main

April 13, 2022

[1]:
```python
# Import required packages
import numpy as np
import pandas as pd
import cv2 as cv2
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, roc_auc_score,␣
 ↪accuracy_score, f1_score, confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
import time
import scipy.optimize as optim
import copy
import random
import pickle
from IPython.display import Markdown, display
import seaborn as sns
import matplotlib.patches as patches
from tabulate import tabulate
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
import tempfile
import os
import subprocess
```

0.1 1) Load the dataset

For this project we are using the COMPAS-scores-two-years dataset, a COMPAS dataset that contains the criminal history, jail and prison time, demographics, and COMPAS risk scores for defendants from Broward county from 2013 and 2014, as well as the ground truth on whether or not these individuals actually recidivated within two years after the screening.

There are 7214 data in total.

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

[3]:
```python
raw_data.shape
```

[3]: (7214, 53)

0.2 2) Data processing

0.2.1 2.1) Subset of data with race "African-American" or "Caucasian"

We want to keep only the rows of the dataset that correspond to "African-American" or "Caucasian" race.

[4]:
```
print("The dataset includes defendants of the following races: {}".
↪format(raw_data['race'].unique()))
```

```
The dataset includes defendants of the following races: ['Other' 'African-
American' 'Caucasian' 'Hispanic' 'Native American'
 'Asian']
```

[5]:
```
processed_data = raw_data.loc[raw_data['race'].isin(["African-American",␣
↪"Caucasian"])]
```

[6]:
```
print("The original dataset includes {} African-American and Caucasian␣
↪defendants.".format(processed_data.shape[0]))
```

```
The original dataset includes 6150 African-American and Caucasian defendants.
```

[7]:
```
processed_data
```

[7]:
```
         id               name      first     last compas_screening_date  \
1         3         kevon dixon      kevon    dixon            2013-01-27
2         4            ed philo         ed    philo            2013-04-14
3         5         marcu brown      marcu    brown            2013-01-13
6         8      edward riddle      edward   riddle            2014-02-19
8        10   elizabeth thieme   elizabeth   thieme            2014-03-16
…       …                  …          …       …                    …
7207  10994       jarred payne     jarred    payne            2014-05-10
7208  10995       raheem smith     raheem    smith            2013-10-20
7209  10996      steven butler     steven   butler            2013-11-23
7210  10997    malcolm simmons    malcolm  simmons            2014-02-01
7212  11000        farrah jean     farrah     jean            2014-03-09

         sex         dob  age          age_cat              race  …  \
1       Male  1982-01-22   34          25 - 45  African-American  …
2       Male  1991-05-14   24     Less than 25  African-American  …
3       Male  1993-01-21   23     Less than 25  African-American  …
6       Male  1974-07-23   41          25 - 45         Caucasian  …
8     Female  1976-06-03   39          25 - 45         Caucasian  …
…        …           …  …              …                 …   …
7207    Male  1985-07-31   30          25 - 45  African-American  …
7208    Male  1995-06-28   20     Less than 25  African-American  …
7209    Male  1992-07-17   23     Less than 25  African-American  …
7210    Male  1993-03-25   23     Less than 25  African-American  …
7212  Female  1982-11-17   33          25 - 45  African-American  …
```

```
      v_decile_score  v_score_text  v_screening_date  in_custody  out_custody  \
1                  1           Low        2013-01-27  2013-01-26   2013-02-05
2                  3           Low        2013-04-14  2013-06-16   2013-06-16
3                  6        Medium        2013-01-13         NaN          NaN
6                  2           Low        2014-02-19  2014-03-31   2014-04-18
8                  1           Low        2014-03-16  2014-03-15   2014-03-18
…                …             …                 …          …           …
7207               2           Low        2014-05-10  2015-10-22   2015-10-22
7208               9          High        2013-10-20  2014-04-07   2014-04-27
7209               5        Medium        2013-11-23  2013-11-22   2013-11-24
7210               5        Medium        2014-02-01  2014-01-31   2014-02-02
7212               2           Low        2014-03-09  2014-03-08   2014-03-09

      priors_count.1 start   end event two_year_recid
1                  0     9   159     1              1
2                  4     0    63     0              1
3                  1     0  1174     0              0
6                 14     5    40     1              1
8                  0     2   747     0              0
…                …   …   …   …                …
7207               0     0   529     1              1
7208               0     0   169     0              0
7209               0     1   860     0              0
7210               0     1   790     0              0
7212               3     0   754     0              0

[6150 rows x 53 columns]
```

0.2.2 2.2) Remove unuseful data

Remove unuseful columns (columns with multiple missing data).

[8]:
```
processed_data = processed_data[['sex', 'age', 'age_cat', 'race',␣
↪'decile_score', 'juv_fel_count', 'juv_misd_count', 'juv_other_count',
             'priors_count', 'days_b_screening_arrest', 'c_jail_in',␣
↪'c_jail_out', 'c_charge_degree', 'is_recid',
              'score_text', 'two_year_recid']]
```

[9]:
```
processed_data
```

[9]:
```
         sex  age          age_cat              race  decile_score  \
1       Male   34          25 - 45  African-American             3
2       Male   24  Less than 25  African-American             4
3       Male   23  Less than 25  African-American             8
6       Male   41          25 - 45         Caucasian             6
8     Female   39          25 - 45         Caucasian             1
…        …   …              …                 …             …
```

```
7207    Male   30       25 - 45  African-American            2
7208    Male   20  Less than 25  African-American            9
7209    Male   23  Less than 25  African-American            7
7210    Male   23  Less than 25  African-American            3
7212  Female   33       25 - 45  African-American            2

      juv_fel_count  juv_misd_count  juv_other_count  priors_count  \
1                 0               0                0             0
2                 0               0                1             4
3                 0               1                0             1
6                 0               0                0            14
8                 0               0                0             0
…             …             …               …             …
7207              0               0                0             0
7208              0               0                0             0
7209              0               0                0             0
7210              0               0                0             0
7212              0               0                0             3

      days_b_screening_arrest            c_jail_in           c_jail_out  \
1                        -1.0  2013-01-26 03:45:27  2013-02-05 05:36:53
2                        -1.0  2013-04-13 04:58:34  2013-04-14 07:02:04
3                         NaN                  NaN                  NaN
6                        -1.0  2014-02-18 05:08:24  2014-02-24 12:18:30
8                        -1.0  2014-03-15 05:35:34  2014-03-18 04:28:46
…                      …                  …                  …
7207                     -1.0  2014-05-09 10:01:33  2014-05-10 08:28:12
7208                     -1.0  2013-10-19 11:17:15  2013-10-20 08:13:06
7209                     -1.0  2013-11-22 05:18:27  2013-11-24 02:59:20
7210                     -1.0  2014-01-31 07:13:54  2014-02-02 04:03:52
7212                     -1.0  2014-03-08 08:06:02  2014-03-09 12:18:04

     c_charge_degree  is_recid score_text  two_year_recid
1                  F         1        Low               1
2                  F         1        Low               1
3                  F         0       High               0
6                  F         1     Medium               0
8                  M         0        Low               0
…               …       …         …             …
7207               M         1        Low               1
7208               F         0       High               0
7209               F         0     Medium               0
7210               F         0        Low               0
7212               M         0        Low               0

[6150 rows x 16 columns]
```

According to the ProPublica COMPAS notebook (https://github.com/propublica/compasanalysis/blob/master/Compas%20Analysis.ipynb) there are a number of reasons to remove rows because of missing data: - If the charge date of a defendants Compas scored crime was not within 30 days from when the person was arrested, we can assume that because of data quality reasons, that we do not have the right offense. - The recidivist flag (is_recid) should be -1 if we could not find a compas case at all. - Ordinary traffic offenses (c_charge_degree = 'O') will not result in Jail time and hence are removed (only two of them). - We filtered the underlying data from Broward county to include only those rows representing people who had either recidivated in two years, or had at least two years outside of a correctional facility.

[10]:
```
# If the charge date of a defendants Compas scored crime was not within 30 days
↪from when the person was arrested,
# we can assume that because of data quality reasons, that we do not have the
↪right offense.

processed_data = processed_data.loc[processed_data['days_b_screening_arrest']
↪<= 30]
processed_data = processed_data.loc[processed_data['days_b_screening_arrest']
↪>= -30]
```

[11]:
```
# The recidivist flag (is_recid) should be -1 if we could not find a compas
↪case at all.

processed_data = processed_data.loc[processed_data['is_recid'] != -1]
```

[12]:
```
# Ordinary traffic offenses (c_charge_degree = 'O') will not result in Jail
↪time and hence are removed
# (only two of them).

processed_data = processed_data.loc[processed_data['c_charge_degree'] != 'O']
```

[13]:
```
# score_text shouldn't be 'N/A'

processed_data = processed_data.loc[processed_data['score_text'] != 'N/A']
```

[14]:
```
processed_data['length_of_stay'] = (pd.
↪to_datetime(processed_data['c_jail_out'])-pd.
↪to_datetime(processed_data['c_jail_in'])).apply(lambda x: x.days)
```

[15]:
```
processed_data = processed_data.drop(columns=['c_jail_in', 'c_jail_out'])
```

0.2.3 2.3) Create indicator values out of columns

[16]:
```
# replace the values of the sensitive attribute race as follows: Caucasian ->
↪1, African-American -> 0
processed_data = processed_data.replace({'race': 'Caucasian'}, 1)
processed_data = processed_data.replace({'race': 'African-American'}, 0)
```

```
[17]: # replace the values of sex as follows
processed_data = processed_data.replace({'sex': 'Male'}, 1)
processed_data = processed_data.replace({'sex': 'Female'}, 0)

# replace the values of age_cat as follows
processed_data = processed_data.replace({'age_cat': 'Less than 25'}, 0)
processed_data = processed_data.replace({'age_cat': '25 - 45'}, 1)
processed_data = processed_data.replace({'age_cat': 'Greater than 45'}, 2)

# replace the values of c_charge_degree as follows
processed_data = processed_data.replace({'c_charge_degree': 'F'}, 0)
processed_data = processed_data.replace({'c_charge_degree': 'M'}, 1)

# replace the values of score_text as follows
processed_data = processed_data.replace({'score_text': 'Low'}, 0)
processed_data = processed_data.replace({'score_text': 'Medium'}, 1)
processed_data = processed_data.replace({'score_text': 'High'}, 2)
```

0.2.4 2.4) Check for NaN values

```
[18]: # check whether there are NaN values in the final dataset as well as the number␣
 ↪of unique values per column

unique_NAN_df = pd.DataFrame(columns=['column name', '# of unique values', '#␣
 ↪of NaN values'])
for item in processed_data.columns:
    unique_NAN_df = unique_NAN_df.append({
        'column name': item,
        '# of unique values': len(processed_data[item].unique()),
        '# of NaN values': sum(processed_data[item].isna() == True)},␣
 ↪ignore_index = True)

unique_NAN_df = unique_NAN_df.style.hide_index()
unique_NAN_df
```

```
[18]: <pandas.io.formats.style.Styler at 0x7ff02b888eb0>
```

```
[19]: # move two_year_recid to the end

cols = list(processed_data.columns.values)
cols.pop(cols.index('two_year_recid'))
processed_data = processed_data[cols+['two_year_recid']]
```

```
[20]: # move race to the first column

race_column = processed_data.pop('race')
processed_data.insert(0, 'race', race_column)
```

[21]:
```
processed_data
```

[21]:
```
      race  sex  age  age_cat  decile_score  juv_fel_count  juv_misd_count  \
1        0    1   34        1             3              0               0
2        0    1   24        0             4              0               0
6        1    1   41        1             6              0               0
8        1    0   39        1             1              0               0
10       1    1   27        1             4              0               0
…      …    …    …        …             …              …
7207     0    1   30        1             2              0               0
7208     0    1   20        0             9              0               0
7209     0    1   23        0             7              0               0
7210     0    1   23        0             3              0               0
7212     0    0   33        1             2              0               0

      juv_other_count  priors_count  days_b_screening_arrest  c_charge_degree  \
1                   0             0                     -1.0                0
2                   1             4                     -1.0                0
6                   0            14                     -1.0                0
8                   0             0                     -1.0                1
10                  0             0                     -1.0                0
…                 …             …                       …              …
7207                0             0                     -1.0                1
7208                0             0                     -1.0                0
7209                0             0                     -1.0                0
7210                0             0                     -1.0                0
7212                0             3                     -1.0                1

      is_recid  score_text  length_of_stay  two_year_recid
1            1           0              10               1
2            1           0               1               1
6            1           1               6               1
8            0           0               2               0
10           0           0               1               0
…          …           …              …              …
7207         1           0               0               1
7208         0           2               0               0
7209         0           1               1               0
7210         0           0               1               0
7212         0           0               1               0

[5278 rows x 15 columns]
```

[22]:
```
processed_data = processed_data.drop(columns=['age', 'juv_fel_count',␣
↪'juv_misd_count', 'juv_other_count'])
```

[23]:
```
# save final data set to csv

processed_data.to_csv("../output/processed-compas-scores-two-years.csv",␣
↪index=False)
```

0.3 3) Split data

We will first get the labels and the sensitive data.

[24]:
```
data = np.array(processed_data)
y = np.array(data[:,-1]).flatten()
data = data[:,:-1]
sensitive = data[:,0]
data = preprocessing.scale(data)
data = data[:,1:]
```

Split data into sensitive and nonsensitive data (sensitive –> race: Caucasian)

[25]:
```
sensitive_idx = np.array(np.where(sensitive==1))[0].flatten()
nonsensitive_idx = np.array(np.where(sensitive!=1))[0].flatten()
data_sensitive = data[sensitive_idx,:]
data_nonsensitive = data[nonsensitive_idx,:]
y_sensitive = y[sensitive_idx]
y_nonsensitive = y[nonsensitive_idx]
```

Split data into training, validation, and testing sets (training: validation: testing = 6:2:2).

[26]:
```
# split sensitive data into training, validation, and testing sets

X_train_s, X_test_s, y_train_s, y_test_s = train_test_split(data_sensitive,␣
↪y_sensitive, test_size= 0.2, random_state=42)
X_train_s, X_valid_s, y_train_s, y_valid_s = train_test_split(X_train_s,␣
↪y_train_s, test_size = 0.25, random_state=42)
```

[27]:
```
# split non-sensitive data into training, validation, and testing sets

X_train_n, X_test_n, y_train_n, y_test_n = train_test_split(data_nonsensitive,␣
↪y_nonsensitive, test_size= 0.2, random_state=42)
X_train_n, X_valid_n, y_train_n, y_valid_n = train_test_split(X_train_n,␣
↪y_train_n, test_size = 0.25, random_state=42)
```

[28]:
```
# create final training, validation, and testing sets

X_train = np.concatenate((X_train_s, X_train_n))
X_valid = np.concatenate((X_valid_s, X_valid_n))
X_test = np.concatenate((X_test_s, X_test_n))

Y_train = np.concatenate((y_train_s, y_train_n))
```

```
Y_valid = np.concatenate((y_valid_s, y_valid_n))
Y_test = np.concatenate((y_test_s, y_test_n))
```

0.4 4) Learning Fair Representations (LFR)

The goal in LFR model is to learn a good prototype set Z such that:

1. the mapping from X_0 to Z satisfies statistical parity;
2. the mapping to Z-space retains information in X (except for membership in the protected set); and
3. the induced mapping from X to Y (by first mapping each $\mathbf{x}$ probabilistically to Z-space, and then mapping Z to Y) is close to f.

The objective function is: minimize $L = A_z * L_z + A_x * L_x + A_y * L_y$

, where A_x, A_y, A_z are hyper-parameters governing the trade-off between the system desiderata.

The model is defined in LFR.py in the lib folder, which was adapted from https://github.com/zjelveh/learning-fair-representations.

[29]:
```
import sys

sys.path.append('../lib/')
import LFR

sys.path.append('../lib/')
from EvalMetrics import *

sys.path.append('../lib/')
%run '../lib/LFR.py'

sys.path.append('../lib/')
%run '../lib/EvalMetrics.py'
```

0.4.1 4.1) Check for best number of interations (cross validation) and save best (trained) model

[37]:
```
iter_max = 1500

model_train_time = []
train_Accuracy = []
val_Accuracy = []
train_Calibration = []
val_Calibration = []

best_accuracy = 0

for i in range(100, iter_max+100, 100):
```

```
    #model training
    start = time.time()
    #random.seed(1024); np.random.seed(1024)
    final_parameters = LFR(X_train_s, X_train_n, y_train_s, y_train_n, 10,␣
↪1e-4, 0.1, 1000, iter = i)
    model_train_time.append(time.time() - start)

    #Train set accuracy and calibration
    pred_train_s, pred_train_n = predict(final_parameters, X_train_s,␣
↪X_train_n, 10)
    acc_sen, acc_nsen, total_accuracy = calc_accuracy(pred_train_s,␣
↪pred_train_n, y_train_s, y_train_n)
    train_Accuracy.append(total_accuracy)

    calibration = calc_calibration(acc_sen, acc_nsen)
    train_Calibration.append(calibration)

    #Validation set accuracy and calibration
    pred_val_s, pred_val_n = predict(final_parameters, X_valid_s, X_valid_n, 10)
    acc_sen, acc_nsen, total_accuracy = calc_accuracy(pred_val_s, pred_val_n,␣
↪y_valid_s, y_valid_n)
    val_Accuracy.append(total_accuracy)

    calibration = calc_calibration(acc_sen, acc_nsen)
    val_Calibration.append(calibration)

    if total_accuracy > best_accuracy:
    best_accuracy = total_accuracy
    best_model = copy.deepcopy(final_parameters)

    print("Finished for " + str(i) + " iterations in " + str(time.time() -␣
↪start) + " secs")
```

```
Finished for 100 iterations in 306.8652307987213 secs
Finished for 200 iterations in 447.8896050453186 secs
Finished for 300 iterations in 444.3408987522125 secs
Finished for 400 iterations in 591.8047559261322 secs
Finished for 500 iterations in 741.7820808887482 secs
Finished for 600 iterations in 887.2942533493042 secs
Finished for 700 iterations in 1031.751314163208 secs
Finished for 800 iterations in 1029.604287147522 secs
Finished for 900 iterations in 1361.9073469638824 secs
Finished for 1000 iterations in 1616.1541967391968 secs
Finished for 1100 iterations in 1456.9709899425507 secs
Finished for 1200 iterations in 1601.153074979782 secs
Finished for 1300 iterations in 1756.151284456253 secs
Finished for 1400 iterations in 1909.3943445682526 secs
```

```
Finished for 1500 iterations in 2208.3647351264954 secs
```

[39]:
```python
# from google.colab import drive
# drive.mount('/content/drive', force_remount=True)
```

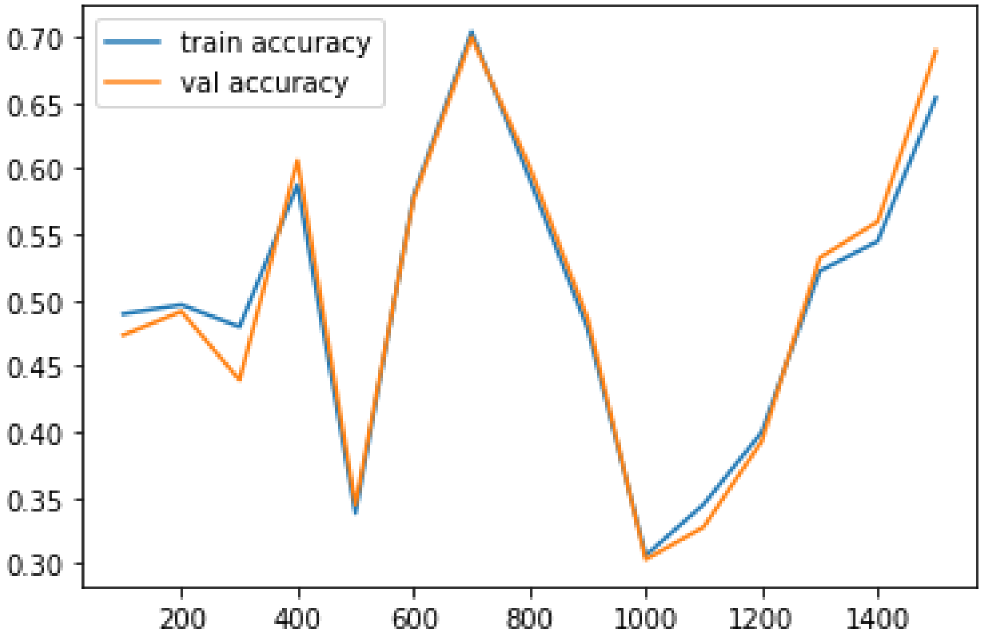
```
Mounted at /content/drive
```

[40]:
```python
import pickle

filename = 'best_model.sav'
pickle.dump(best_model, open(filename, 'wb'))

# !cp "best_model.sav" "/content/drive/My Drive/Colab Notebooks/ADS Proj 4"
```

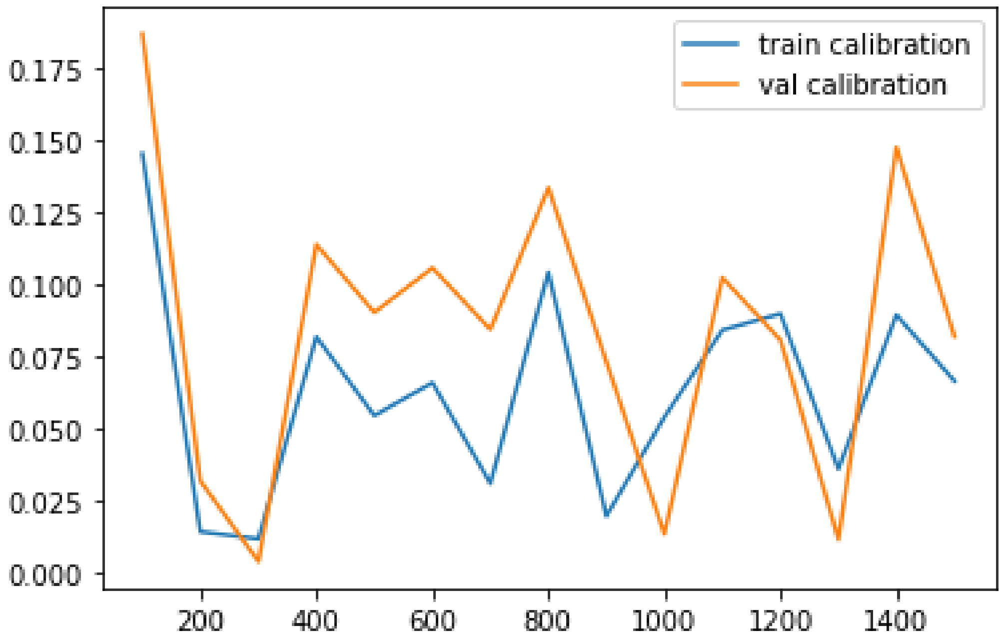
[41]:
```python
iterations = [i for i in range(100, iter_max+100, 100)]

print(iterations)
print(model_train_time)
print(train_Accuracy)
print(val_Accuracy)
print(train_Calibration)
print(val_Calibration)
```

```
[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400,
1500]
[305.7239451408386, 446.75790548324585, 443.2332410812378, 590.6827621459961,
740.6412780284882, 886.1439714431763, 1030.6187875270844, 1028.476889848709,
1360.7818098068237, 1615.0316441059113, 1455.8666932582855, 1600.0446200370789,
1755.049386024475, 1908.2708160877228, 2207.2562866210938]
[0.4895767530006317, 0.4965255843335439, 0.47978521794061907,
0.5874921036007581, 0.3382817435249526, 0.5789639924194567, 0.7037271004421983,
0.5922299431459255, 0.47820593809222994, 0.3060644346178143, 0.344914718888187,
0.3998736576121289, 0.5221099178774479, 0.5448515476942514, 0.6538218572331017]
[0.4734848484848485, 0.4914772727272727, 0.4393939393939394, 0.6060606060606061,
0.3446969696969697, 0.5767045454545454, 0.6998106060606061, 0.6013257575757576,
0.48579545454545453, 0.30303030303030304, 0.32765151515151514,
0.39299242424242425, 0.5321969696969697, 0.5596590909090909, 0.6893939393939394]
[0.14544470600968695, 0.01434099088129448, 0.011873674394982936,
0.08194055045260495, 0.05459858754769059, 0.06599478395890446,
0.03110392327049527, 0.10437951798451839, 0.019792648837214155,
0.05410570704831602, 0.08426674659323413, 0.08993820260968571,
0.03608642892675684, 0.08954481403543824, 0.06659714720433929]
[0.18698636542166203, 0.031948678624198146, 0.004009950062655421,
0.1139544017805375, 0.09038846391231975, 0.1058335047786485,
0.08445209194456393, 0.13367871771372997, 0.07315540426805317,
0.013526100211345315, 0.10247068285110444, 0.08077879813716876,
0.011633343931771067, 0.14766865543232277, 0.08202442628163165]
```

[42]:
```
import matplotlib.pyplot as plt

plt.plot(iterations, train_Accuracy, label = "train accuracy")
plt.plot(iterations, val_Accuracy, label = "val accuracy")
plt.legend()
plt.savefig("accuracy.png")
plt.show()

# !cp "accuracy.png" "/content/drive/My Drive/Colab Notebooks/ADS Proj 4"
```



[43]:
```
plt.plot(iterations, train_Calibration, label = "train calibration")
plt.plot(iterations, val_Calibration, label = "val calibration")
plt.legend()
plt.savefig("calibration.png")
plt.show()

# !cp "calibration.png" "/content/drive/My Drive/Colab Notebooks/ADS Proj 4"
```



[44]:
```
# Saving the iterations data

df = pd.DataFrame(list(zip(iterations, model_train_time, train_Accuracy,␣
↪val_Accuracy, train_Calibration, val_Calibration)),
                  columns =['iterations', 'model_train_time', 'train_Accuracy',␣
↪'val_Accuracy', 'train_Calibration', 'val_Calibration'])
```

[45]:
```
df.to_csv("All_iterations_info.csv", index=False)

# !cp "All_iterations_info.csv" "/content/drive/My Drive/Colab Notebooks/ADS␣
↪Proj 4"
```

0.4.2 4.2) Read saved best LFR model (trained)

[30]:
```
sys.path.append('../lib/')
%run '../lib/LFR.py'

sys.path.append('../lib/')
%run '../lib/EvalMetrics.py'
```

[31]:
```
filename = '../output/best_model.sav'
loaded_model_LFR = pickle.load(open(filename, 'rb'))
```

0.4.3 4.3) Accuracy and calibration of LFR on Training and Validations Sets

[32]:
```
# get predictions for the training dataset

pred_train_s, pred_train_n = predict(loaded_model_LFR, X_train_s, X_train_n, 10)

# get accuracy for the training dataset

acc_sen, acc_nsen, total_accuracy = calc_accuracy(pred_train_s, pred_train_n,␣
 ↪y_train_s, y_train_n)

print("The accuracy for Caucasians is: ", acc_sen)
print("The accuracy for African-Americans is: ", acc_nsen)
print("The total accuracy is: ", total_accuracy)

# get calibration for the training dataset

calibration = calc_calibration(acc_sen, acc_nsen)

print("The calibration is: ", calibration)
```

```
The accuracy for Caucasians is:  0.7224425059476606
The accuracy for African-Americans is:  0.6913385826771653
The total accuracy is:  0.7037271004421983
The calibration is:  0.03110392327049527
```

[33]:
```
# get predictions for the validation dataset

pred_val_s, pred_val_n = predict(loaded_model_LFR, X_valid_s, X_valid_n, 10)

# get accuracy for the validation dataset

acc_sen, acc_nsen, total_accuracy = calc_accuracy(pred_val_s, pred_val_n,␣
 ↪y_valid_s, y_valid_n)

print("The accuracy for Caucasians is: ", acc_sen)
print("The accuracy for African-Americans is: ", acc_nsen)
print("The total accuracy is: ", total_accuracy)

# get calibration for the validation dataset

calibration = calc_calibration(acc_sen, acc_nsen)

print("The calibration is: ", calibration)
```

```
The accuracy for Caucasians is:  0.7505938242280285
The accuracy for African-Americans is:  0.6661417322834645
The total accuracy is:  0.6998106060606061
```

```
The calibration is:  0.08445209194456393
```

0.4.4 4.4) Evaluation of LFR on Test data

We will now evaluate the model using the test data (10% of the data).

[34]:
```
# get predictions for the testing dataset

pred_LFR_test_s, pred_LFR_test_n = predict(loaded_model_LFR, X_test_s,␣
 ↪X_test_n, 10)
```

[35]:
```
sys.path.append('../lib/')
%run '../lib/EvalMetrics.py'
```

LFR accuracy and f1-score on sensitive, nonsensitive, and all data.

[36]:
```
pred_LFR_test = np.concatenate((pred_LFR_test_s, pred_LFR_test_n))
y_test = np.concatenate((y_test_s, y_test_n))

plot_model_performance(pred_LFR_test_s, pred_LFR_test_n, pred_LFR_test,␣
 ↪y_test_s, y_test_n, y_test)
```

Sensitive data (Caucasians):

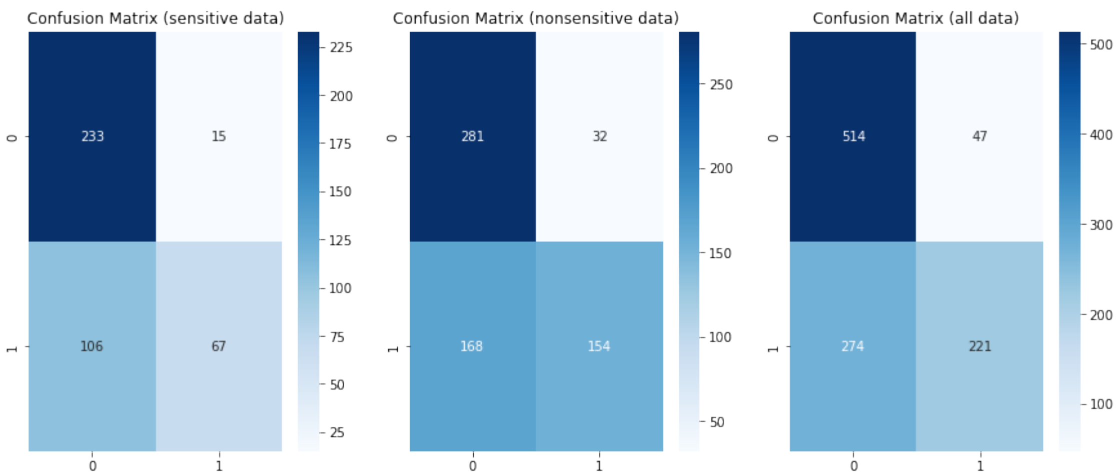
```
Accuracy: 0.7125890736342043
F1 score: 0.5254901960784313
```

Nonsensitive data (African-Americans):

```
Accuracy: 0.6850393700787402
F1 score: 0.6062992125984252
```

All data:

```
Accuracy: 0.6960227272727273
F1 score: 0.5792922673656619
```



LFR bias metrics

[37]:
```
fair_metrics_LFR = fair_metrics(pred_LFR_test_s, pred_LFR_test_n,␣
↪pred_LFR_test, y_test_s, y_test_n, y_test)
```

[38]:
```
plot_fair_metrics(fair_metrics_LFR)
display(fair_metrics_LFR)
```

0.4.5 Check bias metrics :

A model can be considered bias if just one of these four metrics show that this model is biased.

For the Race attribute : With default thresholds, bias against unprivileged group detected in **0** out of 4 metrics

```
           calibration  equal_opportunity_difference  \
objective      0.00000                      0.000000
Race           0.02755                     -0.083375

           average_abs_odds_difference  disparate_impact
objective                      0.00000          1.000000
Race                           0.04907          1.040216
```

0.5 5) Fairness-aware Classifier with Prejudice Remover Regularizer

This PR model is an in-processing technique that adds a discrimination-aware regularization term to the learning objective.

In this model, parameters are estimated based on maximum likehood principle.

[39]:
```
import sys

sys.path.append('../lib/')
import LFR
```

```
sys.path.append('../lib/')
from EvalMetrics import *

sys.path.append('../lib/')
%run '../lib/LFR.py'

sys.path.append('../lib/')
%run '../lib/EvalMetrics.py'
```

0.5.1 5.1) Import libraries & Reconstruct the dataset

The **PrejudiceRemover** function inputs *StandardDataset*, so we need to process the dataset in a different way than above methods.

This class is very loosely based on code from https://github.com/algofairness/fairness-comparison.

[35]:
```
# pip install aif360
```

[37]:
```
# pip install fairlearn
```

[40]:
```
# Libraries to study
from aif360.datasets import StandardDataset
from aif360.algorithms.preprocessing import LFR, Reweighing
from aif360.algorithms.inprocessing import AdversarialDebiasing,␣
↪PrejudiceRemover
```

[41]:
```
privileged_race = np.array([0]) # African-American
privileged_sex = np.array([1]) # Male

data_orig = StandardDataset(processed_data,
                              label_name='two_year_recid',
                              favorable_classes=[1],
                              protected_attribute_names=['race', 'sex'],
                             ␣
↪privileged_classes=[privileged_race,privileged_sex]
                              )

def meta_data(dataset):
    # print out some labels, names, etc.
    display(Markdown("#### Dataset shape"))
    print(dataset.features.shape)
    display(Markdown("#### Favorable and unfavorable labels"))
    print(dataset.favorable_label, dataset.unfavorable_label)
    display(Markdown("#### Protected attribute names"))
    print(dataset.protected_attribute_names)
    display(Markdown("#### Privileged and unprivileged protected attribute␣
↪values"))
```

```
    print(dataset.privileged_protected_attributes, dataset.
 ↪unprivileged_protected_attributes)
    display(Markdown("#### Dataset feature names"))
    print(dataset.feature_names)

meta_data(data_orig)
```

Dataset shape

(5278, 10)

Favorable and unfavorable labels

1.0 0.0

Protected attribute names

['race', 'sex']

Privileged and unprivileged protected attribute values

[array([0.]), array([1.])] [array([1.]), array([0.])]

Dataset feature names

['race', 'sex', 'age_cat', 'decile_score', 'priors_count', 'days_b_screening_arrest', 'c_charge_degree', 'is_recid', 'score_text', 'length_of_stay']

[42]:
```
np.random.seed(42)

data_train, data_test = data_orig.split([0.8], shuffle=True) # train:test = 5:1
# data_train, data_valid = data_train.split([0.75], shuffle=True) # 5:1

display(Markdown("#### Train Dataset shape"))
print("Perpetrator Sex :",data_train.features.shape)
# display(Markdown("#### Validation Dataset shape"))
# print("Perpetrator Sex :",data_valid.features.shape)
display(Markdown("#### Test Dataset shape"))
print("Perpetrator Sex :",data_test.features.shape)
```

Train Dataset shape

Perpetrator Sex : (4222, 10)

Test Dataset shape

Perpetrator Sex : (1056, 10)

[43]:
```python
from time import time
t0 = time()
debiased_model = PrejudiceRemover(sensitive_attr="race", eta = 25.0)
debiased_model.fit(data_train)
```

```
/Applications/anaconda3/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[43]:
```
<aif360.algorithms.inprocessing.prejudice_remover.PrejudiceRemover at
0x7ff01638c9d0>
```

[44]:
```python
a=debiased_model.predict(data_test).features
np.shape(a)
test_race = pd.DataFrame(a).iloc[:,0]
test_race = pd.DataFrame(test_race ).rename(columns={0: 'race'})
# test_race

test_true_y=pd.DataFrame(data_test.labels.ravel()).iloc[:,0] #true y
test_true_y = pd.DataFrame(test_true_y).rename(columns={0: 'y_true'})
# test_true_y

test_pred=pd.DataFrame(debiased_model.predict(data_test).scores>= 0.5).
 ↪astype(float) # predicted y
test_pred = test_pred.rename(columns={0: 'y_pred'})
# test_pred
```

[45]:
```python
df = test_race.join(test_true_y,how="left")
df = df.join(test_pred,how="left")
df
```

[45]:

	race	y_true	y_pred
0	0.0	1.0	1.0
1	0.0	1.0	1.0
2	0.0	1.0	1.0
3	0.0	0.0	0.0
4	0.0	1.0	1.0
…	…	…	…
1051	0.0	1.0	1.0
1052	1.0	0.0	0.0

```
1053    0.0     1.0     1.0
1054    0.0     1.0     1.0
1055    0.0     1.0     1.0

[1056 rows x 3 columns]
```

[46]:
```
pred_PR_test_s = df['y_pred'][df['race']==1]
pred_PR_test_n = df['y_pred'][df['race']==0]
pred_PR_test = df['y_pred']
y_PR_test_s = df['y_true'][df['race']==1]
y_PR_test_n = df['y_true'][df['race']==0]
y_PR_test = df['y_true']
```

[47]:
```
pred_PR_test_s=np.array(pred_PR_test_s)
pred_PR_test_n=np.array(pred_PR_test_n)
pred_PR_test=np.array(pred_PR_test)

y_PR_test_s=np.array(y_PR_test_s)
y_PR_test_n=np.array(y_PR_test_n)
y_PR_test=np.array(y_PR_test)
```

0.5.2 5.2) Evaluation of PR on Test data

[56]:
```
sys.path.append('../lib/')
%run '../lib/EvalMetrics.py'
```

PR accuracy and f1-score on sensitive, nonsensitive, and all data.

[49]:
```
plot_model_performance(pred_PR_test_s, pred_PR_test_n, pred_PR_test,␣
↪y_PR_test_s, y_PR_test_n, y_PR_test)
```

Sensitive data (Caucasians):

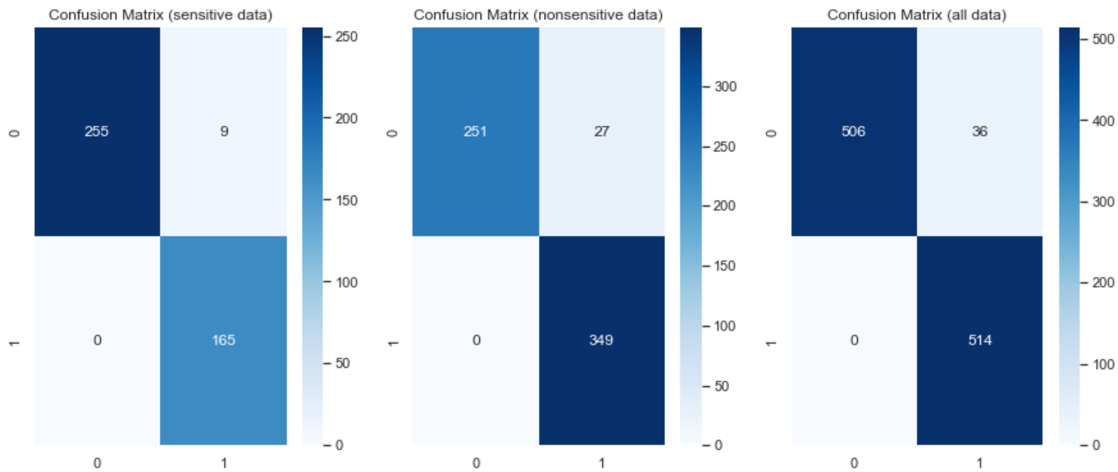
```
Accuracy: 0.9790209790209791
F1 score: 0.9734513274336283
```

Nonsensitive data (African-Americans):

```
Accuracy: 0.9569377990430622
F1 score: 0.9627586206896551
```

All data:

```
Accuracy: 0.9659090909090909
F1 score: 0.9661654135338346
```



PR bias metrics

[50]:
```
fair_metrics_PR = fair_metrics(pred_PR_test_s, pred_PR_test_n, pred_PR_test,␣
↪y_PR_test_s, y_PR_test_n, y_PR_test)
```

[51]:
```
plot_fair_metrics(fair_metrics_PR)
display(fair_metrics_PR)
```

0.5.3 Check bias metrics :

A model can be considered bias if just one of these four metrics show that this model is biased.

For the Race attribute : With default thresholds, bias against unprivileged group detected in **1** out of 4 metrics

```
           calibration  equal_opportunity_difference  \
objective     0.000000                      0.000000
Race          0.022083                     -0.172003

           average_abs_odds_difference  disparate_impact
objective                     0.000000          1.000000
Race                          0.097043          1.023077
```

0.6 6) LFR vs PR

We use 5 evaluation metrics to compare the two algorithms:

- Accuracy;
- Calibration: a difference between the accuracy in the privileged group and unprivilidged group

  $(= 1| =)-(= 1| =)$
- Equal Opportunity Difference : a difference between the true positive rate of privileged group and the true positive rate of unprivileged group
- Average Absolute Odds Difference: using both false positive rate and true positive rate to calculate the bias
- Disparate Impact

These evaluation metrics are defined in EvalMetrics.py in the lib folder. The results are displayed as follows.

[57]:
```
compare_models(pred_LFR_test_s, pred_LFR_test_n, pred_PR_test_s,␣
↪pred_PR_test_n, y_test_s, y_test_n, y_PR_test_s, y_PR_test_n,
                  fair_metrics_LFR, fair_metrics_PR, 'LFR', 'PR')
```

```
metric                              LFR          PR
----------------------------  ----------  ----------
accuracy                       0.696023    0.965909
calibration                    0.0275497   0.0220832
equal_opportunity_difference  -0.0833748  -0.172003
average_abs_odds_difference    0.0490695   0.0970433
disparate_impact               1.04022     1.02308
```

Comparison:

- PR (A5) model demonstrated better performance in trade-off between accuracy and bias than LFR (A1) model.
- PR method is inferior to LFR in equal opportunity difference and average absolute odds difference.