# Trustworthy AI & Fairness

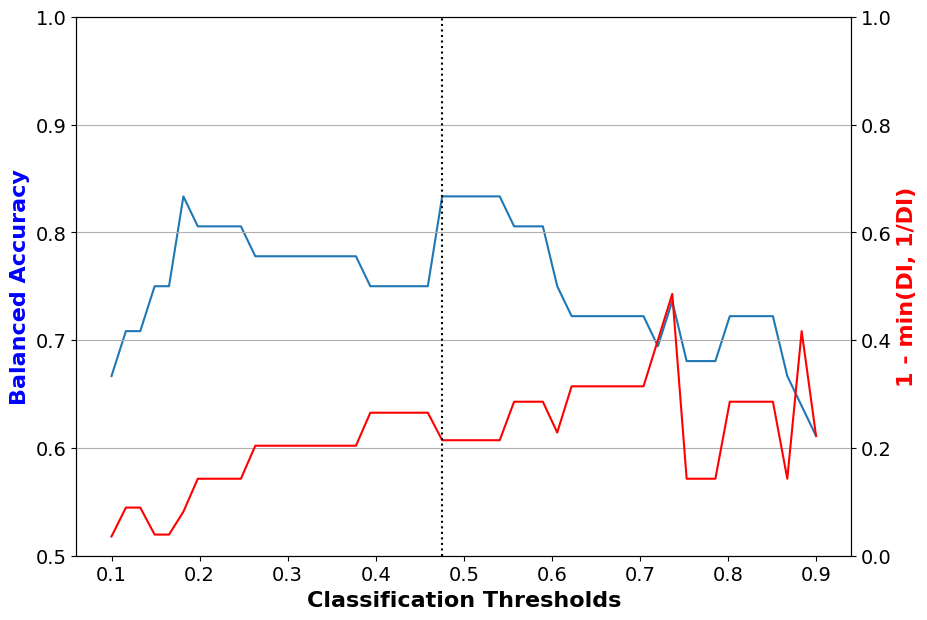
In this part we take a student mental health dataset- [data](https://www.kaggle.com/datasets/shariful07/student-mental-health)  and try to predict the depression levels of students while applying fairness metrics and trustworthy AI to help avoid any kind of bias in the dataset or model. Here we will make use of the python version of the AI Fairness 360 Toolkit which is an open-source library containing ample number of techniques to help detect and remove bias in machine learning pipelines throughout the lifecycle of models. The AI Fairness 360 toolkit includes metrics for models and datasets to test bias and algorithms to mitigate this bias.

First we load the student mental health data and apply the label encoder to convert the strings to integer format.

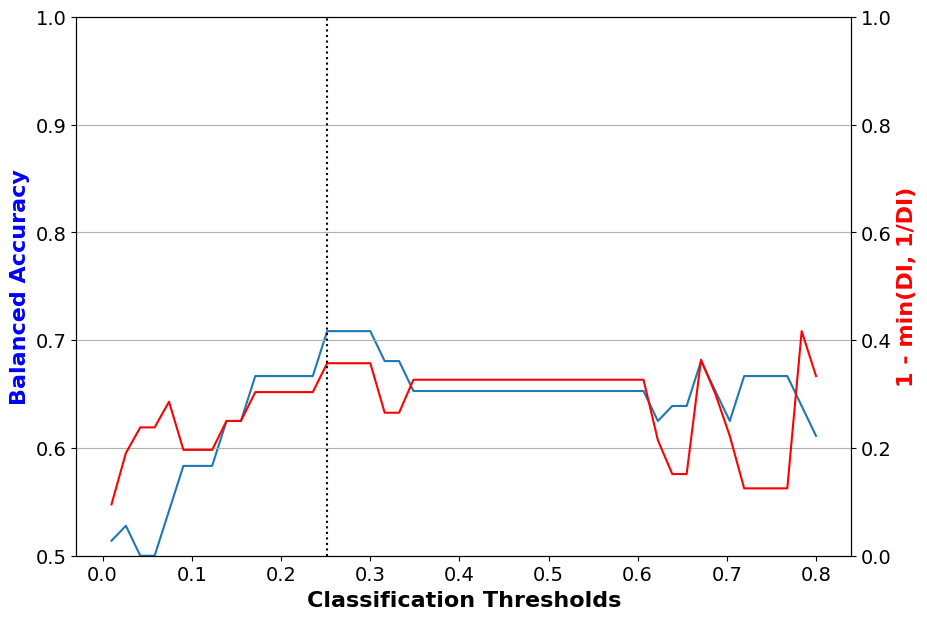


After that we create an AIF360 dataset using this dataframe where we specify the target column and the protected classes for which fairness is desired. We also define the privileged attributes which are considered privileged from a fairness perspective. We make use of the Disparate impact metric.

Disparate impact = (probability of favorable outcome for unprivileged instances / probability of favorable outcome for privileged instances)

After this we create the supplementary functions to test and plot the metrics and after that we actually build the pipeline using the logistic regression from sklearn.  
  


After evaluating the model and finding out the bias, we use the prejudice remover tool from the aif360 library to remove the bias from the dataset. We get astonishing results as follows.



## CODE:

## Installation

Pip install pandas sklearn 'aif360[all]' dataframe-image lime

## Data Loading

df = pd.read\_csv('/content/Student Mental health.csv')

## Data processing

le = preprocessing.LabelEncoder()

df = df.apply(le.fit\_transform)

df.drop(columns=['Timestamp'], inplace=True)

def get\_favourable1(stress\_level):

return stress\_level==0

DW = StandardDataset(

df,

"Do you have Depression?",

get\_favourable1,

["Choose your gender"],

[[0]])

(dataset\_train,

dataset\_val,

dataset\_test) = DW.split([0.5, 0.8], shuffle=True)

sens\_ind = 0

sens\_attr = dataset\_train.protected\_attribute\_names[sens\_ind]

unprivileged\_groups = [{sens\_attr: v} for v in

dataset\_train.unprivileged\_protected\_attributes[sens\_ind]]

privileged\_groups = [{sens\_attr: v} for v in

dataset\_train.privileged\_protected\_attributes[sens\_ind]]

## Supplementary functions

def test(dataset, model, thresh\_arr):

try:

# sklearn classifier

y\_val\_pred\_prob = model.predict\_proba(dataset.features)

pos\_ind = np.where(model.classes\_ == dataset.favorable\_label)[0][0]

except AttributeError:

# aif360 inprocessing algorithm

print('aif360 inprocessing algorithm')

y\_val\_pred\_prob = model.predict(dataset).scores

pos\_ind = 0

metric\_arrs = defaultdict(list)

for thresh in thresh\_arr:

y\_val\_pred = (y\_val\_pred\_prob[:, pos\_ind] > thresh).astype(np.float64)

dataset\_pred = dataset.copy()

dataset\_pred.labels = y\_val\_pred

metric = ClassificationMetric(

dataset, dataset\_pred,

unprivileged\_groups=unprivileged\_groups,

privileged\_groups=privileged\_groups)

# calculate the F1 score

metric\_arrs['bal\_acc'].append((metric.true\_positive\_rate()

+ metric.true\_negative\_rate()) / 2)

metric\_arrs['F1\_score'].append(metric.true\_positive\_rate() /

(metric.true\_positive\_rate() + (0.5 \*(metric.false\_positive\_rate() + metric.false\_negative\_rate()))))

metric\_arrs['FP Diff'].append(metric.false\_positive\_rate\_difference())

metric\_arrs['disp\_imp'].append(metric.disparate\_impact())

metric\_arrs['avg\_odds\_diff'].append(metric.average\_odds\_difference())

metric\_arrs['stat\_par\_diff'].append(metric.statistical\_parity\_difference())

metric\_arrs['eq\_opp\_diff'].append(metric.equal\_opportunity\_difference())

#metric\_arrs['theil\_ind'].append(metric.theil\_index())

return metric\_arrs

def plot(x, x\_name, y\_left, y\_left\_name, y\_right, y\_right\_name, filename=None):

fig, ax1 = plt.subplots(figsize=(10,7))

ax1.plot(x, y\_left)

ax1.set\_xlabel(x\_name, fontsize=16, fontweight='bold')

ax1.set\_ylabel(y\_left\_name, color='b', fontsize=16, fontweight='bold')

ax1.xaxis.set\_tick\_params(labelsize=14)

ax1.yaxis.set\_tick\_params(labelsize=14)

ax1.set\_ylim(0.5, 1)

ax2 = ax1.twinx()

ax2.plot(x, y\_right, color='r')

ax2.set\_ylabel(y\_right\_name, color='r', fontsize=16, fontweight='bold')

if 'DI' in y\_right\_name:

ax2.set\_ylim(0., 1)

else:

ax2.set\_ylim(-0.25, 1)

best\_ind = np.argmax(y\_left)

ax2.axvline(np.array(x)[best\_ind], color='k', linestyle=':')

ax2.yaxis.set\_tick\_params(labelsize=14)

ax2.grid(True)

if filename:

fig.savefig(filename)

def describe\_metrics(metrics, thresh\_arr):

best\_ind = np.argmax(metrics['bal\_acc'])

print("Threshold corresponding to Best balanced accuracy: {:6.4f}".format(thresh\_arr[best\_ind]))

print("Best balanced accuracy: {:6.4f}".format(metrics['bal\_acc'][best\_ind]))

# disp\_imp\_at\_best\_ind = np.abs(1 - np.array(metrics['disp\_imp']))[best\_ind]

disp\_imp\_at\_best\_ind = 1 - min(metrics['disp\_imp'][best\_ind], 1/metrics['disp\_imp'][best\_ind])

print("F1 score: {:6.4f}".format(metrics['F1\_score'][best\_ind]))

print("FP Diff: {:6.4f}".format(metrics['FP Diff'][best\_ind]))

print("Corresponding 1-min(DI, 1/DI) value: {:6.4f}".format(disp\_imp\_at\_best\_ind))

print("Corresponding average odds difference value: {:6.4f}".format(metrics['avg\_odds\_diff'][best\_ind]))

print("Corresponding statistical parity difference value: {:6.4f}".format(metrics['stat\_par\_diff'][best\_ind]))

print("Corresponding equal opportunity difference value: {:6.4f}".format(metrics['eq\_opp\_diff'][best\_ind]))

print('Corresponding disparate impact value: {:6.4f}'.format(metrics['disp\_imp'][best\_ind]))

## Model Building

dataset = dataset\_train

model = make\_pipeline(StandardScaler(),

LogisticRegression(solver='liblinear', random\_state=1))

fit\_params = {'logisticregression\_\_sample\_weight': dataset.instance\_weights}

lr\_dataset = model.fit(dataset.features, dataset.labels.ravel(), \*\*fit\_params)

## Model evaluation

thresh\_arr = np.linspace(0.1, 0.9, 50)

val\_metrics = test(dataset=dataset\_val,

model=lr\_dataset,

thresh\_arr=thresh\_arr)

lr\_orig\_best\_ind = np.argmax(val\_metrics['bal\_acc'])

## Plotting

disp\_imp = np.array(val\_metrics['disp\_imp'])

disp\_imp\_err = 1 - np.minimum(disp\_imp, 1/disp\_imp)

plot(thresh\_arr, 'Classification Thresholds',

val\_metrics['bal\_acc'], 'Balanced Accuracy',

disp\_imp\_err, '1 - min(DI, 1/DI)',

filename='LR\_DI\_none.eps')

## Bias removal

model = PrejudiceRemover(sensitive\_attr=sens\_attr, eta=25.0)

pr\_orig\_scaler = StandardScaler()

dataset = dataset\_train.copy()

dataset.features = pr\_orig\_scaler.fit\_transform(dataset.features)

pr\_orig\_DW = model.fit(dataset)

## Model re-evaluation

thresh\_arr = np.linspace(0.01, 0.8, 50)

dataset = dataset\_val.copy()

dataset.features = pr\_orig\_scaler.transform(dataset.features)

val\_metrics = test(dataset=dataset,

model=pr\_orig\_DW,

thresh\_arr=thresh\_arr)

pr\_orig\_best\_ind = np.argmax(val\_metrics['bal\_acc'])

## Plotting

disp\_imp = np.array(val\_metrics['disp\_imp'])

disp\_imp\_err = 1 - np.minimum(disp\_imp, 1/disp\_imp)

plot(thresh\_arr, 'Classification Thresholds',

val\_metrics['bal\_acc'], 'Balanced Accuracy',

disp\_imp\_err, '1 - min(DI, 1/DI)',

filename='LR\_DI\_PR.eps')