Note: This notebook requires the Tensorflow enviornment

1 Executive Summary

In this notebook, I approach the problem of designing ML models that account for the natural tension between efficiency and fairness. More precisely, I consider and test a rich family of metrics and algorithms that have been introduced in the literature. I characterize the trade-off achieved between efficiency and fairness to show that we can build several models for different objectives and managerial decisions based on this trade-off.

2 Introduction

To depict the natural tension between efficiency and fairness, I will construct a set of models based on an artificially made dataset to identify where the biase is located and if so, how to mitigate the bias using some of the functionality of AI Fairness 360 [6].

3 Definition of fairness

There is not yet an accepted definition of “fairness,” and there seems to be a disconnect between what it means to be fair for an individual versus a population. There are some interesting proposals in the literature such as the one proposed by Dwork et al. "Individual fairness means that “similar individuals are treated similarly."

4 De-biasing techniques

Both fairness metrics and de-biasing techniques can be performed at various stages of the machine learning pipeline.

4.1 Pre-processing

Techniques: (Learning Fair Representation (LFR), Reweighing, Disparate Impact Remover, Optimized Preprocessing)

This is called pre-processing mitigation because it happens before the creation of the model so the bias is removed from the data before training the model.

4.2 In-processing

Techniques: (Adversarial Debiasing, Prejudice Remover Regularizer, ART Classifier)

the information about sensitive features is used to guide model training.

4.3 Post-processing

Techniques: (Calibrated Equality of Odds, Equality of Odds, Reject Option Classification)

The predictions of the model are adjusted to minimize some fairness metric. For example, one can reweigh predictions to make the prediction distribution for privileged and unprivileged group equal and hence minimize the equal opportunity metric. It lets you use all the tools you want and allows you to apply fixes post-hoc but it doesn’t work with many fairness metrics.

5 Metrics

1) Statistical Parity Difference,
In words, it is the difference between the probability that a random individual drawn from S ("protected” subset of the population) is labeled 1 and the probability that a random individual from the complement subset is labeled 1. It measures the difference that the majority and protected classes get a particular outcome. When that difference is small, the classifier is said to have “statistical parity”.

Example1: If 30% of normal-hair-colored people get loans, statistical parity requires roughly 30% of teals to also get loans,Example2: it must shows an equal probability for male and female applicants to have good predicted credit score.

Pr(Y=1|D=unprivileged)−Pr(Y=1|D=privileged)

2) Equal Opportunity Difference,
This metric is just a difference between the true positive rate (recall scores)of unprivileged group and the true positive rate of privileged group, so smaller difference values are better

.A value of 0 indicates equality of opportunity. TPR_{D=unprivileged}−TPR_{D=privileged}

3) Average Absolute Odds Difference,
This measure is using both false positive rate and true positive rate to calculate the bias. It's calculating the equality of odds with the next formula and it should be zero to be fair:

1/2 * (|FPR_{D=unprivileged} - FPR_{D=privileged} | + | TPR_{D=unprivileged} - TPR_{D=privileged}|)

4) Disparate Impact,

We use both probabities of a random individual drawn from unprivileged or privileged with a label of 1 but here it's a ratio. For the disparate impact it's 1 that we need.

Pr(Y=1|D=unprivileged)/Pr(Y=1|D=privileged)
Therefore, this is the ratio of probability of favorable outcomes between the unprivileged and privileged groups.
The industry standard is a four-fifths rule: if the unprivileged group receives a positive outcome less than 80% of their proportion of the privilege group, this is a disparate impact violation. However, we may decide to increase this for our business. The threshold can be 0.8 for example so that we deem the calculated value to be unfair.

5) Theil Index (α=1).

The most commonly used flavor of generalized entropy is the Theil index. Generalized entropy is calculated for each group and then compared. This method can be used to measure fairness not only at a group level but also at the individual level. It needs to be close to 0 to be fair.

6 Checking the Data, Identifying the Location of Biases

6.1 Importing the libriaries

In [85]:
```
# Data Manipulation and Visualization Libraries↔
```

In [86]:
```
# ML Libraries↔
```

In [87]:
```
# Metrics and techniques↔
```

In [88]:
```
# AIf360 Libraries↔
```

6.2 Loading the Data

In [89]:
```
dataset=pd.read_csv('Gsd.csv')
dataset.head(5)
```

Out[89]:

	Unnamed: 0	gender	found_job	age	class
0	0	male	0	56	B
1	1	male	0	29	C
2	2	male	0	64	C
3	3	female	0	63	B
4	4	male	0	64	A

Some information about the this dataset
Found_job (No = 0 Yes = 1)
Number of people who got jobs in Class A >B >C
Age of people who obtained a job is lower than those who did not get it
Men got more jobs than wemen

In [90]:
```
#Create a DataFrame object for our calculated scores
dfObj = pd.DataFrame(columns = ['Balanced_Accuracy','DI','SPD', 'Average_odds_difference',
                                'Equal_opportunity_difference', 'Theil_index'] ,
                     index=['Reweighing','Odds_equalizing','ROC','Prejuduce Remover'])
dfObj
```

Out[90]:

	Balanced_Accuracy	DI	SPD	Average_odds_difference	Equal_opportunity_difference	Theil_index
Reweighing	NaN	NaN	NaN	NaN	NaN	NaN
Odds_equalizing	NaN	NaN	NaN	NaN	NaN	NaN
ROC	NaN	NaN	NaN	NaN	NaN	NaN
Prejuduce Remover	NaN	NaN	NaN	NaN	NaN	NaN

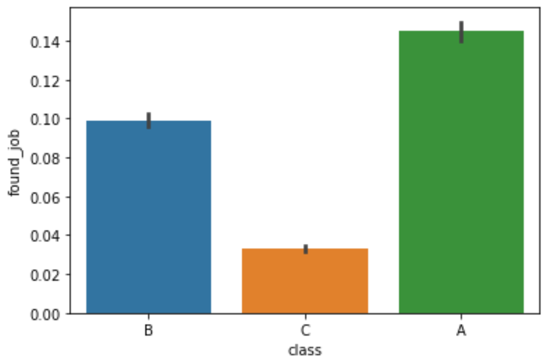
6.3 EDA

In [91]:
```
dataset['age_category'] = np.where(dataset['age'] > 65, 'Elderly',
                                   np.where(dataset['age'] < 28, 'Young', 'Adult'))
```

In [92]:
```
unprivileged_group = dataset[dataset['class'] == "C"][['gender','found_job','age','class', 'age_category']]
privileged_groupA = dataset[dataset['class'] == "A"][['gender','found_job','age','class', 'age_category']]
privileged_groupB = dataset[dataset['class'] == "B"][['gender','found_job','age','class','age_category']]
# combination of group a and B =AB
privileged_groupAB = privileged_groupA.append([privileged_groupB]).reset_index(drop=True)
```

In [93]:
```
sns.barplot(x=dataset["class"], y=dataset["found_job"])
```

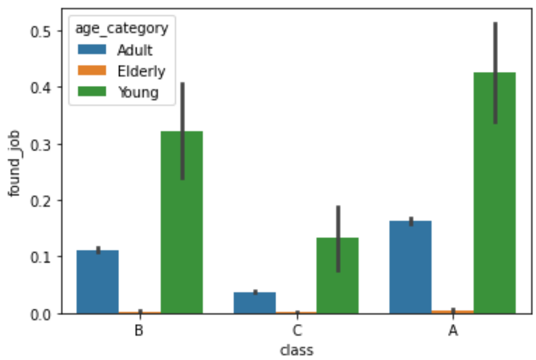
Out[93]: <AxesSubplot:xlabel='class', ylabel='found_job'>



People in Class A managed to find more jobs than Class B and C

In [94]:
```
sns.barplot(x=dataset["class"], y=dataset["found_job"], hue=dataset["age_category"])
```

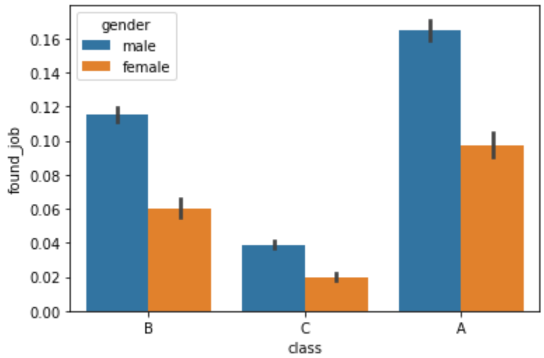
Out[94]: <AxesSubplot:xlabel='class', ylabel='found_job'>



It is clear that Within all the classes (A, B and C), young people have higher chances of getting jobs and the probability of old people getting a job is quite lower than Adult or young peole

In [95]:
```
sns.barplot(x=dataset["class"], y=dataset["found_job"], hue=dataset["gender"])
```

Out[95]: <AxesSubplot:xlabel='class', ylabel='found_job'>



It is also clear that within all the classes (A, B and C), men have had higher chance of getting the job than women

We can conclude that our data is biased with respect to three attributes which are mainly Classes, Gender and Age and they can be used as the grouping column. Let us consider the Disparate Impact Factor (DIF) and Statistical Parity Difference (SPD) as our Fairness metrics here. Therefore, we need to look into the ratio of probability of favorable outcomes between the unprivileged and privileged groups and their differences respectively.

Pr(Y=1|D=unprivileged)/Pr(Y=1|D=privileged)
We need to determined each attribute’s disparate impact, or the probability of a favorable outcome for unprivileged instances divided by the probability of a favorable outcome for privileged instances, after binarizing the attributes.

Pr(Y=1|D=unprivileged)−Pr(Y=1|D=privileged)

In [96]:
```
# A Function to calculate the probabilities of the favorable outcome in both groups↔
```

In [97]:
```
# If we consider "Class" as the Biased Attribute↔
```

```
probability of observing a record with a favorable outcome in class C is:0.033
probability of observing a record with a favorable outcome in class A is:0.145
probability of observing a record with a favorable outcome in class B is:0.099
*****************************************************
(DIF) Disparate Impact Factor of C/A:0.228
(DIF) Disparate Impact Factor of C/B:0.334
(SPD) Statistical Parity Difference of C-A:-0.112
(SPD) Statistical Parity Difference of C-B:-0.066
```

We can see that the calculated DIF values are so close to 0 and this confirms that the data is biased with respect to the "Class=C" attribute. SPD also is negative which shows that the class C is unpreviledged and the difference is larger when we compare it to class A.

In [98]:
```
# If we consider "Gender" as the Biased Attribute↔
```

```
probability of observing a record with a favorable outcome for women is:0.049
probability of observing a record with a favorable outcome for men is:0.09
*****************************************************
(DIF) Disparate Impact Factor of female/male:0.541
(SPD) Statistical Parity Difference of female-male:-0.041
```

Similar to what we have observed before, the calculated DIF values are low and this confirms that the data is biased with respect to the "Gender=female" attribute. SPD also is negative which shows that the female group is unpreviledged.

In [99]:
```
# If we consider "Age" as the Biased Attribute↔
```

```
probability of observing a record with a favorable outcome for young people is:0.269
probability of observing a record with a favorable outcome for old people is:0.002
*****************************************************
(DIF) Disparate Impact Factor of Elderly/Young:0.009
(SPD) Statistical Parity Difference of Elderly-Young:-0.266
```

Here also we can see that the dataset is biased with respect to the "age_category=Elderly" attributes. SPD also is negative which shows that the class "age_category=Elderly" is unpreviledged.

In adition to that, the order of DIF and SPD scores has the following patterns.. Age> Class> Gender

In [100]:
```
##Note1: The other two Metrics requires building a model on top of data
```

6.4 One-Hot Encoding and Scaling

In [101]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 6 columns):
 #   Column        Non-Null Count   Dtype 
---  ------        --------------   ----- 
 0   Unnamed: 0    100000 non-null  int64 
 1   gender        100000 non-null  object
 2   found_job     100000 non-null  int64 
 3   age           100000 non-null  int64 
 4   class         100000 non-null  object
 5   age_category  100000 non-null  object
dtypes: int64(3), object(3)
memory usage: 4.6+ MB
```

Categorical variables ('gender', 'class', 'age_category') are converted into a form that could be provided to ML algorithms to do a better job in prediction.

In [102]:
```
dataset_onehot = pd.concat([dataset[['found_job', 'age']],
                            pd.get_dummies(dataset[['gender', 'class', 'age_category']])], axis=1)
```

In [103]:
```
dataset_onehot= dataset_onehot.drop(["gender_male"], axis=1)
```

In [104]:
```
numerical_feature = ['age']
for feature in numerical_feature:
    #numpy.newaxis represents a new axis in numpy array
    fetched_val = dataset_onehot[feature].values[:, np.newaxis]
    scaler = MinMaxScaler().fit(fetched_val)
    dataset_onehot[feature] = scaler.transform(fetched_val)
```

In [105]:
```
dataset_onehot.head(5)
```

Out[105]:

	found_job	age	gender_female	class_A	class_B	class_C	age_category_Adult	age_category_Elderly	age_category_Young
0	0	0.457447	0	0	1	0	1	0	0
1	0	0.170213	0	0	0	1	1	0	0
2	0	0.542553	0	0	0	1	1	0	0
3	0	0.531915	1	0	1	0	1	0	0
4	0	0.542553	0	1	0	0	1	0	0

6.5 Building a Logistic Regression Model (LRM)

6.6 Pre-Prerocessing Techniques

6.6.1 Bias Mitigation with Reweighing

Reweighing [1] also is a preprocessing technique that Weights the examples in each (group, label) combination differently to ensure fairness before classification.This techniques does not change the label of the the original records.

• Every object X will be assigned a weight.
• the weight of an object will be the expected probability to see an instance with its sensitive attribute value and class given independence, divided by its observed probability.

(Step 1):Re-weight the {x,y} tuples in the training dataset so that cases where the protected attribute p predicts that the disadvantaged group will get a positive outcome are more highly weighted.
(Step 2):Train a classifier that makes use of these weights in its cost function or Alternately, they re-sampling the training data according to these weights and using a standard classifier.

In [106]:
```
# Let us load the required libraries
from aif360.algorithms.preprocessing import Reweighing
from aif360.metrics import BinaryLabelDatasetMetric
from aif360.metrics import ClassificationMetric
from sklearn.preprocessing import StandardScaler
from tqdm import tqdm
```

In [115]:
```
# Several implemented functions↔
```

In [108]:
```
# This function plot the threshould array, the calculated balanced accuracy and the disparity_index for↔
```

In [110]:
```
privileged_group = [{'gender_female': 0}]
unprivileged_group = [{'gender_female': 1}]
df= BinaryLabelDataset(df=dataset_onehot, label_names=['found_job'],
                       protected_attribute_names=['gender_female'])
# partition the data in to training, validation, and test datasets.
# Here first element denotes size for training set (0.5), second element denotes size for
#validation set (1-0.8 = 0.2) and difference between the two denotes size for test set(0.8-0.5 = 0.3).
df_trn, df_val, df_tst = df.split([0.5, 0.8], shuffle=True)
```

6.6.1.1 LRM on Biased Data

In [111]:
```
# caluclate the disparity_index on the training set↔
```

```
1-min(DI, 1/DI): 0.448
```

For our fairness benchmark, we require that 1 - min(DI, 1/DI) < 0.2, therefore, it is obvious that our original training set is biased.

In [112]:
```
# Now make a model and perform a logistic regression model on the original training data, we make the predic
# and return the model and its scale
lr_model_orig, lr_scale_orig = train_lr_model(df_trn)
```

In [113]:
```
# now we calculate the probabilities of the occurrence of each target value for every records of the validat
thresh_arr = np.linspace(0.01, 0.5, 100)
pred_prob = pred_prob_lr(scale=lr_scale_orig, model=lr_model_orig, dataset=df_val)
# pred_prob contains two types of probbailities one for class 0 (job_found=0) and one for class 1  (job_foun
```

In [116]:
```
#  Now we use the pred_prob and our validation set to find the followings:
# acc_metrics_orig: d dataframe consisting of the the best accuracy,  the best threshold, its index, and its
# bal_acc_arr_orig: the Accuracy of our model with respect to different classification threshoulds
# disp_imp_arr_orig: the Disparate Impact with respect to different classification threshoulds
# dataset_pred_labels_orig: The lables of the validation dataset for the last value of threshold (0.5)
acc_metrics, bal_acc_arr, disp_imp_arr, dataset_pred_labels, gfnr_arr, odds_diff_arr = \
get_best_cutoff(pred_prob=pred_prob, dataset=df_val)
```

```
100%|████████████████████████████████████████████████████████████████████| 100/100 [00:02<00:00, 35.16it/s]
```

In [117]:
```
# Let us have a look at the values within acc_metrics↔
```

Out[117]:

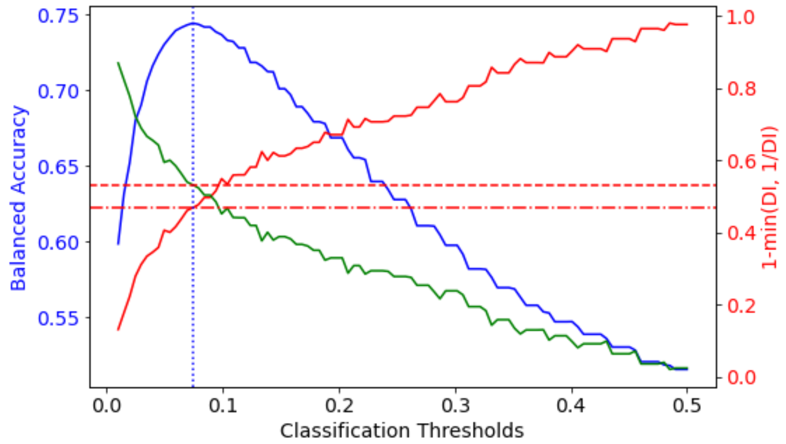
	0
thresh_arr_best_ind	13.000000
thresh_arr_best	0.074343
best_bal_acc	0.744459
disp_imp_at_best_bal_acc	0.531737
SPD_at_best_bal_acc	-0.176754

In [118]:
```
print('Threshold corresponding to best balanced accuracy:', acc_metrics.loc['thresh_arr_best', 0].round(3))
print('Best balanced accuracy:', acc_metrics.loc['best_bal_acc', 0].round(3))
print('1-min(DI, 1/DI):', get_disparity_index(acc_metrics.loc['disp_imp_at_best_bal_acc', 0]).round(3))
```

```
Threshold corresponding to best balanced accuracy: 0.074
Best balanced accuracy: 0.744
1-min(DI, 1/DI): 0.468
```

Obviously, the biased in our dataset has been propagated to the model which we built because the disparity index value is around 0.46 still which is bigger than 0.2.

In [119]:
```
# now let us plot the obtained accuracy versus fairness values for each classification threshold↔
```



The green curve shows the original calculated DI values for each classification threshold.
The red curv shows the converted DI values to 1-min(DI,1/DI).
The blue curve shows the accuracy of model for each for each classification threshold.
Obviously the patterns shows the model accuracy of 0.758345 in the presence of 0.46 as the DI index.

In [120]:
```
# Let us check our test data↔
```

In [121]:
```
print('Threshold corresponding to best balanced accuracy:', acc_metrics.loc['thresh_arr_best', 0].round(3))
print('Best balanced accuracy:', get_bal_acc(classified_metric_orig).round(3))
print('1-min(DI, 1/DI):', get_disparity_index(metric_pred_orig.disparate_impact()).round(3))
```

```
Threshold corresponding to best balanced accuracy: 0.074
Best balanced accuracy: 0.746
1-min(DI, 1/DI): 0.471
```

The DI index value also is not good on the test set. Now let us apply reweighting techniques and remove the biase.

6.6.1.2 LRM on Unbiased Data

In [122]:
```
# First we transform our training dataset
Reweighing_groups = Reweighing(unprivileged_group, privileged_group)
df_transf_trn = Reweighing_groups.fit_transform(df_trn)
```

In [123]:
```
metric_transf_trn = BinaryLabelDatasetMetric(df_trn, unprivileged_group, privileged_group)
print('1-min(DI, 1/DI):', get_disparity_index(metric_transf_trn.disparate_impact()).round(3))
```

```
1-min(DI, 1/DI): 0.448
```

As we showed before, the DI index for the original training data is 0.43. Now let us run the model on the the transformed training data

In [124]:
```
lr_transf, lr_scale_transf = train_lr_model(df_transf_trn)
```

In [125]:
```
pred_prob_transf = pred_prob_lr(scale=lr_scale_transf, model=lr_transf, dataset=df_val)
```

In [126]:
```
acc_metrics_transf, bal_acc_arr_transf, disp_imp_arr_transf, dataset_pred_labels_transf, gfnr_arr_transf, oc
get_best_bal_acc_cutoff(pred_prob=pred_prob_transf, dataset=df_val)
```

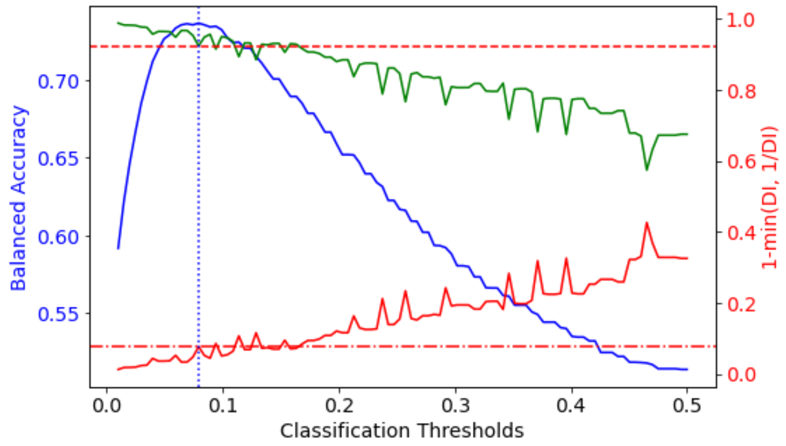
```
100%|██████████████████████████████████████████| 100/100 [00:02<00:00, 34.53it/s]
```

In [127]:
```
print('Threshold corresponding to best balanced accuracy:', acc_metrics_transf.loc['thresh_arr_best', 0].rou
print('Best balanced accuracy:', acc_metrics_transf.loc['best_bal_acc', 0].round(3))
print('1-min(DI, 1/DI):', get_disparity_index(acc_metrics_transf.loc['disp_imp_at_best_bal_acc', 0]).round(3
```

```
Threshold corresponding to best balanced accuracy: 0.079
Best balanced accuracy: 0.737
1-min(DI, 1/DI): 0.076
```

As we can see the calculated Index of DI this time shows a value less than 0.2 which is an accepted threshold for us.

In [129]:
```
#now let us plot the obtained accuracy versus fairness values for our transfered data and for each classific
```



As we can see, the calculated values of both DI and its Index are within an acceptance threshold. Less than 0.2 for the index and 0.85 for the original DI values.

In [130]:
```
# Now let us check our test data↔
```

In [131]:
```
print('Threshold corresponding to best balanced accuracy:', acc_metrics_transf.loc['thresh_arr_best', 0].rou
print('Best balanced accuracy:', get_bal_acc(classified_metric_transf).round(3))
print('1-min(DI, 1/DI):', get_disparity_index(metric_pred_transf.disparate_impact()).round(3))
```

```
Threshold corresponding to best balanced accuracy: 0.079
Best balanced accuracy: 0.745
1-min(DI, 1/DI): 0.101
```

Here also the DI index value (0.094)is within the thereshold. The accuracy of the model is 0.73 which is close to the accuracy on our validation set.

In [132]:
```
#let us save the result in our dfObj dataframe↔
```

Out[132]:

	Balanced_Accuracy	DI	SPD	Average_odds_difference	Equal_opportunity_difference	Theil_index
Reweighing	0.7238	0.899266	-0.0332602	-0.0162376	-0.020656	0.0739673
Odds_equalizing	NaN	NaN	NaN	NaN	NaN	NaN
ROC	NaN	NaN	NaN	NaN	NaN	NaN
Prejuduce Remover	NaN	NaN	NaN	NaN	NaN	NaN

6.6.1.3 Discussion over the findings:

My conclusion is that through reweighting technique we can give up a little bit of the model's accuracy in exchange for introducing more fairness to our model.

6.7 Post-Processing Techniques

6.7.1 Odds-equalizing post-processing algorithm[2]

Given two groups, equalized odds aims to ensure that no error rate disproportionately affects any group. In other words, both groups should have the same false-positive rate, and both groups should have the same false-negative rate.

We would like to achieve this definition of non-discrimination while maintaining calibrated probability estimates. However, achieving both of these goals is impossible. Therefore, we seek to maintain calibration while matching a single cost constraint. The equal cost constraint can be:

Equal false-negative rates between the two groups
Equal false-positive rates between the two groups
Equal weighted combination of the error rates between the two groups

6.7.1.1 Loading Libraries

In [133]:
```
from aif360.algorithms.postprocessing.calibrated_eq_odds_postprocessing import CalibratedEqOddsPostprocessir
from tqdm import tqdm
```

6.7.1.2 Loading the Data

In [134]:
```
# We will load our data
privileged_group = [{'gender_female': 0}]
unprivileged_group = [{'gender_female': 1}]
df= BinaryLabelDataset(df=dataset_onehot, label_names=['found_job'],
                       protected_attribute_names=['gender_female'])
df_trn, df_val, df_tst = df.split([0.5, 0.8], shuffle=True)
```

In [135]:
```
#  Let us look at some of the fairness metrics on our data↔
```

```
SPD on unprivileged and privileged groups (training set) = -0.040019
DI on unprivileged and privileged groups (training set) = 0.553075
SPD on unprivileged and privileged groups (validation set) = -0.040944
DI on unprivileged and privileged groups (validation set) = 0.555463
SPD on unprivileged and privileged groups (test set) = -0.044811
DI on unprivileged and privileged groups (test set) = 0.488704
```

The metrics shows that "Men" are the priviledged group and we are not close to our ideal values for acheiving a higher level of fairness.

6.7.1.3 Building a Logistic Regression Model

In [136]:
```
# first we build a logistic regression model on top of our training data
lr_model_orig, lr_scale_orig = train_lr_model(df_trn)
```

In [137]:
```
# then we get the predictions for our validation and test sets
pred_prob_val = pred_prob_lr(scale=lr_scale_orig, model=lr_model_orig, dataset=df_val)
pred_prob_test = pred_prob_lr(scale=lr_scale_orig, model=lr_model_orig, dataset=df_tst)
```

In [138]:
```
# the favorite outcome (1) is located at the index 1 of the calculated probabilities.
df_val_pred = df_val.copy(deepcopy=True)
df_test_pred= df_tst.copy(deepcopy=True)
df_val_pred.scores=pred_prob_val[:,1].reshape(-1,1)
df_test_pred.scores=pred_prob_test[:,1].reshape(-1,1)
```

In [139]:
```
class_thresh = 0.5

# now we need to make some adjustments to the labels of our new datasets which contains (as score values)
# the calcuclated probabilities by our LR model. First we set the threshold = 0.5.
# we think of the records which has the score value > this threshold as the records
# with favorable outcome(1) and the rests are adjusted with the unfovorable lable (0)
fav_inds = df_val_pred.scores >= class_thresh
df_val_pred.labels[fav_inds] = df_val_pred.favorable_label
df_val_pred.labels[~fav_inds] = df_val_pred.unfavorable_label

fav_inds = df_test_pred.scores >= class_thresh
df_test_pred.labels[fav_inds] = df_test_pred.favorable_label
df_test_pred.labels[~fav_inds] = df_test_pred.unfavorable_label
```

In [140]:
```
# Let us calculate some metrics ↔
```

```
In our training data
Difference in GFPR between unprivileged and privileged groups
-0.03410412385608905
Difference in GFNR between unprivileged and privileged groups
0.08055712274761961
In our test data
Difference in GFPR between unprivileged and privileged groups
-0.03287173581531995
Difference in GFNR between unprivileged and privileged groups
0.08921395741102656
```

6.7.1.4 Perform odds equalizing post processing on scores

In [141]:

```
cost_constraint = "weighted" # "fnr", "fpr", "weighted"
#random seed for calibrated equal odds prediction
randseed = 101

# Learn parameters to equalize odds and apply to create a new dataset
cpp = CalibratedEqOddsPostprocessing(privileged_groups = privileged_group,
                                     unprivileged_groups = unprivileged_group,
                                     cost_constraint=cost_constraint,
                                     seed=randseed)
cpp = cpp.fit(df_val, df_val_pred)
```

In [142]:

```
df_transf_val_pred = cpp.predict(df_val_pred)
df_transf_tst_pred = cpp.predict(df_test_pred)
```

In [143]:

```
# Let us calculate some metrics ↔
```

```
In our transformed validation data
Difference in GFPR between unprivileged and privileged groups
-0.006320779562812747
Difference in GFNR between unprivileged and privileged groups
0.05399792315680163
In our transformed test data
Difference in GFPR between unprivileged and privileged groups
-0.03742880390561432
Difference in GFNR between unprivileged and privileged groups
0.004611536657808335
```

The results show noticable reductions in the calculated metrics. But all we have done was for the standard classification threshold of 0.5 and we need to test it on a range of values. For this purpose we need to create a loop and stores the obtained results (before and after transformations) in different arrays.

In [144]:

```
# Now let us loop this through the following classification threshld np.linspace(0.01, 0.99, 25)↔
```

```
100%|████████████████████████████████████████████████████████████| 25/25 [00:28<00:00,  1.12s/it]
```

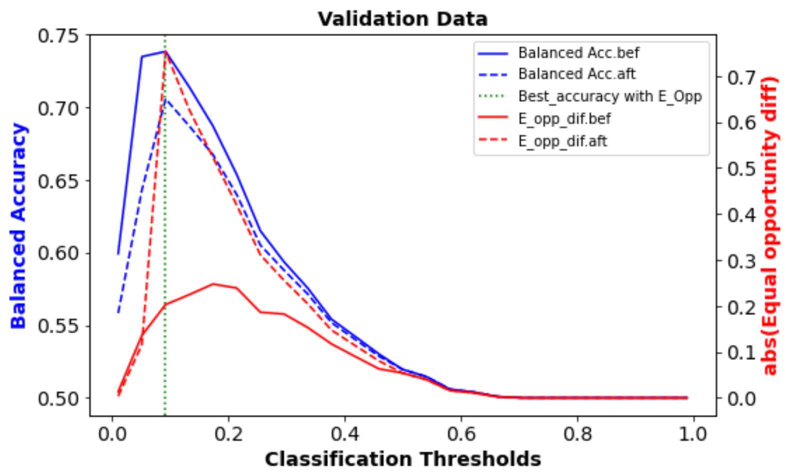
In [145]:

```
# Find threshold for the best transformed accuracy
thresh_arr_best_ind_val = np.where(aft_bal_acc_valid == np.max(aft_bal_acc_valid))[0][0]
thresh_arr_best_ind_tst = np.where(aft_bal_acc_test == np.max(aft_bal_acc_test))[0][0]
```

In [146]:

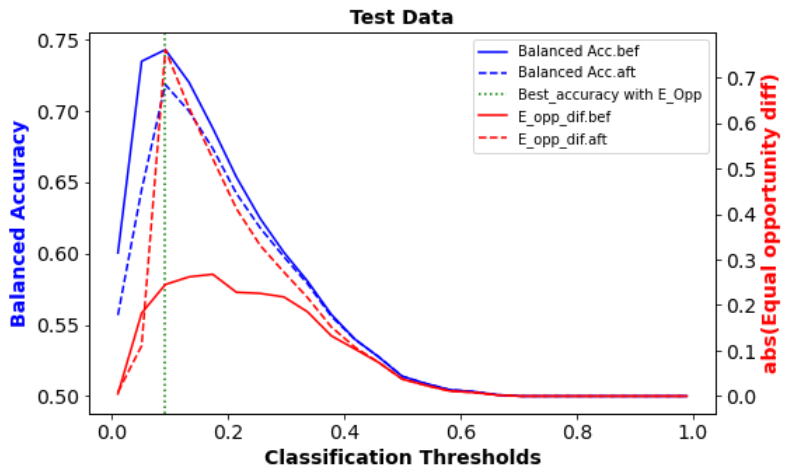
```
#Let us plot the results on the validation set↔
```

Out[146]: <matplotlib.legend.Legend at 0x14635fdc888>



In [147]:
```
1 #Let us plot the results on the test set↔
```

Out[147]: <matplotlib.legend.Legend at 0x14699ecd708>



In [148]:
```
1 #Let us save the result in our dfObj dataframe↔
```

Out[148]:

	Balanced_Accuracy	DI	SPD	Average_odds_difference	Equal_opportunity_difference	Theil_index
Reweighing	0.7238	0.899266	-0.0332602	-0.0162376	-0.020656	0.0739673
Odds_equalizing	0.718298	0	-0.321543	-0.520065	-0.760781	0.0743887
ROC	NaN	NaN	NaN	NaN	NaN	NaN
Prejuduce Remover	NaN	NaN	NaN	NaN	NaN	NaN

6.7.1.5 Discussion over findings:

What we observe here is that we can maintain the balanced aacuracy around a desired value (here 70 to 75%) but simultaniously acheive lower level of discrimination among our priviledged and unpreviledged groups.

6.7.2 Reject Option Classification (ROC)

In this approach[3], the assumption is that most discrimination occurs when a model is least certain of the prediction i.e. around the decision boundary (classification threshold). Thus by exploiting the low confidence region of a classifier for discrimination reduction and rejecting its predictions, we can reduce the bias in model predictions. For example, with a classification threshold of 0.5, if the model prediction is 0.81 or 0.1, we would consider the model certain of its prediction but for 0.51 or 0.49, the model is not certain about the chosen category. In ROC, for model predictions with the highest uncertainty around the decision boundary, when the favorable outcome is given to the privileged group or the unfavorable outcome is given to the unprivileged, we modify them.

6.7.2.1 Loading Libraries

In [162]:
```
from aif360.algorithms.postprocessing.reject_option_classification\
        import RejectOptionClassification
from common_utils import compute_metrics
```

6.7.2.2 Loading the Data

In [150]:

```
privileged_group = [{'gender_female': 0}]
unprivileged_group = [{'gender_female': 1}]
df= BinaryLabelDataset(df=dataset_onehot, label_names=['found_job'],
                       protected_attribute_names=['gender_female'])
df_trn, df_val, df_tst = df.split([0.5, 0.8], shuffle=True)
```

6.7.2.3 Building a Logistic Regression Model

In [151]:

```
# At first, we build a model on top of our training data
lr_model_orig, lr_scale_orig = train_lr_model(df_trn)
```

In [152]:

```
# We get the predictions (probability of belonging to the target groups) for our validation and test sets
pred_prob_val = pred_prob_lr(scale=lr_scale_orig, model=lr_model_orig, dataset=df_val)
pred_prob_test = pred_prob_lr(scale=lr_scale_orig, model=lr_model_orig, dataset=df_tst)
```

In [153]:

```
# the favorite outcome (1) is located at the index 1 of the calculated probabilities.
df_val_pred = df_val.copy(deepcopy=True)
df_test_pred= df_tst.copy(deepcopy=True)
df_val_pred.scores=pred_prob_val[:,1].reshape(-1,1)
df_test_pred.scores=pred_prob_test[:,1].reshape(-1,1)
```

In [154]:

```
# We need to find out the threshold at which the balanced accuracy is the highest
thresh_arr = np.linspace(0.01, 0.99, 100)
acc_metrics_val, bal_acc_arr_val, disp_imp_arr_val, dataset_pred_labels_val, gfnr_arr_val, odds_diff_arr_val
get_best_bal_acc_cutoff(pred_prob=pred_prob_val, dataset=df_val)
```

```
100%|██████████████████████████████████████████████████| 100/100 [00:02<00:00, 36.57it/s]
```

In [155]:

```
# Let us look at the returned dataframe out of our analysis↔
```

Out[155]:

	0
thresh_arr_best_ind	7.000000
thresh_arr_best	0.079293
best_bal_acc	0.745217
disp_imp_at_best_bal_acc	0.522875
SPD_at_best_bal_acc	-0.171031

In [156]:

```
# We only need two peices of information in this dataframe
print("Best balanced accuracy (without any fairness constraints) = %.4f" % acc_metrics_val.loc['best_bal_acc
print("Optimal classification threshold (without any fairness constraints) = %.4f" % acc_metrics_val.loc['th
```

```
Best balanced accuracy (without any fairness constraints) = 0.7452
Optimal classification threshold (without any fairness constraints) = 0.0793
```

The result shows that the best balanced accuracy can be reached with the classification threhsold equals to 0.079

6.7.2.4 Estimate optimal parameters for the ROC method

In [157]:

```
metric_name = "Statistical parity difference"
# Upper and lower bound on the fairness metric used
metric_ub = 0.1
metric_lb = -0.1
#random seed for calibrated equal odds prediction
np.random.seed(1)
# metric_name should be one of the allowed metrics
allowed_metrics = ["Statistical parity difference",
                   "Average odds difference",
                   "Equal opportunity difference"]
ROC = RejectOptionClassification(unprivileged_groups=unprivileged_group,
                                 privileged_groups=privileged_group,
                                 low_class_thresh=0.01, high_class_thresh=0.99,
                                 num_class_thresh=100, num_ROC_margin=50,
                                 metric_name=metric_name,
                                 metric_ub=metric_ub, metric_lb=metric_lb)

# We will fit the ROC on our original vaidation data and the one with the predicted labels
ROC = ROC.fit(df_val, df_val_pred)
```

In [160]:

```
print("Optimal classification threshold (with fairness constraints) = %.4f" % ROC.classification_threshold)
print("Optimal ROC margin = %.4f" % ROC.ROC_margin)
```

```
Optimal classification threshold (with fairness constraints) = 0.0694
Optimal ROC margin = 0.0099
```

This result shows that the optimal classification threshold in the presence of our fairness constraints is lower that what we discovered before (0.07).

In [163]:

```
# now we need to make some adjustments to the labels of our new datasets which contains (as score values)
# the calcuclated probabilities by our LR model. Now that we know which threshold gives us
# the highest accuracy, we think of the records which has the score value > this threshold as the records
# with favorable outcome(1) and the rests are adjusted with the unfovorable lable (0)
fav_inds = df_val_pred.scores > acc_metrics_val.loc['thresh_arr_best', 0]
df_val_pred.labels[fav_inds] = df_val_pred.favorable_label
df_val_pred.labels[~fav_inds] = df_val_pred.unfavorable_label

# Let us look at the some fairness metrics here
metric_valid_bef = compute_metrics(df_val, df_val_pred,
                unprivileged_group, privileged_group)
```

```
Balanced accuracy = 0.7452
Statistical parity difference = -0.1710
Disparate impact = 0.5229
Average odds difference = -0.1807
Equal opportunity difference = -0.2137
Theil index = 0.0727
```

The metrics shows that "Men" are the priviledged group and we are not close to our ideal values for acheiving a higher level of fairness.

In [164]:

```
# Now we transform our new validation set with the help of ROC
df_transf_val_pred = ROC.predict(df_val_pred)

# Let us look at the some fairness metrics here
metric_valid_aft = compute_metrics(df_val, df_transf_val_pred,
                unprivileged_group, privileged_group)
```

```
Balanced accuracy = 0.7431
Statistical parity difference = -0.0972
Disparate impact = 0.7290
Average odds difference = -0.0947
Equal opportunity difference = -0.1144
Theil index = 0.0729
```

First of all, the drope in the accuracy in insignifanct. Second, the calculated values our of the fainess metrics shows a higher level of fairness.

In [165]:

```
# Let us repeat the same steps for our test set↔
```

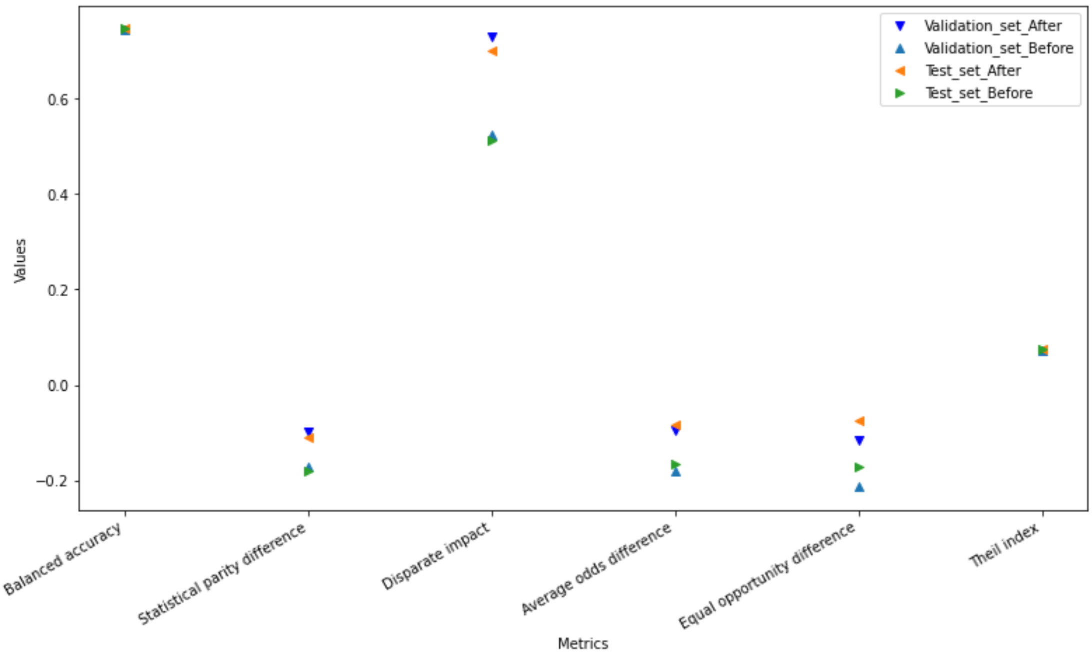
```
Balanced accuracy = 0.7467
Statistical parity difference = -0.1791
Disparate impact = 0.5116
Average odds difference = -0.1660
Equal opportunity difference = -0.1727
Theil index = 0.0738
```

In [166]:

```
# Let us repeat the same steps for our transformed test set↔
```

```
Balanced accuracy = 0.7459
Statistical parity difference = -0.1098
Disparate impact = 0.7005
Average odds difference = -0.0833
Equal opportunity difference = -0.0749
Theil index = 0.0737
```

In [167]:
```
1 #Let us plot the results↔
```



First of all, the drope in the accuracy in insignifanct (they are all around 73%). Second, the calculated values out of the painess metrics shows improvemnets in level of fairness. DI is becomming closer to 1, SPD are becomming close to 0 after transformations.

In [168]:
```
1 #Let us save the result in our dfObj dataframe↔
```

Out[168]:

	Balanced_Accuracy	DI	SPD	Average_odds_difference	Equal_opportunity_difference	Theil_index
Reweighing	0.7238	0.899266	-0.0332602	-0.0162376	-0.020656	0.0739673
Odds_equalizing	0.718298	0	-0.321543	-0.520065	-0.760781	0.0743887
ROC	0.745939	0.700504	-0.109793	-0.0833034	-0.0749049	0.073652
Prejuduce Remover	NaN	NaN	NaN	NaN	NaN	NaN

6.8 In-Processing Techniques

6.8.1 Prejudice Remover Regularizer

This method [4, 5] is focused on classification algorithms and utilizes regularization called a prejudice remover. The prejudice remover actually utilizes two regularizers: the first is a standard regularizer that removes overfitting of the model, whereas the second computes the minimum of the prejudice index defined as a function of the overall data population, the underprivileged class (or sensitive class), and the privileged class (nonsensitive class).

indirect prejudice
the dependency between a objective Y and a sensitive feature S
from the information theoretic perspective the mutual information between Y and S is non-zero
from the viewpoint of privacy-preservation We can expect the leakage of sensitive information when an objective variable is known
This method adds two fairness regularizers (a mathematical constraint to ensure fairness in the model) to a logistic regression model. One regularizer models the mutual information between S (sensitive info) and Y (target label) so as to make Y independent from S. The other regularizer

6.8.1.1 Loading Libraries

In [169]:
```
from aif360.algorithms.inprocessing import PrejudiceRemover
from aif360.metrics import BinaryLabelDatasetMetric, ClassificationMetric
```

6.8.1.2 Loading the Data

In [170]:
```
privileged_group = [{'gender_female': 0}]
unprivileged_group = [{'gender_female': 1}]
df= BinaryLabelDataset(df=dataset_onehot, label_names=['found_job'],
                       protected_attribute_names=['gender_female'])
df_trn, df_val, df_tst = df.split([0.5, 0.8], shuffle=True)
```

In [171]:
```
metric_val = BinaryLabelDatasetMetric(df_val,
                                             unprivileged_groups=unprivileged_group,
                                             privileged_groups=privileged_group)
print("Train set: Difference in mean outcomes between unprivileged and privileged groups = %f" % metric_val.
```

```
Train set: Difference in mean outcomes between unprivileged and privileged groups = -0.040742
```

In [172]:
```
# Eta is a regularization parameter. The lager value more enforces fairness.
debiased_model = PrejudiceRemover(sensitive_attr="gender_female", eta = 25.0)
debiased_model.fit(df_trn)
```

Out[172]:
```
<aif360.algorithms.inprocessing.prejudice_remover.PrejudiceRemover at 0x146b17540c8>
```

In [173]:
```
df_deb_val_pred = debiased_model.predict(df_val)
df_deb_tst_pred = debiased_model.predict(df_tst)
```

In [174]:
```
metric_debiasing_val_pred = BinaryLabelDatasetMetric(df_deb_val_pred,
                                             unprivileged_groups=unprivileged_group,
                                             privileged_groups=privileged_group)
print("Train set: Difference in mean outcomes between unprivileged and privileged groups = %f" % metric_debi
```

```
Train set: Difference in mean outcomes between unprivileged and privileged groups = -0.008240
```

In [175]:
```
metric_deb_val = compute_metrics(df_val, df_deb_val_pred,
                unprivileged_group, privileged_group)
```

```
Balanced accuracy = 0.5167
Statistical parity difference = -0.0082
Disparate impact = 0.0000
Average odds difference = -0.0249
Equal opportunity difference = -0.0452
Theil index = 0.0797
```

In [176]:
```
metric_deb_test = compute_metrics(df_tst, df_deb_tst_pred,
                unprivileged_group, privileged_group)
```

```
Balanced accuracy = 0.5163
Statistical parity difference = -0.0079
Disparate impact = 0.0000
Average odds difference = -0.0244
Equal opportunity difference = -0.0443
Theil index = 0.0768
```

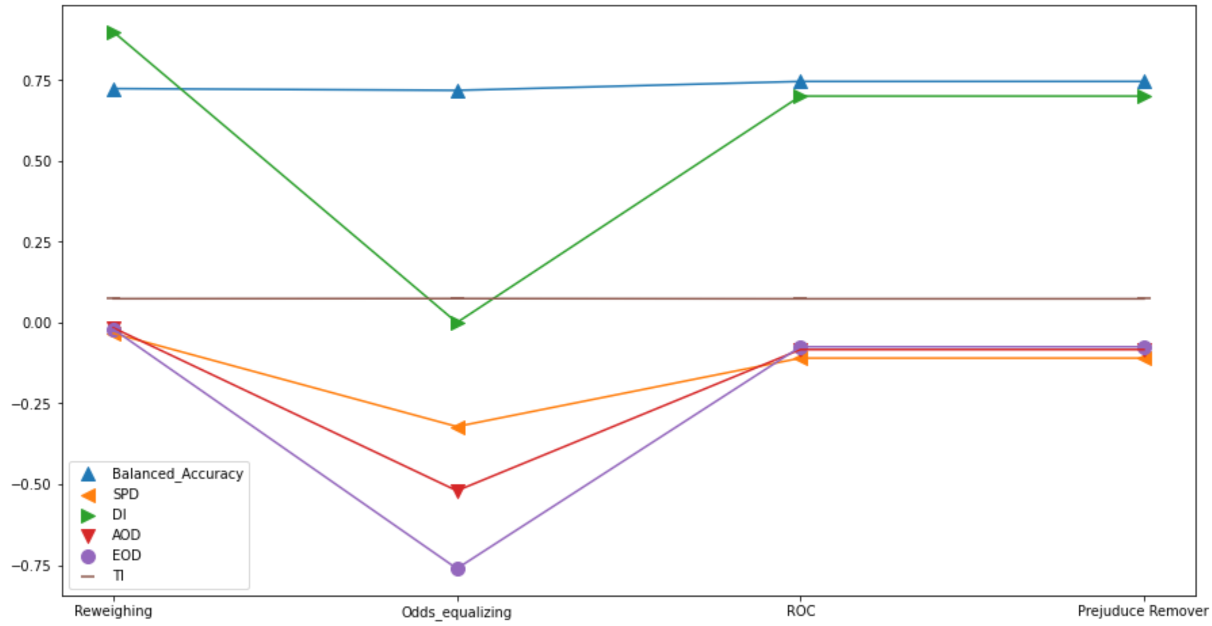
In [177]:
```
#let us save the result in our dfObj dataframe↔
```

Out[177]:

	Balanced_Accuracy	DI	SPD	Average_odds_difference	Equal_opportunity_difference	Theil_index
Reweighing	0.7238	0.899266	-0.0332602	-0.0162376	-0.020656	0.0739673
Odds_equalizing	0.718298	0	-0.321543	-0.520065	-0.760781	0.0743887
ROC	0.745939	0.700504	-0.109793	-0.0833034	-0.0749049	0.073652
Prejuduce Remover	0.745939	0.700504	-0.109793	-0.0833034	-0.0749049	0.073652

In [178]:
```
1 #Let plot compare and plot our findings↔
```

Out[178]: <matplotlib.legend.Legend at 0x146b144ec08>



First of all, the calculated balanced accuracy values are almost identical in all the methods.
Reweighting as a preprocessing algorithm has produced a better Disparate Impact and Statistical
Parity Difference values comparing to other In_ and Post_rocessong methods. The perfomance of Odds-Equalizing
method is not very good and it might be due to seeting the cost_constraint = "weighted". We have to repeat and test
this algorithm with other constraints.

References:

[1] https://link.springer.com/content/pdf/10.1007/s10115-011-0463-8.pdf (https://link.springer.com/content/pdf/10.1007/s10115-011-0463-8.pdf)
[2] https://papers.nips.cc/paper/7151-on-fairness-and-calibration.pdf (https://papers.nips.cc/paper/7151-on-fairness-and-calibration.pdf)
[3] https://mine.kaust.edu.sa/Documents/papers/ICDM_2012.pdf (https://mine.kaust.edu.sa/Documents/papers/ICDM_2012.pdf)
[4] https://link.springer.com/chapter/10.1007/978-3-642-33486-3_3 (https://link.springer.com/chapter/10.1007/978-3-642-33486-3_3)
[5] http://www.kamishima.net/archive/2011-ws-icdm_padm-HN.pdf (http://www.kamishima.net/archive/2011-ws-icdm_padm-HN.pdf)
[6] https://github.com/Trusted-AI/AIF360/tree/master/examples (https://github.com/Trusted-AI/AIF360/tree/master/examples)

In [ ]:
```
1
```