\$\mathrm{D} = \$ \ \{(y, \mathbf{x}, s)\}$, we can define the following terms:

- $\mathcal{M}\ [Y|X, S; \nleq]$ conditional probability of a class given non-sensitive and sensitive features model.
- $\nleq$ set of model parameters. These parameters are estimates based on the maximum likelihood principle:
$$\mathcal{L}(\mathcal{D}, \nleq) = \sum_{(\dagger_\rangle, \mathbf{x}_\rangle, \int_\rangle)\in\mathcal{D}} \ln \mathcal{M}\ [\dagger_\rangle | \mathbf{x}_\rangle, \int_\rangle; \nleq].$$

For the optimization process, we use two types of regularizers, the L_2 regularizer $||\nleq||_2^2$ and a second regularizer $R(\mathcal{D}, \nleq)$, introduced to enforce fair classification. After applying both regularizing techniques, are objective function becomes:

$$-\mathcal{L}(\mathcal{D}, \nleq) + \eta \mathrm{R}(\mathcal{D}, \nleq) + \frac{\lambda}{\in} ||\nleq||_\in^\in.$$

0.7 Prejudice Remover

A *prejudice remover* regularizer directly tries to reduce the prejudice index and is denoted by R_{PR}. Recall that the prejudice index is defined as

$$\mathrm{PI} = \sum_{Y,S} \hat{\Pr}[Y,S] \ln \frac{\hat{\Pr}[Y,S]}{\hat{\Pr}[Y]\hat{\Pr}[S]}$$

where

$$\hat{\Pr}[y|s_i] \approx \frac{\sum_{(\mathbf{x}_i,s_i)\in\mathcal{D} \text{ s.t. } s_i=s} \mathcal{M}[y|\mathbf{x}_i,s;\not\lesssim]}{|\{(\mathbf{x}_i,s_i)\in\mathcal{D} \text{ s.t. } s_i=s\}|}.$$

$$\hat{\Pr}[y] \approx \frac{\sum_{(\mathbf{x}_i,s_i)\in\mathcal{D}} \mathcal{M}[y|\mathbf{x}_i,s_i;\not\lesssim]}{|\mathcal{D}|}.$$

And the prejudice remover regularizer $\mathrm{R}_{\mathrm{PR}}(\mathcal{D},\not\lesssim)$ is defined as

$$\sum_{(\mathbf{x}_i,s_i)\in\mathcal{D}} \sum_{y\in\{0,1\}} \mathcal{M}[y|\mathbf{x}_i,s_i;\not\lesssim] \ln \frac{\hat{\Pr}[y|s_i]}{\hat{\Pr}[y]}$$

This regularizer becomes increasingly large as a class y becomes more likely to be predicted for a sensitive group s than for the entire population, thus making the overall model is influenced less by the sensitive variables.

[30]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import random

import torch as t
import torch.nn as nn
```

[2]:
```
data = pd.read_csv('./compas-scores-two-years.csv')
```

[3]:
```
data.head()
```

[3]:
```
   id                name   first         last compas_screening_date   sex  \
0   1    miguel hernandez  miguel    hernandez            2013-08-14  Male
1   3         kevon dixon   kevon        dixon            2013-01-27  Male
2   4            ed philo      ed        philo            2013-04-14  Male
3   5         marcu brown   marcu        brown            2013-01-13  Male
4   6  bouthy pierrelouis  bouthy  pierrelouis            2013-03-26  Male

          dob  age          age_cat              race  …  v_decile_score  \
0  1947-04-18   69  Greater than 45             Other  …               1
1  1982-01-22   34          25 - 45  African-American  …               1
2  1991-05-14   24     Less than 25  African-American  …               3
3  1993-01-21   23     Less than 25  African-American  …               6
4  1973-01-22   43          25 - 45             Other  …               1

   v_score_text  v_screening_date  in_custody  out_custody  priors_count.1  \
0           Low        2013-08-14  2014-07-07   2014-07-14               0
1           Low        2013-01-27  2013-01-26   2013-02-05               0
```

```
2          Low        2013-04-14  2013-06-16   2013-06-16                 4
3       Medium        2013-01-13         NaN          NaN                 1
4          Low        2013-03-26         NaN          NaN                 2

  start   end event two_year_recid
0     0   327     0              0
1     9   159     1              1
2     0    63     0              1
3     0  1174     0              0
4     0  1102     0              0

[5 rows x 53 columns]
```

[4]:
```
data = data[['age', 'c_charge_degree', 'race', 'age_cat', 'score_text', 'sex',␣
↪'priors_count',
             'days_b_screening_arrest', 'decile_score', 'is_recid',␣
↪'two_year_recid', 'c_jail_in', 'c_jail_out']]
```

[5]:
```
data = data.loc[(data['days_b_screening_arrest'] <= 30) &␣
↪(data['days_b_screening_arrest'] >= -30)]
```

[6]:
```
data = data.loc[data['is_recid'] != -1]
```

[7]:
```
data = data.loc[data['c_charge_degree'] != 'O']
```

[8]:
```
data = data.loc[data['score_text'] != 'N/A']
```

[9]:
```
data["length_of_stay"] = (pd.to_datetime(data['c_jail_out'])-pd.
↪to_datetime(data['c_jail_in'])).dt.days
```

[10]:
```
data = data.drop(columns=['c_jail_in', 'c_jail_out'])
```

[11]:
```
data.head()
```

[11]:
```
   age c_charge_degree              race          age_cat score_text   sex  \
0   69               F             Other  Greater than 45        Low  Male
1   34               F  African-American          25 - 45        Low  Male
2   24               F  African-American     Less than 25        Low  Male
5   44               M             Other          25 - 45        Low  Male
6   41               F         Caucasian          25 - 45     Medium  Male

   priors_count  days_b_screening_arrest  decile_score  is_recid  \
0             0                     -1.0             1         0
1             0                     -1.0             3         1
2             4                     -1.0             4         1
5             0                      0.0             1         0
6            14                     -1.0             6         1
```

```
   two_year_recid  length_of_stay
0               0               0
1               1              10
2               1               1
5               0               1
6               1               6
```

[12]:
```python
data.shape
```

[12]:
```
(6172, 12)
```

[13]:
```python
data = data.loc[(data["race"] == "African-American") | (data["race"] ==␣
 ↪"Caucasian")]
```

[14]:
```python
data = data.replace({'race': 'Caucasian'}, 1)
data = data.replace({'race': 'African-American'}, 0)

data = data.replace({'sex': 'Male'}, 1)
data = data.replace({'sex': 'Female'}, 0)

data = data.replace({'age_cat': 'Less than 25'}, 0)
data = data.replace({'age_cat': '25 - 45'}, 1)
data = data.replace({'age_cat': 'Greater than 45'}, 2)

data = data.replace({'c_charge_degree': 'F'}, 0)
data = data.replace({'c_charge_degree': 'M'}, 1)

data = data.replace({'score_text': 'Low'}, 0)
data = data.replace({'score_text': 'Medium'}, 1)
data = data.replace({'score_text': 'High'}, 2)
```

[15]:
```python
data.head()
```

[15]:
```
    age  c_charge_degree  race  age_cat  score_text  sex  priors_count  \
1    34                0     0        1           0    1             0
2    24                0     0        0           0    1             4
6    41                0     1        1           1    1            14
8    39                1     1        1           0    0             0
10   27                0     1        1           0    1             0

    days_b_screening_arrest  decile_score  is_recid  two_year_recid  \
1                      -1.0             3         1               1
2                      -1.0             4         1               1
6                      -1.0             6         1               1
8                      -1.0             1         0               0
10                     -1.0             4         0               0
```

```
    length_of_stay
1               10
2                1
6                6
8                2
10               1
```

[16]:

```
data.drop_duplicates()
```

[16]:

```
      age  c_charge_degree  race  age_cat  score_text  sex  priors_count  \
1      34                0     0        1           0    1             0
2      24                0     0        0           0    1             4
6      41                0     1        1           1    1            14
8      39                1     1        1           0    0             0
10     27                0     1        1           0    1             0
…   …               …   …        …          …  …           …
7206   21                1     1        0           1    1             0
7207   30                1     0        1           0    1             0
7208   20                0     0        0           2    1             0
7209   23                0     0        0           1    1             0
7212   33                1     0        1           0    0             3

      days_b_screening_arrest  decile_score  is_recid  two_year_recid  \
1                        -1.0             3         1               1
2                        -1.0             4         1               1
6                        -1.0             6         1               1
8                        -1.0             1         0               0
10                       -1.0             4         0               0
…                        …            …       …              …
7206                     -1.0             6         1               1
7207                     -1.0             2         1               1
7208                     -1.0             9         0               0
7209                     -1.0             7         0               0
7212                     -1.0             2         0               0

      length_of_stay
1                 10
2                  1
6                  6
8                  2
10                 1
…              …
7206               3
7207               0
7208               0
7209               1
```

```
7212                1

[5094 rows x 12 columns]
```

[17]:
```
X = data.drop(columns=["two_year_recid"])
y = data["two_year_recid"]
```

[18]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

[19]:
```
X_train_a = X_train[(X_train['race'] == 0)]
X_train_c = X_train[(X_train['race'] == 1)]
y_train_a = y_train[(X_train['race'] == 0)]
y_train_c = y_train[(X_train['race'] == 1)]
```

[20]:
```
X_test_a = X_test[(X_test['race'] == 0)]
X_test_c = X_test[(X_test['race'] == 1)]
y_test_a = y_test[(X_test['race'] == 0)]
y_test_c = y_test[(X_test['race'] == 1)]
```

[21]:
```
X_train_a = t.tensor(np.array(X_train_a)).to(t.float32)
y_train_a = t.from_numpy(np.array(y_train_a).astype('float32')).
↪reshape(X_train_a.shape[0], 1)
X_train_c = t.tensor(np.array(X_train_c)).to(t.float32)
y_train_c = t.from_numpy(np.array(y_train_c).astype('float32')).
↪reshape(X_train_c.shape[0], 1)

X_test_a = t.tensor(np.array(X_test_a)).to(t.float32)
y_test_a = t.from_numpy(np.array(y_test_a).astype('float32')).reshape(X_test_a.
↪shape[0], 1)
X_test_c = t.tensor(np.array(X_test_c)).to(t.float32)
y_test_c = t.from_numpy(np.array(y_test_c).astype('float32')).reshape(X_test_c.
↪shape[0], 1)
```

[22]:
```
class LogisticRegression(nn.Module):
    def __init__(self,data):
        super(LogisticRegression, self).__init__()
        self.w = nn.Linear(data.shape[1], out_features=1, bias=True)
        self.sigmod = nn.Sigmoid()
    def forward(self, x):
        w = self.w(x)
        output = self.sigmod(w)
        return output
```

[24]:
```
def metrics_cal(Model_a,Model_c, X_a, y_a, X_c, y_c):
    y_pred_a = (Model_a(X_a) >= 0.5)
    y_pred_c = (Model_c(X_c) >= 0.5)
    accuracy_a  = t.sum(y_pred_a.flatten() == y_a.flatten()) / y_a.shape[0]
```

```python
    accuracy_c  = t.sum(y_pred_c.flatten() == y_c.flatten()) / y_c.shape[0]
    accuracy = (accuracy_a + accuracy_c) / 2
    calibration = t.abs(accuracy_a - accuracy_c)
    return round(accuracy.item(),4), round(calibration.item(),4)
```

[25]:
```python
class PRLoss():
    def __init__(self, eta=1.0):
        super(PRLoss, self).__init__()
        self.eta = eta

    def forward(self,output_a,output_c):
        N_a = t.tensor(output_a.shape[0])
        N_c   = t.tensor(output_c.shape[0])
        Dxisi = t.stack((N_a,N_c),axis=0)
        # Pr[y|s]
        y_pred_a = t.sum(output_a)
        y_pred_c   = t.sum(output_c)
        P_ys = t.stack((y_pred_a,y_pred_c),axis=0) / Dxisi
        # Pr[y]
        P = t.cat((output_a,output_c),0)
        P_y = t.sum(P) / (X_train_a.shape[0]+X_train_c.shape[0])
        # P(siyi)
        P_s1y1 = t.log(P_ys[1]) - t.log(P_y)
        P_s1y0 = t.log(1-P_ys[1]) - t.log(1-P_y)
        P_s0y1 = t.log(P_ys[0]) - t.log(P_y)
        P_s0y0 = t.log(1-P_ys[0]) - t.log(1-P_y)
        # PI
        PI_s1y1 = output_a * P_s1y1
        PI_s1y0 =(1- output_a) * P_s1y0
        PI_s0y1 = output_c * P_s0y1
        PI_s0y0 = (1- output_c )* P_s0y0
        PI = t.sum(PI_s1y1) + t.sum(PI_s1y0) + t.sum(PI_s0y1) + t.sum(PI_s0y0)
        PI = self.eta * PI
        return PI
```

[26]:
```python
class PRLR():

    def __init__(self, eta=0.0, epochs=100, lr = 0.01):
        super(PRLR, self).__init__()
        self.eta = eta
        self.epochs = epochs
        self.lr = lr

    def fit(self,X_train_a,y_train_a,X_train_c,y_train_c,
            X_test_a, y_test_a, X_test_c, y_test_c):
        model_a = LogisticRegression(X_train_a)
        model_c = LogisticRegression(X_train_c)
```

```
        criterion = nn.BCELoss(reduction='sum')
        PI = PRLoss(eta=self.eta)
        epochs = self.epochs
        optimizer = t.optim.Adam(list(model_a.parameters())+ list(model_c.
 ↪parameters()), self.lr, weight_decay=1e-5)

        for epoch in range(epochs):
            model_a.train()
            model_c.train()
            optimizer.zero_grad()
            output_a = model_a(X_train_a)
            output_c = model_c(X_train_c)
            logloss = criterion(output_a, y_train_a)+ criterion(output_c,␣
 ↪y_train_c)
            PIloss = PI.forward(output_a,output_c)
            loss = PIloss +logloss
            loss.backward()
            optimizer.step()

        model_a.eval()
        model_c.eval()
        accuracy, calibration = metrics_cal(model_a,model_c,X_test_a, y_test_a,␣
 ↪X_test_c, y_test_c)
        return accuracy, calibration
```

[27]:
```
PR = PRLR(eta = 1.0, epochs = 50, lr = 0.01)
```

[29]:
```
PR.fit(X_train_a,y_train_a,X_train_c,y_train_c,
       X_test_a, y_test_a, X_test_c, y_test_c)
```

[29]: (0.5705, 0.13)