Logistic Regression without Prejudice Regularizer

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
import torch
import torch.nn as nn

import pandas as pd
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
data = pd.read_csv("compas-scores-two-years.csv")
data.columns.values
data.head()
```

Out[2]:

	id	name	first	last	compas_screening_date	sex	dob	age	age_cat	race	...	v_decile_score	v_score
0	1	miguel hernandez	miguel	hernandez	2013-08-14	Male	1947-04-18	69	Greater than 45	Other	...	1	
1	3	kevon dixon	kevon	dixon	2013-01-27	Male	1982-01-22	34	25 - 45	African-American	...	1	
2	4	ed philo	ed	philo	2013-04-14	Male	1991-05-14	24	Less than 25	African-American	...	3	
3	5	marcu brown	marcu	brown	2013-01-13	Male	1993-01-21	23	Less than 25	African-American	...	6	Me
4	6	bouthy pierrelouis	bouthy	pierrelouis	2013-03-26	Male	1973-01-22	43	25 - 45	Other	...	1	

5 rows × 53 columns

Data Preprocessing

In [3]:
```
data = pd.read_csv("compas-scores-two-years.csv")
df = data[['age', 'c_charge_degree', 'race', 'score_text', 'priors_count',
            #'decile_score',
            'two_year_recid', 'c_jail_in', 'c_jail_out', 'is_violent_recid']]


df = df.loc[df['race'].isin(('African-American', 'Caucasian'))]

df.loc[df["race"] == "African-American", "race"] = 0
df.loc[df["race"] == "Caucasian", "race"] = 1


df = df.loc[df['c_charge_degree'] != 'O']
df = df.loc[df['score_text'] != 'N/A']

df['length_of_stay'] = (df['c_jail_out'].apply(pd.to_datetime) - df['c_jail_in'].apply(pd.to_datetime)).c
df = df.dropna(subset = ['length_of_stay'])

df = df.drop(columns=['c_jail_in', 'c_jail_out'])

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

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

df = df.drop_duplicates()

df.head()
```

Out[3]:

	age	c_charge_degree	race	score_text	priors_count	two_year_recid	is_violent_recid	length_of_stay
1	34	0	0	0	0	1	1	10.0
2	24	0	0	0	4	1	0	1.0
6	41	0	1	1	14	1	0	6.0
8	39	1	1	0	0	0	0	2.0
9	21	0	1	0	1	1	1	0.0

In [4]:

```
from sklearn.model_selection import train_test_split

#split dataset so that training:validation:testing=5:1:1
df_train, df_rem = train_test_split(df,train_size=5/7.0)
df_valid, df_test = train_test_split(df_rem, test_size = 0.5)
```

In [5]:

```
df_train_a = df_train[df_train['race'] == 0]
df_train_c = df_train[df_train['race'] == 1]

df_test_a = df_test[df_test['race'] == 0]
df_test_c = df_test[df_test['race'] == 1]

df_valid_a = df_valid[df_valid['race'] == 0]
df_valid_c = df_valid[df_valid['race'] == 1]
```

In [6]:
```
X_train_a = df_train_a.drop(columns = ['two_year_recid', 'race'])
X_train_c = df_train_c.drop(columns = ['two_year_recid', 'race'])
Y_train_a = df_train_a['two_year_recid']
Y_train_c = df_train_c['two_year_recid']
S_train_a = df_train_a['race']
S_train_c = df_train_c['race']

X_test_a = df_test_a.drop(columns = ['two_year_recid', 'race'])
X_test_c = df_test_c.drop(columns = ['two_year_recid', 'race'])
Y_test_a = df_test_a['two_year_recid']
Y_test_c = df_test_c['two_year_recid']
S_test_a = df_test_a['race']
S_test_c = df_test_c['race']

X_valid_a = df_valid_a.drop(columns = ['two_year_recid', 'race'])
X_valid_c = df_valid_c.drop(columns = ['two_year_recid', 'race'])
Y_valid_a = df_valid_a['two_year_recid']
Y_valid_c = df_valid_c['two_year_recid']
S_valid_a = df_valid_a['race']
S_valid_c = df_valid_c['race']
```

In [7]:
```
import torch as t

train_X_c=t.tensor(np.array(X_train_c).astype('float32'))
train_Y_c=t.from_numpy(np.array(Y_train_c).astype('float32')).reshape(X_train_c.shape[0],1)
train_X_a=t.tensor(np.array(X_train_a).astype('float32'))
train_Y_a=t.from_numpy(np.array(Y_train_a).astype('float32')).reshape(X_train_a.shape[0],1)

valid_X_c=t.tensor(np.array(X_valid_c).astype('float32'))
valid_Y_c=t.from_numpy(np.array(Y_valid_c).astype('float32')).reshape(X_valid_c.shape[0],1)
valid_X_a=t.tensor(np.array(X_valid_a).astype('float32'))
valid_Y_a=t.from_numpy(np.array(Y_valid_a).astype('float32')).reshape(X_valid_a.shape[0],1)

test_X_c=t.tensor(np.array(X_test_c).astype('float32'))
test_Y_c=t.from_numpy(np.array(Y_test_c).astype('float32')).reshape(X_test_c.shape[0],1)
test_X_a=t.tensor(np.array(X_test_a).astype('float32'))
test_Y_a=t.from_numpy(np.array(Y_test_a).astype('float32')).reshape(X_test_a.shape[0],1)
```

In [8]:
```
# Accuracy for group of African-American
from sklearn.linear_model import LogisticRegression

clf_a = LogisticRegression(random_state=0).fit(train_X_a, train_Y_a)
accuracy_a = clf_a.score(valid_X_a,valid_Y_a)
accuracy_a
```

Out[8]:
```
0.6729411764705883
```

In [9]:
```
clf_c = LogisticRegression().fit(train_X_c, train_Y_c)
accuracy_c = clf_c.score(valid_X_c,valid_Y_c)
accuracy_c
```

Out[9]:
```
0.7128378378378378
```

In [10]:
```
# Accuracy in general and Calibration
accuracy = (accuracy_a+accuracy_c)/2
calibration = abs(accuracy_a - accuracy_c)

print("Validation accuracy: ", accuracy)
print("Validation calibration score: ", calibration)
```

```
Validation accuracy:  0.692889507154213
Validation calibration score:  0.039896661367249564
```

In [11]:
```
accuracy_a_test = clf_a.score(test_X_a,test_Y_a)
accuracy_c_test = clf_c.score(test_X_c,test_Y_c)
print(accuracy_a_test)
print(accuracy_c_test)
print("Test accuracy: ", (accuracy_a_test + accuracy_c_test)/2)
print("Test calibration score: ", abs(accuracy_a_test - accuracy_c_test))
```

```
0.6605080831408776
0.7152777777777778
Test accuracy:  0.6878929304593278
Test calibration score:  0.05476969463690018
```

In [12]:
```
def accuracy( Model_c,Model_a, df_c_X_train,df_c_y_train,df_a_X_train,df_a_y_train):
    yc_pred = (Model_c(df_c_X_train) >= 0.5)
    ya_pred = (Model_a(df_a_X_train) >= 0.5)
    accu_c  = t.sum(yc_pred.flatten() == df_c_y_train.flatten()) / df_c_X_train.shape[0]
    #accu_c = mean(yc_pred == df_c_y_train)
    accu_a  = t.sum(ya_pred.flatten() == df_a_y_train.flatten()) / df_a_X_train.shape[0]
    #accu_a = mean(ya_pred == df_a_y_train)
    accuracy = (accu_c + accu_a) / 2
    calibration=abs(accu_c-accu_a)
    return round(accuracy.item(),4),round(calibration.item(),4)
    print("Accuracy : %.3f" % (accuracy * 100)+'%')
    print("Calibration : %.3f" % (calibration * 100)+'%')
```

Logistic Regression with Prejudice Regularizer

Prejudice Index

In [13]:
```
import torch as t

class PRLoss():#using linear
    def __init__(self, eta=1.0):
        super(PRLoss, self).__init__()
        self.eta = eta
    def forward(self,output_c,output_a):
        # For the mutual information,
        # eqn(9): Pr[y|s] = sum{(xi,si),si=s} sigma(xi,s) / D[xs]
        #D[xs]
        N_cau = t.tensor(output_c.shape[0])
        N_aa   = t.tensor(output_a.shape[0])
        Dxisi = t.stack((N_aa,N_cau),axis=0) # African-American sample (s0), #Caucasian sample (s1)
        # Pr[y|s]
        y_pred_cau = t.sum(output_c)
        y_pred_aa   = t.sum(output_a)
        P_ys = t.stack((y_pred_aa,y_pred_cau),axis=0) / Dxisi
        # eqn(10): Pr[y]~sum{(xi,si)} sigma(xi,si) / |D[xs]|
        P = t.cat((output_c,output_a),0)
        P_y = t.sum(P) / (train_X_a.shape[0]+train_X_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)
        # eqn(11) RPR
        # PI=sum{xi,si}sum{y}M*ln(Pr[y|si]/Pr[y])=sum{xi,si}sum{y}M*ln(Pr[Y,S]/(Pr[S]pR[Y]))
        PI_s1y1 = output_c * P_s1y1
        PI_s1y0 =(1- output_c) * P_s1y0
        PI_s0y1 = output_a * P_s0y1
        PI_s0y0 = (1- output_a )* 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
```

Prejudice Remover in Logistic Regression

In [14]:
```
import torch.nn as nn
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
```

In [15]:
```
class PRLR():
    def __init__(self, eta = 1, iters = 100, step = 0.01):
        super(PRLR, self).__init__()
        self.eta = eta
        self.step = step
        self.iters = iters

    def fit(self, X_train_c, Y_train_c, X_train_a, Y_train_a,
            X_valid_c, Y_valid_c, X_valid_a, Y_valid_a,
            X_test_c, Y_test_c, X_test_a, Y_test_a):
        modela = LogisticRegression(X_train_a)      # African-American
        modelc = LogisticRegression(X_train_c)      # Caucasian
        loss = nn.BCELoss(reduction='sum')
        iters = self.iters
        PI_term = PRLoss(eta = self.eta)
        #L2_optimizer = t.optim.Adam(list(np.abs(model0.parameters()) + np.abs(model1.parameters())), sel
        L2_optimizer = t.optim.Adam(list(modela.parameters())+list(modelc.parameters()), self.step, weigh


        train_losses = []
        val_losses = []
        for iter in range(iters):
            modela.train()
            modelc.train()
            L2_optimizer.zero_grad()

            ## sigmoid probability and loss
```

```
            output_a = modela(X_train_a)    # A-A
            output_c = modelc(X_train_c)
            # Loss_func is the sum of LogLoss and PI Loss
            loss_function_train = loss(output_c, Y_train_c) + loss(output_a, Y_train_a) + PI_term.forwarc
            loss_function_train.backward()
            L2_optimizer.step()
            train_losses.append(loss_function_train)


            output_a_valid = modela(X_valid_a)
            output_c_valid = modelc(X_valid_c)
            loss_function_val = loss(output_c_valid, Y_valid_c) + loss(output_a_valid, Y_valid_a) + PI_te

            val_losses.append(loss_function_val)


        modela.eval()
        modelc.eval()
        # accuracy
        accu = accuracy(modelc,modela,X_train_c,Y_train_c,X_train_a,Y_train_a)
        accu_val = accuracy(modelc,modela,X_valid_c,Y_valid_c,X_valid_a,Y_valid_a)
        accu_test = accuracy(modelc,modela,X_test_c,Y_test_c,X_test_a,Y_test_a)

        # PI index
        # pi_train = PI_term.forward(modela(X_train_a), modelc(X_train_c))
        # pi_valid = PI_term.forward(modela(X_valid_a), modelc(X_valid_c))
        # pi_test = PI_term.forward(modela(X_test_a), modelc(X_test_c))

        return accu, accu_val, accu_test
```

In [16]:
```
eta_value = [0.0,1.0,2.0,3.0,4.0,5.0,10.0,15.0,20.0,25.0]
accur = list()
accur_val = list()
accur_test = list()
# PI_train = list()
# PI_val = list()
# PI_test = list()
for i in range(0,len(eta_value)):
    #print("Theta Value: %d" % eta_value[e])
    PR = PRLR(eta = eta_value[i], iters = 3000, step = 0.01)
    accur_eta,accur_val_eta,accur_test_eta = PR.fit(train_X_c,train_Y_c,train_X_a,train_Y_a,
                                                    valid_X_c,valid_Y_c,valid_X_a,valid_Y_a,
                                                    test_X_c,test_Y_c,test_X_a,test_Y_a)
    accur.append(accur_eta)
    accur_val.append(accur_val_eta)
    accur_test.append(accur_test_eta)
```

In [17]:
```
#train
accur
```

Out[17]:
```
[(0.7041, 0.0187),
 (0.7066, 0.0181),
 (0.706, 0.0168),
 (0.7048, 0.0101),
 (0.705, 0.0105),
 (0.7052, 0.011),
 (0.7058, 0.008),
 (0.7067, 0.0126),
 (0.7042, 0.0176),
 (0.7058, 0.0135)]
```

In [18]:
```
#validation
accur_val
```

Out[18]:
```
[(0.6963, 0.0467),
 (0.6883, 0.0355),
 (0.6873, 0.024),
 (0.6856, 0.0207),
 (0.6811, 0.0162),
 (0.6844, 0.023),
 (0.6816, 0.022),
 (0.6816, 0.022),
 (0.6828, 0.0196),
 (0.6828, 0.0196)]
```

In [19]:
```
#test
accur_test
```

Out[19]:
```
[(0.689, 0.0525),
 (0.6798, 0.0154),
 (0.6827, 0.0166),
 (0.6757, 0.0027),
 (0.6775, 0.0062),
 (0.6763, 0.0085),
 (0.678, 0.0019),
 (0.6751, 0.0031),
 (0.6694, 0.0193),
 (0.6751, 0.0031)]
```

Final Model

In [20]:
```
PR_eta2 = PRLR(eta = 2.0, iters = 3000, step = 0.01)
PR_eta2.fit(train_X_c,train_Y_c,train_X_a,train_Y_a,
                                                    valid_X_c,valid_Y_c,valid_X_a,valid_Y_a,
                                                    test_X_c,test_Y_c,test_X_a,test_Y_a)
```

Out[20]:
```
((0.706, 0.0168), (0.6873, 0.024), (0.6827, 0.0166))
```

References

1. https://colab.research.google.com/github/sony/nnabla-examples/blob/master/interactive-demos/prejudice_remover_regularizer.ipynb#scrollTo=r45NcxtY6OzB

2. Toshihiro Kamishima, Shotaro Akaho, Hideki Asoh & Jun Sakuma. "Fairness-aware classifier with prejudice remover regularizer." Joint European Conference on Machine Learning and Knowledge Discovery in Databases ECML PKDD 2012: Machine Learning and Knowledge Discovery in Databases pp 35–50.